Advanced Query Mechanisms for Biological Databases *

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

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

Abstract

Existing query interfaces for biological databases are either based on fixed forms or textual query languages. Users of a fixed form-based query interface are limited to performing some pre-defined queries providing a fixed view of the underlying database, while users of a free text query language-based interface have to understand the underlying data models, specific query languages and application schemas in order to formulate queries. Further, operations on application-specific complex data (e.g., DNA sequences, proteins), which are usually provided by a variety of software packages with their own format requirements and peculiarities, are not available as part of, nor integrated with biological query interfaces.

In this paper, we describe generic tools that provide powerful and flexible support for interactively exploring biological databases in a uniform and consistent way, that is via common data models, formats, and notations, in the framework of the Object-Protocol Model (OPM). These tools include (i) a Java graphical query construction tool with support for automatic generation of Web query forms that can be either used for further specifying conditions, or can be saved and customized; (ii) query processors for interpreting and executing queries that may involve complex, application-specific, objects, and that could span multiple heterogeneous databases and file systems; and (iii) utilities for automatic generation of HTML pages containing query results, that can be browsed using a Web browser. These tools avoid the restrictions imposed by traditional fixed-form query interfaces, while providing users with simple and intuitive facilities for formulating ad hoc queries across heterogeneous databases, without the need to understand the underlying data models and query languages.

1 Introduction

An increasing number of biological databases are providing publically accessible query interfaces. For example, archival databases such as the Mouse Genome Database (MGD) at the Jackson Laboratory, Genome Database (GDB) [7] at Johns Hopkins School of Medicine, the Genome Sequence Database (GSDB) at National Center for Genome Resources[8], and Genbank at the National Center for Biotechnology Information [15], can all be queried via Web based interfaces.

Exploring data in biological databases involves examining the structure (metadata) of the databases, browsing and querying the databases, interpreting the results of queries, and processing and viewing application-specific data types, such as protein and DNA sequences, using special data type-specific operations, such as sequence comparison, structure comparison, and protein structure visualization. These operations are often available as individual software packages with their own

*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-40340.

input and output formats. (e.g., BLAST and FASTA for DNA sequence comparison, DALI for structure comparison, RasMol for viewing macromolecules).

In order to support querying and data exploration, biological databases must offer facilities for easy formulation of queries and interpretation of query results as well as support for seamless manipulation of application-specific data. The study of [12] showed that most archival databases provide limited query support. Their Web based query interfaces, for example, have a fixed structure involving a predetermined set of database components (e.g., tables, classes), and a predetermined set of attributes for each database component. These interfaces are based on canned queries that can be parameterized on certain values and may allow users to have some control over conditions or set the values used in the conditions. The main limitation of fixed-form interfaces is that they conform to and provide only some predetermined view of the underlying databases. Furthermore, fixed-form interfaces may need to be changed whenever the underlying database changes or new features need to be supported. Some Web interfaces provide the ability to specify queries over a database using free-form textual query languages, while not providing proper metadata support, so that most users are not able to make use of these facilities. Even worse, users may specify semantically incorrect queries without any mechanism to provide help in detecting such queries. Further, biological query interfaces do not provide support for manipulating application-specific data.

In this paper, we describe tools that provide advanced query mechanisms for biological databases in the context of an object model, the *Object-Protocol Model* (OPM). These tools are used in conjunction with the OPM retrofitting tools for constructing OPM views of existing relational and structured file databases [3].

The OPM query tools are generic (*schema-driven*), that is, they are driven by the metadata associated with the underlying database and allow ad-hoc queries to be constructed using graphical, Web based interfaces. Query construction is based on the premise that users are familiar with Web forms, and therefore follows a two stage approach: (1) a query tree is constructed by graphically browsing the object schema of the underlying database and iteratively selecting classes and attributes of interest; and (2) a Web form is generated automatically from the query tree, which can be used for further specifying conditions or can be saved and customized.

The query tools generate queries in an object-oriented query language, OPM-QL, which are then processed using OPM query translators. The query translators generate equivalent, possibly semantically optimized, queries for the underlying relational database [4] or flat file system.

Querying support for complex (application-specific) objects is provided via OPM Application-Specific Data Types (ASDTs) and methods. ASDTs are supported on top of existing relational DBMSs, such as Oracle 8 and Sybase 11, but may also take advantage of the advanced features of the emerging *object-relational* DBMSs, such as the Oracle 8 and Informix Universal Servers which provide mechanisms for incorporating ASDTs and their associated methods into the DBMS.

The remainder of this paper is organized as follows. In section 2 we briefly overview our framework for exploring databases on the Web. The query construction tools together with the techniques used for implementing them are described in section 3. Section 4 discusses query processing and query result browsing. Section 5 describes related work and gives concluding remarks.

