

Using the Functional Data Model to Integrate Distributed Biological Data Sources

Graham J.L. Kemp, Joel Dupont and Peter M.D. Gray

Department of Computing Science
King's College, University of Aberdeen
Aberdeen, Scotland, U.K. AB9 2UE
E-Mail: {gjlk|pgray}@csd.abdn.ac.uk
URL: <http://www.csd.abdn.ac.uk/~pfdm>

Abstract

There is unavoidable heterogeneity in the databanks and databases currently used for biological data collections. We are aiming to develop a system that will provide uniform access to heterogeneous databases via a single high-level query language or graphical interface and will enable multi-database queries. We have taken an approach in which high-level code in an object-oriented database system (based on the functional data model) is used effectively as a mediator between distributed heterogeneous databases. The resulting system enables us to ask queries that combine, for example, geometric calculations on three-dimensional protein structures stored locally and access to databanks held at the European Bioinformatics Institute, while making use of existing search engines and indexes when accessing remote data.

Keywords: bioinformatics, functional data model, mediator, multi-database, Prolog.

1 Introduction

Molecular biology provides one of the most challenging application domains for database research. This is because of the rich variety of data, from genome sequences through data on protein structure and function to data on full organisms, and also the large quantities of data that are becoming available at present through modern experimental techniques. This is a rewarding application area since developments in our ability to integrate and analyse these data can lead to an increase in our understanding of biological function at all levels.

The term *bioinformatics* refers to the application

of computers in addressing biological problems, and is most frequently used in relation to storing and searching genome sequence data and protein sequence data. These data sets are essentially “one-dimensional”, with *genome sequences* consisting of letters representing the sequence of the four bases in a nucleic acid chain, and *protein sequences* consisting of letters representing the sequence of amino acid residues in a protein chain. The European Bioinformatics Institute (EBI) was established in 1994 as an outstation of the European Molecular Biology Laboratory (EMBL), with the development and distribution of sequence databases as one of its major roles.

Bioinformatics covers other areas including protein structure analysis and modelling, and this has been the focus of our work in Aberdeen [10] [15]. This work has made use of an object-oriented database, P/FDM [9], which is based on the functional data model [22]. Protein structure data has very different characteristics to sequence data, and we have found P/FDM to be well suited to answering queries requiring geometric calculations combined with data retrieval.

This paper describes recent work in which we have used the functional data model and the P/FDM system to integrate data held locally on protein structures with databanks at the EBI. The resulting system enables us to ask queries that combine, for example, geometric calculations on three-dimensional protein structures and access to data at the EBI, while making use of existing search engines and indexes developed at EMBL/EBI when accessing remote data. First, we shall look at some examples of bioinformatics data resources and consider why heterogeneity is important. We shall then discuss two alternative architectures for achieving our objectives. We shall describe with examples how we have integrated remote

and local data, and then discuss how Prolog and the P/FDM system play the role of a mediator in the resulting multi-database system.

2 Heterogeneity in molecular biology databases

Many different file and database management systems are currently in use with biological data sets. There are several databanks with data on proteins and other biological molecules at the EBI. These databanks consist of readable flat files of data. Each databank contains a different kind of data, and each has its own format for its entries. The EMBL and Swissprot databanks are respectively the principal genome and protein sequence databank at the EBI. Other databanks include the Medline bibliographic databank and HSSP databank of homology-derived structural information about proteins.

The Protein Data Bank (PDB) [2] is a collection of three-dimensional protein structures which have been determined by X-ray crystallography or nuclear magnetic resonance. The PDB is distributed in flat-file format, although various projects have taken data from the PDB and reorganised it using relational or object-based systems. The BIPED [12] and SESAM [11] projects both used commercial database management systems. The ALI interface to SESAM provides a query language that enables queries involving amino acid residues at relative offset positions in a protein chain to be expressed easily, and answered efficiently by taking advantage of the known tuple ordering.

