

OBJECT-PROTOCOL MODEL DATA MANAGEMENT TOOLS `97

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

*Data Management Research and Development Group
Lawrence Berkeley National Laboratory, Berkeley, CA 94720**

Introduction

The development of the Object-Protocol Model (OPM) and OPM data management tools started in 1992. The main motivation for developing OPM at the time was the need to provide data management support for large scale DNA sequencing laboratories. We designed the *protocol class* as a construct for modeling sequencing, as well as other scientific, experiments. Since object-based data models were well suited for modeling the complex data underlying the scientific and traditional database applications targeted by OPM, we decided to incorporate the protocol class construct into the framework of an object data model [2].

The development of the OPM data management tools can be split into three stages. Initially, the OPM data management tools aimed only at providing

* Current affiliation: BIOINFORMATICS SYSTEMS, GENE LOGIC, INC., 2001 Center Str., Suite 600, Berkeley, CA 94704.

Email: {vmmarkowitz, ichen, anthony, szeto} @ genelogic.com

wrapper facilities for developing and querying individual databases implemented with commercial relational database management systems (DBMSs). The OPM Database Development and Database Query tools provided such facilities, first for Sybase and later Oracle, both widely used for implementing large production biological databases. These tools were employed for developing and maintaining several biological databases, such as version 6 of the Genome Data Base (GDB)¹ at Johns Hopkins School of Medicine in Baltimore, and the Primary Database of the German Human Genome Resource Center (RZPD)² in Berlin, Germany.

Next, the OPM Retrofitting tools were developed in order to provide support for constructing OPM views for databases that were not originally developed using the OPM tools, and for providing the infrastructure required for using the OPM Database Query tools to access these databases. We developed retrofitting tools for relational DBMSs and applied them to databases such as the Genome Sequence Database (GSDB)³ at the National Center for Genome Resources (NCGR). More recently, we have developed retrofitting tools for structured flat files such as GenBank.⁴

Finally, we took advantage of the ability to build uniform OPM views on top of diverse databases by developing the OPM Multidatabase Tools for querying and exploring multiple heterogeneous databases via native OPM schemas or retrofitted OPM views. These tools have been applied to the construction of a Molecular Biology Database Federation that includes GDB, GSDB, and GenBank.

In the remainder of this paper we will briefly overview the OPM data management tools and will discuss the experience gained in the past five years of developing and applying these tools to scientific database applications. Details of these tools and their underlying methodologies can be found in the OPM papers listed as references; most of these papers are available on the Web at <http://gizmo.lbl.gov/opm.html>.

The OPM project is currently at a crossroads. The OPM tools were developed by members of the Data Management Research and Development Group at Lawrence Berkeley National Laboratory, with funding from the Office of Biological and Environmental Research and the Mathematical, Information, and Computational Sciences Division of the US Department of Energy. In September 1997 the developers of the OPM tools joined Gene Logic Inc., forming its Bioinformatics Systems division, where the next generation of OPM

¹ <http://gdbwww.gdb.org/>

² <http://www.rzpd.de/>

³ <http://www.ncgr.org/gsdb/>

⁴ <http://www.ncbi.nlm.nih.gov/>

data management tools will be developed. The future versions of the OPM tools will include enhancements of existing facilities, such as more powerful Web-based interfaces, as well as new facilities, such as mechanisms for integrating data management and analytical tools, and providing support for database evolution.

The Object-Protocol Model

The Object-Protocol Model (OPM) is the result of incorporating constructs for modeling scientific experiments (protocols) into an object data model. We will briefly review the main features of OPM below; details can be found in [1].

OPM is a data model whose object part is closely related to the ODMG standard for object-oriented data models [10]. Objects in OPM are uniquely identified by object identifiers, are qualified by attributes, and are classified into classes. Classes can be organized in subclass-superclass hierarchies, where multiple inheritance in such hierarchies is supported.

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.

Part of an OPM schema for GDB 6, viewed using the Java-based OPM Schema Browser, is shown in Figure 1, where the class `Map` is shown together with its attributes. For example, `copiedFrom`, `mapOf` and `chromosome` are abstract attributes with value classes `Map`, `GenomicSegment` and `Chromosome` respectively, while `minCoord` and `maxCoord` are primitive attributes, and `includesMap` is a tuple attribute with components `map` and `orientation`.