2 Background

Our approach for exploring biological databases is based on an object model, the Object-Protocol Model (OPM). An object data model such as OPM can be used to represent heterogeneous databases in a uniform, abstract (system-independent), and consistent way. In addition, OPM provides extensive schema documentation facilities in the form of descriptions, examples and user-

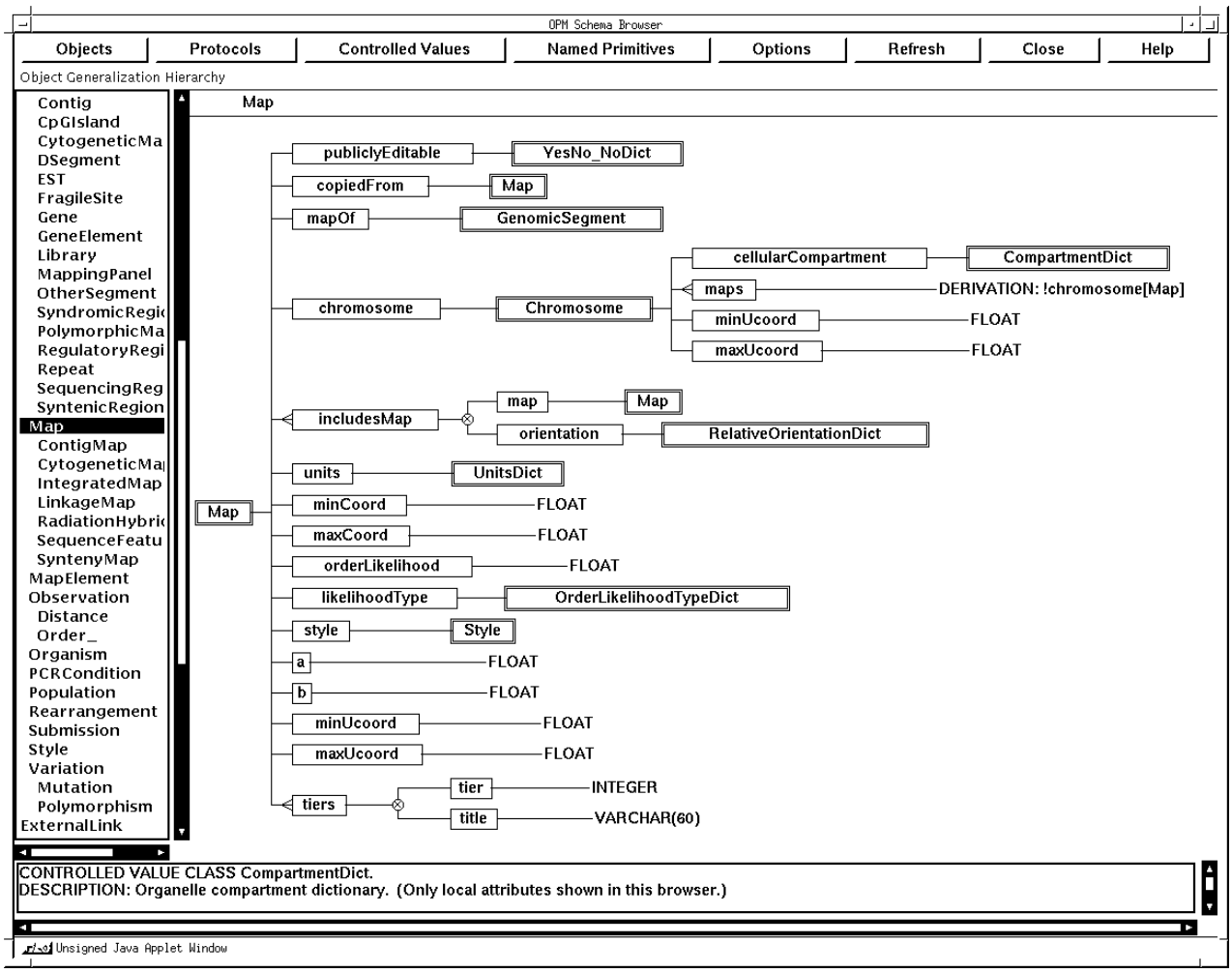

Figure 1: Browsing Classes using the OPM Schema Browser

specified properties. A variety of existing OPM tools provide facilities for developing and accessing databases, for constructing OPM views on top of existing databases and files, for generating alternative schema representations, and for querying heterogeneous databases through uniform OPM views. We briefly review the main constructs of OPM below; OPM is described in detail in [2].