In our own work we have taken protein structure data from BIPED, SESAM and PDB and, in the case of antibodies, protein sequence data from the Kabat databank, storing these in accordance with a functional data model [10] [17].

Object models are gaining in popularity in the genome community, who recognise the need to integrate heterogeneous resources, and believe that schema based on an object model will help with this integration. The current release of the Genome Data Base (GDB Version 6) uses the Object-Protocol Model (OPM) [5], and the ACEDB model [6] is widely used with various specialised collections of genetic data.

There are several reasons why this heterogeneity has arisen. For example, experimental scientists are usually the custodians of these data collections and they are likely to use whatever tools were known and available to them. They may choose a simple physical representation to make porting and exchanging data

easy.

A more significant reason is that certain physical representations are more appropriate than others for particular kinds of data, and are better suited to the kinds of searches performed against those data. Frequently a great deal of effort goes into developing customised search engines to answer particular kinds of query efficiently, and in developing graphical interfaces tailored to the application. We should like to take advantage of this earlier investment by making use of existing search engines whenever possible.

3 Alternative architectures for integrating databases

We are aiming to develop a system that will provide uniform access to heterogeneous databases via a single high-level query language or graphical interface and will enable multi-database queries. This objective is illustrated in Figure 1. Data replication and multi-databases are two alternative approaches that could help us to meet this objective.

3.1 Data replication approach

“Data replication” would be one possible approach. In this architecture, all data from the various databases and databanks of interest would be copied to a single local data repository, under a single database management system. This approach is taken by Rieche and Dittrich [20] who propose an architecture in which the contents of biological databanks including the EMBL nucleotide sequence databank and Swissprot are imported into a central repository.

However, we believe that a data replication approach is not appropriate for this application domain for several reasons:

Space. The amount of biological data in existing databanks and databases is very large, and new data are being generated at an increasing rate. Few sites have sufficient disc space to mirror all data that may be needed by that site’s users. Currently, national bioinformatics nodes provide a repository service for many databanks. However, a site wishing to integrate its private local data with the existing shared resources would be forced to mirror (at least part of) these.

Updates. Scientists want access to the most recent data. They want on-line access to results reported in the current journals as soon as these have been deposited in a databank or database. Whenever one of the contributing databases is updated the

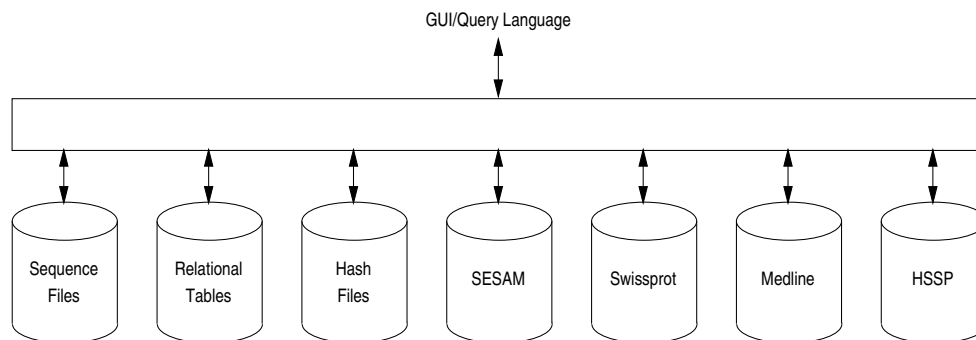


Figure 1: Users should be able to access heterogeneous distributed bioinformatics resources via a single query language or graphical user interface.

same update should be made to the data repository. (Changes and deletions are sometimes made to biological databases, but are much less frequent than additions.)

Advantages of heterogeneity. Significantly, by adopting a data repository approach the advantages of the individual heterogeneous systems are lost. For example, many biological data resources have their own customised graphical interfaces and search engines. These will be tailored to the particular physical representation used with that data set. Rieche and Dittrich [20] acknowledge the need to use existing software and propose implementing “exporters” to export and convert data from the data repository into files that can be used as input to software tools.