OPM supports the specification of derived attributes using derivation rules involving arithmetic expressions, aggregate functions (min, max, sum, avg, count) and attribute composition. In Figure 1, for example, attribute `maps` of class `Chromosome` is a derived attribute defined as the inverse of attribute `chromosome` of class `Map`. OPM also supports derived subclasses and derived superclasses. A derived subclass is defined as a subclass of one or more derived or non-derived object classes with an optional derivation condition. A derived superclass is defined as a union of two or more object classes.

In addition to object classes, OPM supports a protocol class construct for modeling scientific experiments. Similar to object classes, protocol classes have

class names, optional class descriptions, identifiers, and are associated with attributes. Protocol modeling is characterized by the recursive specification (expansion) of generic protocols in terms of alternative subprotocols, sequences of subprotocols, and optional subprotocols. In addition to regular attributes, a protocol class can be associated with special input and output attributes that

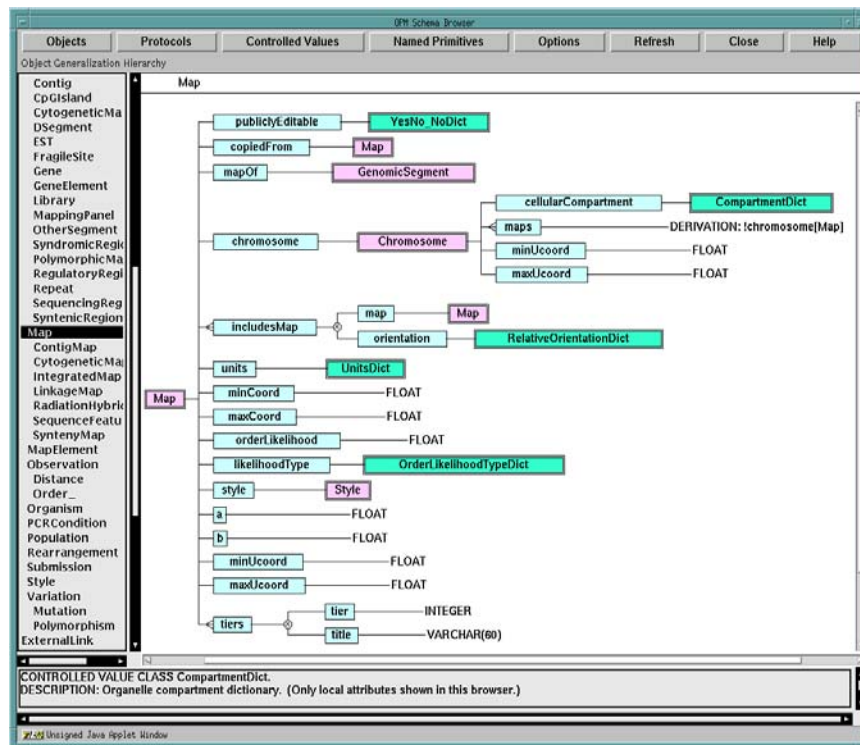


Figure 1: Part of the OPM Schema for GDB 6.

represent input and output data regarding the experiment modeled by the protocol class, and express input-output connections with related protocol classes.

Documentation in the form of descriptions, examples, and application-specific properties can be associated with OPM schemas as well as schema

components such as classes and attributes. Further, classes can be organized into clusters, and clusters can be nested.

The OPM Query Language (OPM-QL) [3] is an object-oriented query language similar to OQL, the ODMG standard for object-oriented query languages [10]. An OPM query consists of a SELECT statement, specifying the values to be retrieved for instantiations of variables satisfying the query condition; a FROM statement, specifying the variables that occur in a query and the classes or attribute values which they range over; and an optional WHERE statement specifying conditions on instantiations, where conditions consist of and/or compositions of simple atomic conditions. A query may also involve local, inherited and derived attributes and path expressions starting with these attributes.

The Object-Protocol Model Data Management Tools

The OPM data management tools provide facilities for developing and accessing databases defined using OPM, for constructing OPM views of existing relational databases and structured files, for representing database schemas using alternative data-models, for publishing schemas in various formats, and for querying databases through uniform OPM views. The OPM multidatabase tools provide facilities for exploring multiple heterogeneous databases that have either native OPM schemas or retrofitted OPM views. We will briefly describe each of the OPM data management tools below.