2.1 Basic Object-Protocol Model Constructs

OPM is a data model whose object part closely resembles the ODMG standard for object-oriented data models [13]. 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 be single-valued, set-valued or list-valued. 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 contains an example of a part of the OPM schema for the Genome Database (GDB)[1] represented in a diagrammatic notation and browsed using the Java based OPM Schema Browser. This example contains several OPM classes, such as `Map` and `Chromosome`; attributes `includesMap` of `Map` and `maps` of `Chromosome` are set-valued, while attributes `mapOf` and `units` of `Map` are single valued; attribute `includesMap` is a tuple attribute consisting of two simple attributes, `map` and `orientation`; attribute `chromosome` is an abstract attribute taking values from class `Chromosome`, while attribute `minCoord` 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.

2.2 Advanced Object-Protocol Model Constructs

OPM has been extended with a new construct, the Application-Specific Data Type (ASDT). ASDTs allow modeling complex data types, such DNA sequences, maps, and gels. For example, the image of a gel could be represented as follows:

```
APPLICATION SPECIFIC DATA TYPE GelImage
    DATATYPE: OPM_BLOB/IMAGE
      STORED: EXTERNAL
      DESCRIPTION: "A binary image type"
      PROPERTIES: "imagetype" "gif"
      METHOD display
           SIGNATURE: "void display(string)"
           LANGUAGE: "java"
           CODE: "/me/code/java/asdts/gelimage.java"
           DESCRIPTION: "displays gel image"
```

In this example, the second line of the GelImage ASDT definition specifies that it's data type is OPM_BLOB/IMAGE, which is an OPM primitive value class. The third line specifies that this type will be stored outside the database. ASDT properties can be used either by the application program or by the methods that operate on the ASDT. For instance, `"imagetype" "tiff"` means that a `"tiff"`-capable viewer needs to be used to display this image.

ASDT methods are also specified. They must have an implementation, in some programming language, that the OPM tools will use in order to execute the method. The signature part of the method specification defines the return type and the formal parameters of the method. Service methods such as display, return void. The code field (optional) specifies the file location where the code of the method resides. The language and description parts are self-explanatory.

2.3 OPM Database Development and Retrofitting Tools

OPM schemas can be specified using a graphical OPM schema editor or a regular text editor. OPM schema translators automatically generate complete definitions for databases implemented with commercial relational database management systems (DBMSs), such as Sybase and Oracle.

[1] `http://gdbwww.gdb.org/gdb/schema.html`

A *mapping dictionary* records the correspondences between the classes and attributes of an OPM schema and the underlying relational tables.

Existing relational or structured file databases that have not been developed using OPM tools, can be retrofitted with an OPM schema using the OPM Retrofitting tools [3]. The retrofitting tools can be used for constructing multiple OPM views for a single (OPM or non-OPM) database; these tools also generate a mapping dictionaries as described above.

3 Query Interfaces

Existing public query interfaces for biological databases are usually available on the Web and are either form-based or textual query language-based. Form-based query interfaces usually have a fixed structure involving a predetermined set of components (e.g., tables, classes, attributes), and provide a limited number of options, such as specifying the values for certain attributes. Such interfaces are based on predetermined or "canned" queries, possibly parameterized on certain values, and provide a single fixed view of the database. They may not reflect the structure of the underlying database, since the list of attributes or fields retrieved by a canned query may involve only a subset of the attributes and fields in the underlying database, and the classes or tables accessed may be only a subset of those in the underlying database. In spite of their restrictions, form-based query interfaces are both easy to implement and use.

Attempts to provide *ad hoc* query facilities usually rely on allowing the user to specify queries using some ad hoc textual query language. Users of such interfaces must have expert knowledge of the underlying query language, data model and database schema. The facilities offered by such interfaces depend on the query language supported by the underlying DBMS: different DBMSs support different flavors of query languages based on different or even identical data models. For example numerous databases are developed using relational DBMSs which support different versions of SQL. Specifying ad-hoc SQL queries therefore requires non-trivial knowledge of the structure and manipulation of relational databases, and of the particular relational DBMS and dialect of SQL being used. This is often beyond the ability of general users.

In this section we describe the OPM query tools. Our strategy is to provide Web based ad-hoc query specification capabilities via a schema-driven Java graphical interface, coupled with dynamically generated HTML query forms. Queries specified using this tool are subsequently passed to OPM query translators for commercial relational DBMSs or structured flat-file systems. Used in conjunction with native or retrofitted OPM schemas, the OPM query tools provide a uniform and intuitive query interface on top of heterogeneous database systems, and allow users to formulate queries while browsing database schemas. Query results are organized in HTML pages with hyperlinks to related objects and metadata definitions in order to facilitate Web browsing.