In summary, we believe that a data replication approach would require software, hardware and human resources beyond what is reasonably available at each site that would want to use the data.

3.2 Multi-database approach

We favour a multi-database approach which makes use of existing remote data sources, with data described in terms of entities, attributes and relationships described in a high-level schema. The schema is designed without regard to the physical storage format(s). Queries are expressed in terms of the conceptual schema and it is the role of a complex software component called a *mediator* [23] to decide what component data sources need to be accessed to answer a particular query, to organise the computation and to combine the results. The suitability of a multi-database approach for integrating biological databases is also proposed by Karp [14].

With reference to the points listed in Section 3.1:

- No extra space is needed locally, apart from a temporary cache for results retrieved from remote sites.

- Since a single copy of the data is used with no local mirroring, all updates to the remote component databases are immediately available.
- Our multi-database does not affect other users of the component data resources who can, if they wish, continue to use these exactly as before. Furthermore, we can take advantage of customised software tools by sending requests to these from the mediator.
- We do not need to import large data sets from a variety of sources. Nor do we need to convert all data for use with a single physical storage schema. However, extra effort is needed to achieve a mapping from the component databases onto the conceptual model.

4 Accessing databanks at the EBI via P/FDM

One way to examine entries in the databanks at the EBI is to browse interactively. This can be done using WWW browsing tools like Netscape. In doing this, databank entries are presented as HTML documents, with links to enable the user to jump to related entries in other databanks. However, the user has to control the navigational search by mouse-clicks, which is awkward when sets of results satisfying various criteria are wanted. Instead, it is better to use a database query language to automate the process.

Browsing across databanks is facilitated by the SRS system [7]. The Sequence Retrieval System (SRS) maintains indexes relating entries in one databank to entries in others. Thus, SRS maintains a large network of related databank entries. SRS provides search tools that enable users to retrieve particular entries or to find the accession codes of entries that satisfy specified criteria, e.g. entries with fields that match a given pattern string, or fields with numeric values in a given

range. SRS also provides “link” operators that enable the user to follow cross references and find all entries in one databank with corresponding entries in another. These can even follow a chain of cross references where there is no “direct” link between the two.

Locally, we store data in either hash files or relational tables. However, in addition to accessing local data, we want to access data in databanks held at EBI both from application programs and high-level queries. In doing this, we aim to *insulate* the application programmer or database user from requiring to know either which particular databanks need to be accessed or the details of the databank’s file format.

In addition to interactive access, SRS can also be accessed from a running program via an FTP server at EBI. Using this, it is possible to write programs that generate FTP commands and thus interrogate the SRS system automatically without intervention from a human user. We have used this facility to provide access to databanks at EBI from a P/FDM system running at another site.

4.1 A simple mapping onto the FDM

In order to integrate data from different databanks, we need to map data held in the databanks onto a conceptual schema. In the functional data model schema for our protein database we have several entity classes including *protein*, *chain* and *residue*. Many databank entries at EBI can be considered to be attributes of a *protein*. A databank entry could simply be regarded as a text string, but this would ignore the useful information that is implicit in the file layout, and which can be interpreted by a scientist looking at a data file. If we want this information to be used by a program, then we need to write code that can extract information from the databank files. The functional data model schema allows us to define some attributes with values stored in the database, while values for others are derived on demand. The details are hidden from the user, since a feature of Daplex is that calls to derived attributes and stored attributes use the same functional syntax.

A straightforward mapping is to map each databank onto an entity class and then to consider each databank entry to be an object instance. Thus we would have entity classes like *swissprot_entry* and *embl_entry*. Fields in these databanks can be modelled by multi-valued string-valued functions. For example, the function *definition* can be defined on the class *swissprot_entry* and will return the lines of a Swissprot databank entry beginning with the code “DE”, and the function *id* can be

defined to return the text from the identification line. The Daplex schema describing the mapping to some of the databanks at EBI can be found at <http://www.csd.abdn.ac.uk/gjlk/ssdbm8/schema.html>. Using this mapping, we can write the following Daplex query to find the identifiers of all Swissprot entries with the word “antigen” in their definition:

```
for each e in swissprot_entry such that
  sub_string("antigen", definition(e))
print(id(e));
```