The OPM Database Development Tools

OPM schemas can be specified using either a regular text editor or using the graphical OPM Schema Editor. The OPM Schema Editor is implemented in Java and provides a graphical interface implemented using the Java Abstract Windowing Toolkit (AWT). The tool allows object and protocol structures to be specified incrementally by defining new OPM classes, modifying existing OPM classes and defining attributes of classes. OPM schemas can be also examined graphically on the Web using the OPM Schema Browser. The OPM Schema Editor and the OPM Schema Browser provide facilities for generating Postscript diagram, LaTeX document and HTML file representations of OPM schemas.

OPM schemas are maintained as ASCII files that can be passed to the OPM Schema Translators described below, in order to generate the corresponding DBMS-specific database definition and constraints.

Individual OPM schemas or several related schemas can be documented in a Database Directory and Schema Library (DD&SL). A DD&SL contains information on individual databases, such as database names, underlying DBMS, access information, and the database schemas represented in OPM and other alternative notations, such as the Extended Entity-Relationship (EER) model, the relational model, and the ASN.1 data exchange notation. In addition the DD&SL contains information about the relationships between databases represented in the DD&SL, or *inter-database links*, which can be used in exploring across databases and formulating multi-database queries. The DD&SL is used by the Multidatabase Query Tools described later, in order to provide the information necessary formulating multidatabase queries. The information in a DD&SL may also be automatically converted to a hierarchy of HTML pages, so that the DD&SL may also be browsed using a Web browser, such as Netscape or Explorer (see [6] for details).

The OPM Schema Translator translates OPM schemas into relational database definitions and database procedures implementing the OPM retrieval and update methods [2]. Informally, the translation of an OPM schema into a relational database definition entails mapping every OPM object or protocol class C into a primary relation R . Depending on their type (primitive, abstract, simple, tuple, etc.), non-derived attributes of C are mapped into local attributes of R , foreign-key attributes of R , or additional auxiliary relations with appropriate foreign-key to primary-key references. Derived OPM attributes are mapped into relational procedures that are used for computing their values at run time. The OPM Schema Translator also generates a *mapping dictionary* containing information on the OPM to relational database mapping.

The OPM Retrofitting Tools

The OPM Retrofitting tools [5] can be used for constructing and maintaining OPM views on top of existing flat files or relational databases that were not developed using OPM. These tools follow an iterative strategy for constructing OPM views: first a canonical (default) OPM view is generated automatically from the underlying database schema; then this canonical OPM view can be refined using schema restructuring operations, such as renaming or removing classes and attributes, merging and splitting classes, adding or removing subclass relationships, defining derived classes and attributes, and so on. A mapping dictionary records the information regarding the relationships between the view (OPM) constructs and their corresponding representations in the underlying database.

The OPM Database Query Tools

The OPM Database Query tools provide support for specifying and processing OPM-QL queries over native OPM databases, generated using the OPM