Formulating Basic Queries. The OPM query interface is designed to provide an extension to the ubiquitous Web (HTML) query forms that users are already familiar with, so that using this tool has a very short learning curve. This is necessary since, in our experience, many users are not computer scientists and are not used to complex graphical query interfaces. Instead of providing predefined query forms, the OPM Web query tool provides support for constructing a query tree by selecting classes and attributes of interest using a graphical user interface, for generating dynamically HTML query forms based on this query tree. Further query condition specification can be carried out by filling in these query forms.

Query construction with our interface is illustrated by the example shown in Figure 2. The graphical user interface shown in the top window in Figure 2 is used to construct a *query tree*.

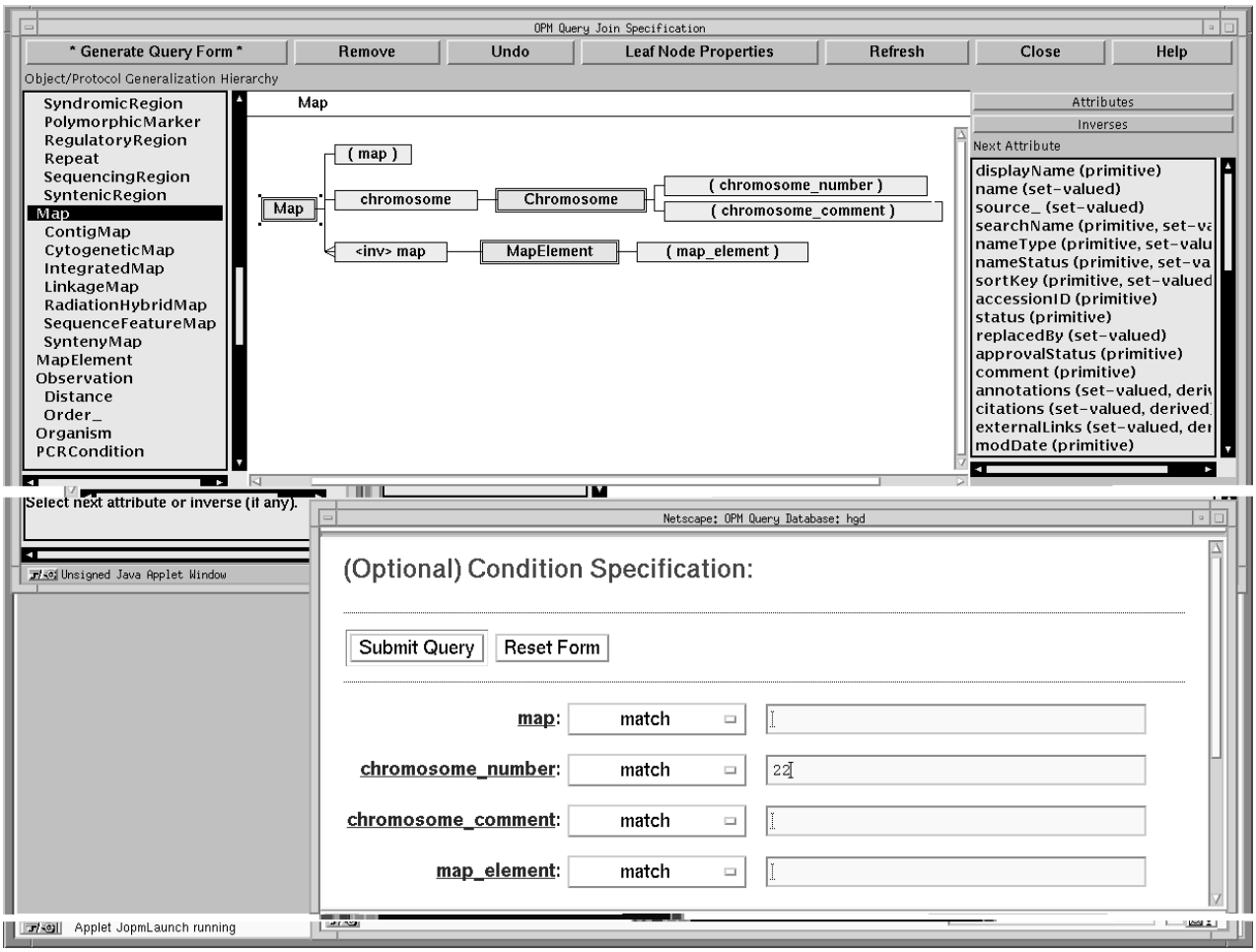

Figure 2: Constructing Web Query Forms with the the OPM Web Query Tool

First a *root* class is selected from the Class list-box on the left of the interface window, and then the query tree is expanded by recursively selecting classes and attributes that are involved in the conditions and/or output of the query. Attributes associated with a selected class, C, are selected from an *Attributes* list-box; inverse attributes associated with C, that is abstract attributes that take values from C, can be selected from an *Inverse* attribute list-box.

In the example shown in Figure 2 class `Map` is selected as the root of the query tree. Attributes, such as `displayName` and `chromosome` are then added to the tree by selecting (clicking on) a class in the diagrammatic representation of the tree and then selecting attributes associated with that class from the Attributes or Inverses list-boxes on the right-hand side. A selected attribute is added to the query tree displayed in the main window. Primitive attributes, such as `displayName` and `comment` form the leaves of the tree. For abstract attributes, such as `chromosome`, a value class, such as `Chromosome`, must also be selected and displayed in the main window; attributes of new selected classes may then be selected in turn, such as attributes `displayName` and `comment` of class `Chromosome` in Figure 2.

The selection process is repeated until all classes and attributes of interest have been added to the query tree. Attributes can be renamed (e.g., attributes `displayName` of `Map` and `displayName` of `Chromosome` in the upper side window of figure 2 are renamed `map` and `chromosome_number`, respectively, in the generated form) and specified as part of the query output, condition or both.

Once the query tree is completed, an HTML query form can be generated (see the form in the lower half of Figure 2). This form is used for specifying conditions on attributes in the familiar Query-by-Example mode (e.g., `chromosome_number` = "22"). 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 submitted to the database via the OPM Query Translator (see section 5), or the form can be saved as an HTML file, that can then 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 the OPM Query Language (OPM-QL), an object-oriented query language similar to OQL, the ODMG standard for object-oriented query languages [4]. For example, the query constructed in figure 2 is expressed by the following OPM query:

```
SELECT map = M.displayName, chromosome_number = C.displayName,
       cytogenetic_marker = CM.displayName, comment = C.comment
FROM   M IN Map,  C IN M.chromosome[Chromosome],
       CM IN M.!map [CytogeneticMarker]
WHERE  C.displayName = "22";
```

The OPM query interface does not support the specification of the full range of queries that can be expressed in the OPM textual query language (OPM-QL). For example, OPM queries involving several root classes or complex conditions involving parentheses and attribute comparisons, are not supported in the current version of the tool, mainly because of the difficulty in supporting the graphical specification of such queries in a simple and intuitive way. However, the OPM query interface supports the specification of queries that are substantially more complex than those underlying fixed forms, while allowing access to all the elements of the underlying database.

Examples of databases that can be accessed using this query tool are available at the OPM Web site[2]. Of particular interest is the Primary Database of the German Genome Resource Center[3], where the OPM Web query tool is available for constructing queries directly, but has also been used for setting up Web query forms[4].

Formulating Queries with ASDTs. OPM queries involving ASDTs may invoke methods for those ASDTs. For example, the following query can be used to display the image of the gel with identity `gel_000111`:

```
SELECT X.gelId, X.gelImage.display()
FROM X in Gel
WHERE X.gelId = "gel_000111"
```

The following query displays only part of `gel_000111`s image:

```
SELECT X.gelId, X.gelImage.crop(0,0,200,400).display()
FROM X in Gel
WHERE X.gelId = "gel_000111"
```

where crop is a method that returns a piece of the image of specified coordinates.

[2] `http://gizmo.lbl.gov/jopmDemo/demoDbs.html`
[3] `http://www.rzpd.de/cgi-bin/public_login`
[4] `http://www.rzpd.de/object_form.html`

Implementation Details. The OPM query interface is implemented using a combination of the Java programming language and HTML forms. OPM schema and mapping information is loaded into the Java interface once it is started. Query trees are drawn using Java Abstract Windowing Toolkit (AWT) based on the schema information and user selections.

After a query is constructed, a click on the "Generate Query From" button in the query interface results in sending a URL to a Common Gateway Interface (CGI) script. The URL includes database and user information, and the query tree encoded in a text string. Based on this information, an HTML form is generated and is available for further condition specification.

There were several reasons for choosing HTML forms for specifying conditions, rather than implementing the forms in Java or specifying conditions directly on the graphical query tree: most users are familiar and comfortable with Web query forms, and are often reluctant to learn new query paradigms; formatting forms in current versions of Java would have been less visually appealing and would have required more development work than using HTML; and, once generated, the HTML forms can be edited, incorporated into other Web pages, and used independently without needing to download the Java applets.

In this implementation, upon generation of the HTML form control is turned over to a Web browser (e.g., Netscape) which opens a second window containing the form. Subsequent actions in this window follow normal Web browser behavior, outside the control of the Java query constructor applet.