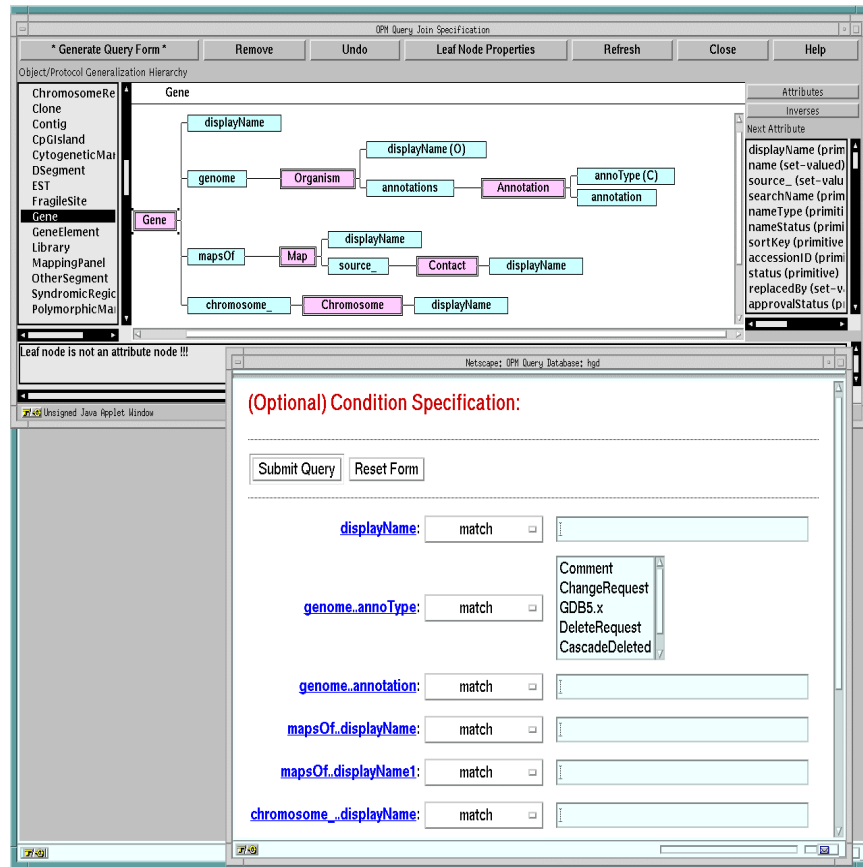


Figure 2: Constructing an OPM Query with the OPM Web Query Interface.

Database Development tools, or databases retrofitted with an OPM view, and for browsing the results of these queries. In addition, for native OPM databases, the query tools support data manipulation (inserting, deleting and updating). The

queries are evaluated using OPM Query Translators which employ the information in the mapping dictionary generated by the Schema Translator or Retrofitting tools in order to generate equivalent queries using the query facilities provided by the underlying DBMS or file system. For relational databases, the OPM Query Translators generate SQL queries in the particular dialect of SQL supported by the underlying relational DBMS, and then convert the query results into an OPM data structure.

Flat files are queried using SRS (Sequence Retrieval System) [8], a system originally developed at the European Bioinformatics Institute for accessing archival sequence databases. SRS parses flat files into an object structure that can be used as the initial (canonical) schema for the OPM Retrofitting Tools, and also indexes these files. The query facilities provided by SRS are limited to regular-expression searches on indexed string fields and comparisons on numeric fields, and therefore OPM queries cannot be entirely translated into SRS queries. Consequently, in order to provide general OPM query facilities on flat files, it is often necessary to perform further local processing of the SRS query results using the OPM Multidatabase Query Processor described below.

Application programs can interact with the OPM query translators either via a C++ API or by calling the query translators as Unix command-line programs. The later can be achieved using Perl or Unix shell scripts, via temporary files for passing OPM queries and query results.

The OPM Web Query Interface [6] has been designed to provide an extension to the ubiquitous Web (HTML) query forms that users are already familiar with, so that using this interface will not require learning an entirely new querying paradigm. Instead of providing predefined query forms, the OPM Web Query Interface provides support for constructing a query tree by selecting classes and attributes of interest using a graphical user interface, and for dynamically generating HTML query forms based on this query tree. Further query condition specification can be carried out by filling in these query forms.

Query specification using the OPM Web Query Interface is illustrated by the example shown in Figure 2, where class Gene is selected as the root of the query tree. Attributes such as `displayName`, `genome`, `mapsOf`, and `chromosome` are then selected from the list of attributes associated with class Gene and added to the query tree. Next, the value classes of abstract attributes, such as `chromosome`, can be selected and their attributes, for example attribute `displayName` of class `Chromosome`, can be added to the query tree. Primitive attributes, such as `displayName` and `annotation`, form the leaves of the tree. Once the query tree is completed, an HTML query form (see the form in the lower half of Figure 2) is generated for specifying conditions.

The OPM Multidatabase Tools

The OPM Multidatabase tools [7] provide facilities for exploring, querying and combining data from multiple heterogeneous databases via their native OPM schemas or retrofitted OPM views. The tools employ a Database Directory as

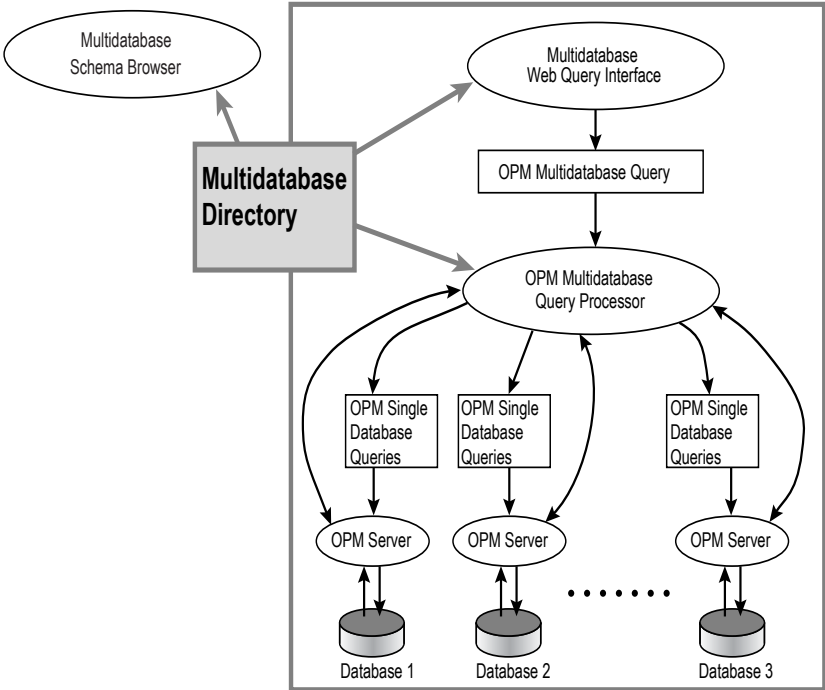


Figure 3: The OPM Multidatabase Tools.

described above, that records the metadata needed to access databases and information about the inter-database links.

The diagram in Figure 3 shows the architecture of the main OPM Multidatabase tools. A Java-based OPM Multidatabase Schema Browser, similar to the OPM Schema Browser for single databases described above, allows browsing the schemas of multiple databases and following inter-database links. A Web-based Multidatabase Query Interface provides support for

The screenshot displays the OPM Multi-Database Join Specification interface. The main window shows a query tree with nodes: (product), GSDb Product, dbpr object, GSDb-DBLink, GSDb_to_GDB_gene, GDB.Gene, dbpr segment, GDB-MapElement, and map. Below the tree is a form for optional condition specification with fields for product (match, %kinase%) and chromosome_nr (=, "4"). A third window shows the generated SQL query:

```

SELECT gene = @gene.displayName,
       accessionID = @gene.accessionID,
       product = @pr.authoritative_name
FROM   @pr in GSDb:Product,
       @dbl in @pr.!object[DBLink],
       @gene in @dbl.GSDb_to_GDB_gene
WHERE  @pr.authoritative_name MATCH "%kinase%"
AND    @gene.!segment[MapElement]map.chromosome.searchName = "4"
;

```

At the bottom, the text "Find protein kinase genes on chromosome 4" is displayed.

Figure 4: Constructing an OPM Query across GDB and GSDB with the OPM Multidatabase Web Query Interface.

interactively specifying OPM queries across multiple databases using a combination of graphical Java-based tools and HTML forms. This query interface is similar to the single-database OPM Web Query Interface, except that one can first select a database from a list, before selecting the classes that will form the query tree. The top two windows in Figure 4 show an example of the Web query interface in use. In this example, first GSDB is selected from the

list of component databases and then class `Project` of GSDb is selected as the root of the query tree. Further, the query tree may involve inter-database links in addition to regular OPM attributes, such as `GSDb_to_GDB_gene` in the example. These links are used to associate classes in different databases. Once the query tree is completed, an HTML query form is generated for specifying conditions, possibly involving attributes of classes from different databases (see the form in the middle part of Figure 4). The Multidatabase Web Query Interface generates queries in the OPM Multidatabase Query Language, OPM*QL, which are then executed using the OPM Multidatabase Query Processor. The bottom window in Figure 4 shows a OPM*QL query equivalent to the query expressed using the Web query tools in the other two windows.

The OPM Multidatabase Query Processor interacts with Database Servers for each database involved in the multidatabase system. Each Database Server provides DBMS specific query translation functions and facilities to execute single-database queries expressed using OPM-QL. The OPM Multidatabase Query Processor interprets OPM*QL, generates queries over the individual databases, and performs local data manipulations necessary to combine the results of individual queries and to provide functionality not supported by the individual databases. Since different databases and DBMSs may support different query facilities, the queries generated for each database are dependent on the particular subset of OPM-QL supported by the Database Server: in general as many of the query conditions as possible are performed by the Database Servers, while conditions which may not be tested by the remote DBMS are evaluated locally by the Multidatabase Query Processor.

Applications and Experience

In this section, we briefly discuss typical applications of the OPM tools and the experience we gained developing and applying these tools.