4 Query Processing

The OPM Web query tools described in section 3 provide a front end to a system for processing queries and analyzing query results. In this section we will describe the remaining tools which comprise this system, and our implementation strategies for these tools.

Query Translators. Queries in an OPM framework are evaluated using *OPM Query Translators*. An OPM query translator takes queries specified in the OPM Query Language (OPM-QL), and generates corresponding queries in the query language supported by the underlying DBMS or file system. The results of these queries are then structured and returned using OPM specific data structures.

The 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 query translators for efficiency.

OPM query translators have been developed for commercial relational DBMSs, such as Sybase and Oracle [4], and for structured flat-file systems. The relational OPM query translators generate queries in the dialect of SQL supported by the underlying DBMS, and employ DBMS-specific C/C++ APIs for database access. In general, the SQL queries generated by these translators are considerably more complex than the OPM-QL queries, since a single OPM class is usually represented by several distinct relational tables.

The flat-file OPM query translator employs SRS (Sequence Retrieval System), a system for parsing and indexing structured flat files developed at the European Molecular Biology Laboratory (EMBL), initially for accessing molecular biology data repositories [6]. The SRS query language has limited power, in that conditions are restricted to simple comparisons of indexed attributes with constants, so that OPM queries can be only partly translated into SRS queries. Consequently

post-processing of SRS query results is carried out locally using an OPM query engine originally developed for evaluating multidatabase queries [11].

Application programs can interact with the OPM query translators either via C++ or CORBA APIs (each query translator shares a common API), or by calling the query translators as Unix command-line programs. The later can be done using Perl or Unix shell scripts, using temporary files for passing OPM-QL queries and query results.

Processing Queries with ASDTs. If an ASDT method appears only in the SELECT clause of an OPM query, then the query is sent to the underlying DBMS through the OPM query translator, and the result is then directed to an ASDT method server. For example, the second OPM query above will retrieve a gels id and a handle to its image data, as a first step, then the method display will be called to present the image.

If a method appears in the WHERE clause, then a local OPM Query Processor must be used evaluating the query. The OPM Query Processor will start by rewriting the query into method-free parts that can be evaluated by the underlying DBMS, together with method calls and further conditions necessary to complete the evaluation of the query. The Query Processor then evaluates the query by evaluating the method-free sub-query using a Query Translator, then performing any method calls based on the results of this sub-query, and finally performing any further local computations or condition evaluations involving the results of the method calls, in order to complete the query. The OPM Query Processor is based on a client-server architecture, with Query Translator servers being used to evaluate the method-free sub-queries, and ASDT-servers being used to evaluate method-calls.

Implementation Details. We initially used the ubiquitous Command Gateway Interface (CGI) for implementing interfaces between the OPM Web query tools and the Query Translators and Processor. CGI was found to be easy to use and maintain given our multiple language (Java and C++) programming environment. CGI is a natural choice for dynamically creating HTML pages: Java is used for implementing the front (query construction) ends, and C++ (with a Perl wrapper) is used at the CGI back end. However, since a standard CGI call requires the creation of a new Unix process, this alternative could be inefficient. In our experience, the response time for CGI calls on the Web is acceptable, and slow responses are usually caused by slow queries or queries with large results, rather than by delays caused by the invocation of Unix processes.

We are currently developing CORBA based interfaces for application communication. CORBA is limited in the data structures that can be passed between applications, and consequently requires extraneous conversions between data structures used in applications and those that can be communicated. Further, despite the C++ mapping defined in the CORBA 2.0 standard, CORBA implementations remain vendor specific, so that a CORBA implementation of the query tools would be tied to a particular CORBA product. Nevertheless CORBA simplities the task of building client-server systems, and makes such issues as implementing languages and platforms, or locations of servers transparent to a client process. We have used CORBA for implementing the communication between the OPM Query Processor and the Query Translator and ASDT servers.

Browsing Query Results. Results of OPM queries submitted using the Web query forms described in section 3 are returned in HTML format. The query results are presented as a table, in order to provide a concise summary of the data retrieved (see the top window in Figure 3) The columns of the table correspond to the fields of the query form, or equivalently, the leaves of the

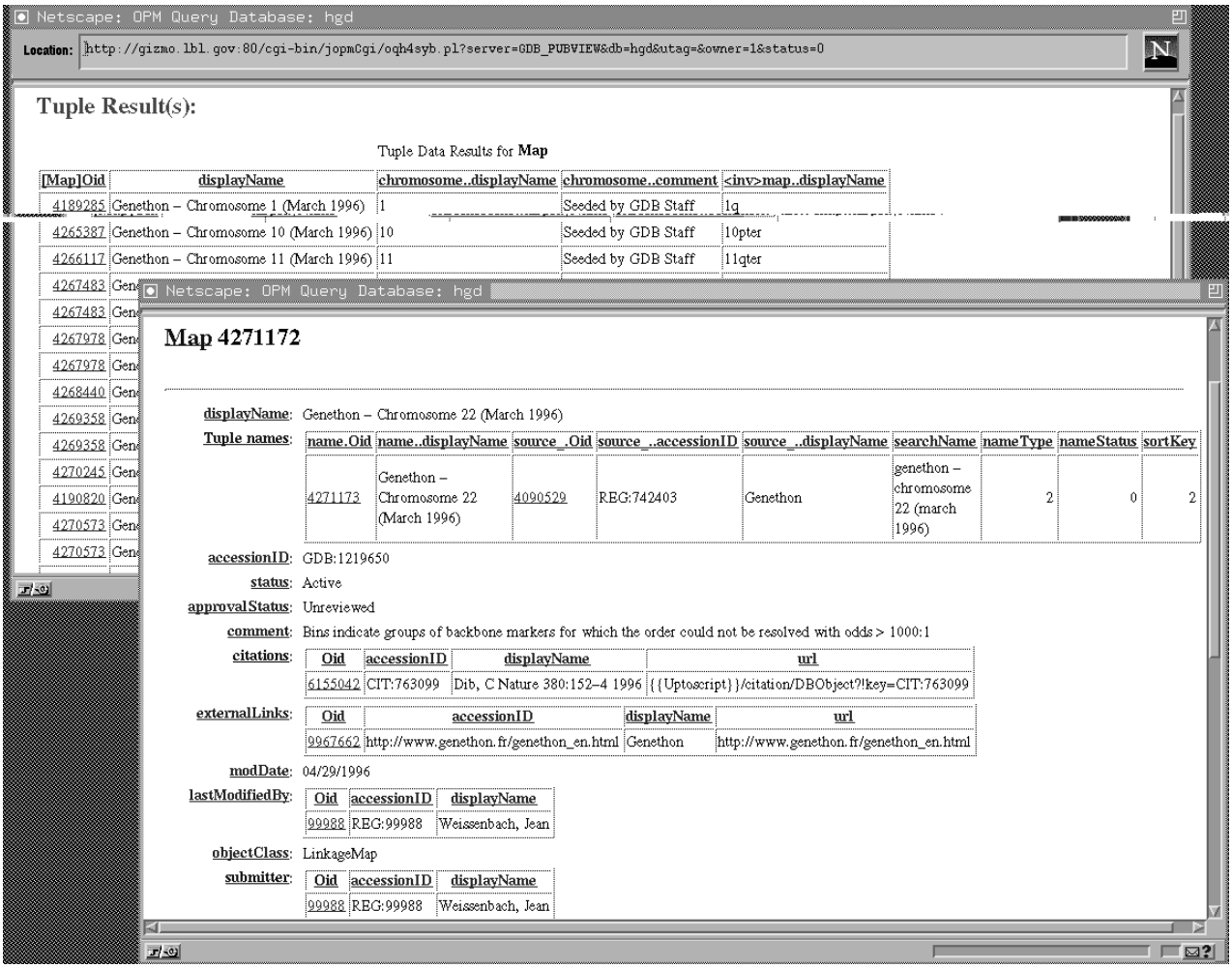

Figure 3: HTML pages displaying query results

original query tree. Clicking on a column label reveals the full definition of the corresponding field, including the description of the relevant attributes in the OPM schema.

The query result table contains a column of object identifiers for the root objects in the query tree (`Map` objects in Figure 3). Selecting (clicking on) an object identifier results in displaying the values of all the attributes of the corresponding object, also represented in HTML format.

The lower window of Figure 3 shows a `Map` instance in object form. Single-valued primitive attributes, such as `accessionID` and `status` are represented simply by their values. Multi-valued or tuple attributes and abstract attributes are represented using HTML tables. The columns of a table representing a tuple attribute consist of the component attributes of the tuple attribute. The columns of a table representing an abstract attribute consist of the *representative attributes* of the value class associated with the abstract attributes, as specified in the OPM schema. For example attribute `citations` of class `Map` takes values from class `Citation`, whose representative attributes are `accessionID`, `displayName` and `url`. Any object identifier included in a query result can be selected (clicked on) in order to display the values of all its attributes. In this way the instances of a database can be browsed interactively, starting from the objects retrieved by the initial query. In order to help interpreting the query results, each label is linked to a description for the corresponding attribute or class.