The OPM Database Development and Query Tools

The OPM Database Development Tools were first used for developing a prototype database for a large-scale DNA sequencing project at Caltech.⁵ Subsequently, the OPM Database Development and Query Tools were employed for developing and then extending version 6 of the Genome Database (GDB) using the Sybase DBMS, and the Primary Database of the German

⁵ See <http://gizmo.lbl.gov/jopmDemo/shotgun.html>

Human Genome Resource Center (RZPD) using the Oracle DBMS, as well as other scientific and traditional databases. OPM Web Query Interfaces are employed for accessing several OPM based databases such as RZPD, while GDB is accessed via custom-built Web query interfaces.

OPM and the OPM tools helped in improving the efficiency of developing and maintaining these databases and, to some degree, in insulating their applications from the underlying DBMSs. As a methodology, OPM encourages and provides support for comprehensive database documentation. Various OPM tools provide facilities for taking advantage of this documentation during database exploration. Further, by insulating applications and users from the underlying DBMSs, the OPM tools simplify the task of transferring databases to other DBMSs such as object-relational DBMSs.

Our strategy of providing support for wrappers on top of commercial relational DBMSs has proved to be effective. In spite of early doubts expressed in the genome database community, relational DBMSs are widely used for implementing biological databases, while usage of object-oriented DBMSs has been limited and problematic (see [9] for details). The emerging object-relational (*Universal Server*) DBMSs, such as Oracle 8 and the Informix Universal Servers, which represent the next generation DBMSs, are evolving from their relational counterparts and will replace them eventually. However, while the added functionality of these new DBMSs allows the development of potentially more powerful databases, it also increases the complexity of designing, implementing and querying such databases (see chapter 15 of [12]). We believe that the OPM tools, appropriately adapted and enhanced, will continue to provide the same advantages for databases implemented with Universal Servers as those currently provided for relational databases.

Since keeping track of historical information is important for archival databases such as GDB, we have incorporated a versioning mechanism into OPM. The research underlying this mechanism, conducted in collaboration with our colleagues at GDB, lead to interesting results [4], including the realization that the implementation of such a mechanism in a relational database framework causes an unacceptable overhead for large production databases.

Our work on the OPM Database Development and Query Tools has benefited from the feedback and suggestions received from our collaborators and users. Especially valuable has been our close collaboration with the GDB staff, including Ken Fasman, Stan Letovsky, Peter Li and their colleagues. This collaboration helped us cope with the peculiarities of the Sybase DBMS and proved invaluable in improving the performance of the SQL code generated by the OPM tools. Further, GDB staff provided suggestions for extending the OPM tools with new capabilities. The more recent collaboration with the RZPD

group lead by Brian Toussaint has also been instrumental in improving the OPM tools.

The OPM Retrofitting and Multidatabase Tools

We have been experimenting with the OPM Retrofitting and Multidatabase Tools since January 1996, and our experience with these is therefore more limited than with the other OPM tools. We have applied the OPM Retrofitting Tools to several relational databases, including the Genome Sequence Database (GSDB) and the bio-collections database of UC Berkeley's Museum of Vertebrate Zoology (MVZ), and to biological flat file databases such as GenBank. Retrofitting allowed us to install the OPM Web- Browsing and Query Interfaces on top of these databases. While more powerful and flexible than the native Web based interfaces provided by these databases, the OPM query interfaces may not perform as well as the canned-queries that have been manually optimized for directly accessing these databases.

Our collaboration with Thure Etzold at the European Bioinformatics Institute in installing the OPM Retrofitting Tools and Query Interfaces on top of SRS, has provided the ability to access via OPM interfaces a wide variety of structured flat-file databases, including many of the major archival molecular biology databases. SRS reads and indexes flat-files using parsers defined using in the Icarus language, and then maps them into objects. Initially, we have employed existing Icarus parsers for various molecular biology databases such as GenBank. Next, we intend to experiment with more sophisticated parser definitions and new retrofitting techniques that would allow constructing more detailed and semantically richer OPM views for such databases.

The OPM Multidatabase Tools have been applied to the construction of a Molecular Biology Database Federation that includes GDB, GSDB, and GenBank. The experience gained with this prototype has driven several enhancements of the Multidatabase Query Tools.

Implementation Issues

The OPM Schema and Query Translators have been developed mainly in C++. The OPM Schema Editor, Browser, and Query Interfaces were first developed using X11/Motif. We encountered numerous problems maintaining and porting these early OPM editors and interfaces. Currently all our OPM editors and interfaces are implemented in Java. In spite of its present instability, using Java substantially reduced the development and maintenance of these tools.