HTML pages containing query results are dynamically generated using the same CGI script that generated the query form, this time with a message "get tuples" or "get objects". Similarly HTML pages representing schema information are generated by sending the CGI script the message "show metadata".

Query results are returned and displayed at once (up to a cut-off value), instead of showing n objects at a time. This approach cannot be avoided: OPM views are built on top of relational databases or structured flat-file systems, where query evaluation usually involves joining several tables, and/or may be followed by post processing of the query results, so that selecting "n objects at a time" cannot be pursued. A cut-off value is set in order prevent loading too many objects, which could potentially crash a Web browser.

5 Concluding Remarks

We have described in this paper a suite of tools that provide advanced querying mechanisms for biological databases in the framework of the Object-Protocol Model (OPM).

A large amount of work exists in the area of graphical query interfaces to databases. Many vendors offer graphical query interfaces, such as Access and Paradox, to relational DBMSs. However these interfaces do not support object-oriented views of the underlying databases, or access to non-relational databases. Various graphical query interfaces for object-oriented or semantic databases have been developed as part of research projects; a survey of such interfaces is provided in [1]. A variety of paradigms for formulating queries and browsing results underlies these interfaces and each interface is based on a trade off between expressivity and ease-of-use.

The paradigm underlying our query tools, of constructing query trees and generating Web based query forms, as well as the ability of these tools to be used for accessing databases on the Web is, as far as we know, unique. Our tools provide a natural and relatively simple, yet powerful, extension to an already well known interface, the Web form. This strategy has proved successful in gaining acceptance for our tools in a large community of non-expert users of scientific (molecular biology and physics) databases. Further, our query tools can be used both for directly accessing databases and for constructing and customizing (fixed-form Web) database interfaces.

Work on generic Web-based query tools, that is tools which are not tied to a specific database schema or underlying DBMS, is much more limited. The Genera system developed by Letovsky [10] allows HTML query forms to be generated automatically from an object-oriented (OPM) schema. Genera provided both the inspiration and the incentive for our query tools which address the problem of each form corresponding to exactly one class and having a predefined structure, by supporting the dynamic construction and generation of forms spanning multiple classes.

The MOBIE system, developed at Stanford as part of the TSIMMIS project [9], provides support for browsing object-oriented query results using hyperlinks for navigating complex, deeply nested data-structures. MOBIE does not address the problem of formulating queries on the Web and is designed to support the TSIMMIS data model of *semi-structured* data, that is, data without a schema. Consequently, this system does not provide support for interpreting and understanding query results, for example, via links to schemas formand database documentation.

We plan to extend our query tools in two areas. First, we are experimenting with alternative visual paradigms suggested by our users for formulating queries and browsing data. For example, we have built a prototype Java tool that allows both query construction and data browsing through the familiar metaphor of files and folders. We also continue to extend our existing query tools in order to enhance their query construction and interpretation capabilities.

References

[1] Batini, C. *et al.*, Visual Query Systems, *Tech. Rep. 04.91 U. di Roma, March 1991.*

[2] 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.

[3] Chen, I.A., Kosky, A.S., Markowitz, V.M., and Szeto, E. Constructing and Maintaining Scientific Database Views, *Proc. of the 9th Int. Conference on Scientific and Statistical Database Management*, 1997.

[4] Chen, I.A., Kosky, A., Markowitz, V.M., and Szeto, E., The OPM Query Language and Translator, Technical Report LBL-33706, 1996; `http://gizmo.lbl.gov/opm.html`.

[5] *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`.

[6] 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`.

[7] 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://gdbgeneral.gdb.org/gdb/schema.html`.

[8] Genome Sequence DataBase (GSDB) 1.0. The National Center for Genome Resources, August 1996; `http://www.ncgr.org/gsdb/gsdb.html`.

[9] Hammer, J., Aranha, R., and Ireland, K., Browsing Object Databases Through the Web, Technical Report 237, Stanford University, October, 1996.

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

[11] Markowitz, V.M., Chen, I.A., and Kosky, A., Exploring Heterogeneous Biological Databases: Tools and Applications. *Proc. of the 6th International Conference on Extending Database Technology*, 1998.

[12] Markowitz, V.M., Chen, I.A., Kosky, A., and Ernest Szeto, Facilities for Exploring Molecular Biology Databases on the Web: A Comparative Study. In *Pacific Symposium on Biocomputing '97*, Altman & al (eds), World Scientific, 1997, pp. 256-267.

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

[14] 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`.

[15] Shuler, G.D., Epstein, J.A., Ohkawa, H., Kans, J.A. Entrez. In *Methods in Enzymology*, (R. Doolittle, ed.). Academic Press, Inc. In press. See also `http://www3.ncbi.nlm.nih.gov/Entrez/`.