For implementing interfaces between the OPM Web Query Interfaces and Query Translators, we are using CGI, which is easy to use and maintain given our multiple-language programming environment. CGI is a natural choice for dynamically creating HTML pages. In the current CGI-based implementation, the Java-based query construction front-end uses CGI to call a Perl script wrapper that invokes a C++ version of the OPM Query Translator adapted for generating HTML files. We have also considered other communication alternatives, including Java and C++ sockets, CORBA based products with Java and C++ interfaces, such as IONA's Orbix and OrbixWeb, and Java 1.1's Remote Method Invocation (RMI) interface, with C++ applications accessed through the Java Native Interface (JNI). These alternatives are discussed in more detail in [6].

The interfaces between the OPM Multidatabase Query Processor and the OPM Database Servers are implemented using a CORBA product (either IONA's Orbix or Visigenic's VisiBroker). As a programming environment, CORBA's object-based communication between applications (possibly developed in different languages) is convenient. However the Interface Definition Language (IDL) is limited in the data structures that can be passed between applications, and therefore requires extraneous conversions between the 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 porting an implementation of our query tools to different CORBA products is time consuming.

Acknowledgments

Between 1992 and 1997, the OPM tool development was carried out in the framework of the Data Management R&D Group at Lawrence Berkeley National Laboratory (LBNL), with funding provided by the Office of Biological and Environmental Research and the Mathematical, Information, and Computational Sciences Division of the Office of Energy Research, U.S. Department of Energy under Contract DE-AC03-76SF00098. We want to thank Arie Shoshani, the head of the Data Management R&D Group at LBNL, for his active support and encouragement.

References

1. Chen, I. A. and Markowitz, V. M. *An Overview of the Object-Protocol Model (OPM) and OPM Data Management Tools*. Information Systems, 20(5), 1995, pp. 393-418.
2. Chen, I. A. and Markowitz, V. M. *The OPM Schema Translator*. Technical Report LBNL-33706, Lawrence Berkeley National Laboratory, 1996.
3. Chen, I.A., Kosky, A., Markowitz, V.M., and Szeto, E. *The OPM Query Translator*. Technical Report LBNL-33706, Lawrence Berkeley National Laboratory, 1996.
4. Chen, I. A., Markowitz, V.M., Letovsky, S.I., Li, P., and Fasman, K.H., *Version Management for Scientific Databases*. Advances in Database Technology- EDBT-96. Lecture Notes in Computer Science, vol. 1057, P. Apers & al (eds), Springer-Verlag, pp. 289-303, 1996.
5. Chen, I.A., Kosky, A.S., Markowitz, V.M., and Szeto, E. *Constructing and Maintaining Scientific Database Views*. Proceedings of the 9th Conference on Scientific and Statistical Database Management, IEEE Computer Society, 1997, pp. 237- 248.
6. Chen, I.A., Kosky, A.S., Markowitz, V.M., and Szeto, E. *Exploring Databases on the Web*. Technical Report LBNL-40340, Lawrence Berkeley National Laboratory, 1997.
7. Chen, I.A., Kosky, A., Markowitz, V.M., and Szeto, E., *Exploring Heterogeneous Biological Databases: Tools and Applications*. Technical Report LBNL-40728, Lawrence Berkeley National Laboratory, 1997.
8. Etzold, T., and Argo, P. SRS, An Indexing and Retrieval Tool for Flat File Data Libraries. Computer Applications of Biosciences, Vol. 9, No.1, pp. 49-57, 1993. See also [hp://www.embl-heidelberg.de/srs/srsc](http://www.embl-heidelberg.de/srs/srsc).
9. Goodman, N. *An Object-Oriented DBMS War Story: Developing a Genome Mapping Database in C++*. In Modern Database Management: Object-Oriented and Multidatabase Techniques, W. Kim (ed), ACM Press, 1994.
10. *The Object Database Standard: ODMG-93*. Cattell, R. G. G. (ed), Morgan Kaufmann, 1996..
11. *Programmer's Reference*. National Center for Biotechnology Information, 1991. See also: <http://www.inria.fr:80/rodeo/personnel/hoschka/asn1.html>.
12. Stonebraker, M. *Object-Relational DBMSs: The Next Great Wave*. Morgan-Kaufman Publishers, Inc., 1996.