This would be translated into the following command which is sent to the SRS FTP server:

```
get getz+[swissprot-def:antigen]+-f+'id'
/tmp/srs1
```

In P/FDM, Daplex queries are first translated to a list comprehension [19] [13] [4] [16], as used in functional programming, and this simplified form of the query is used as the basis for further translation into code that is executed to answer the query. Normally, a Daplex query is translated to Prolog code that includes calls to primitive data access routines (these are discussed elsewhere, e.g. [9]). In our present work we have written an extension to the P/FDM code generator that can produce SRS code matching a given list comprehension for queries. The translation process is illustrated in Figure 2.

The results of running the SRS query are stored in a file on the local machine. For this query, the contents of this file will simply be displayed to the screen. However, for more complex queries, further processing may be done on the result files retrieved from the SRS server (see Section 4.2).

A second example Daplex query demonstrates how we can combine local and remote data (Figure 3). This query accesses our antibody database [17]. Results retrieved from the EBI are presented to the user along with data values retrieved from the local antibody database. The distance calculation made in this query is the simplest kind of geometric calculation that we make. Our Daplex queries frequently include calls to functions written in C to measure bond angles and torsion angles, and calls to FORTRAN code to superpose one structural fragment on another to compare three-dimensional similarity.

We are currently limited to using a single variable to make a join between a local sub-query and the remote sub-query. However, the local sub-query can itself be arbitrarily complex, and the complexity of the remote sub-query is constrained by the capabilities of SRS e.g. all values returned from an SRS query must be from

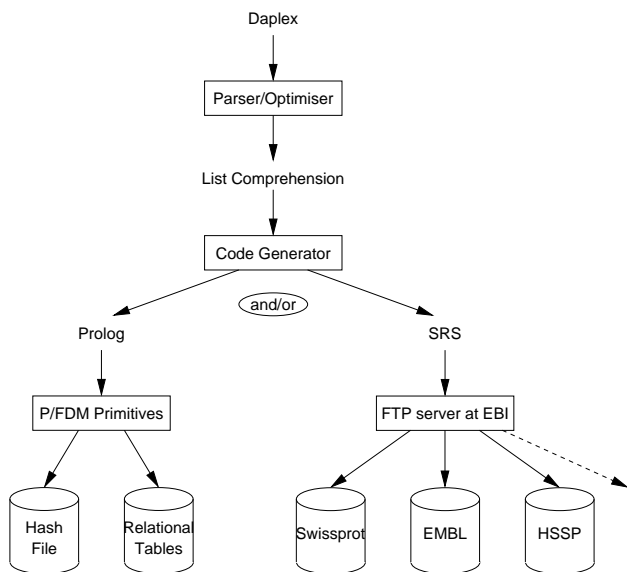


Figure 2: A Daplex query may be translated into Prolog code to access data held locally or SRS code to access data at EBI. However, some Daplex queries will need to access both local and remote data and so will be translated into a combination of Prolog and SRS code.

the same databank (although the query may check on values in related entries in different databanks). Future work includes extending the system to handle more complex queries involving several joins between local and remote databases.

4.2 A more sophisticated mapping onto the FDM

The simple mapping used in the previous section enables us to integrate the databanks at the EBI into the P/FDM system easily. However, this mapping requires users to know the names of the component databanks, their data formats and their field names. While many users may have this knowledge, it would be more convenient to be able to express a query in terms of meaningful concepts declared in a high-level schema. One way to do this is to write named derived functions in Daplex that can be used by non-expert users to perform repeated tasks.

As an example, scientists sometimes find that the same kind of amino acid residue occurs at a particular position in a protein chain in all proteins belonging to the same “family” (“family” here means the set of all proteins with a similar amino acid sequence). Information about these “invariant residues” is useful when trying to construct three-dimensional models of

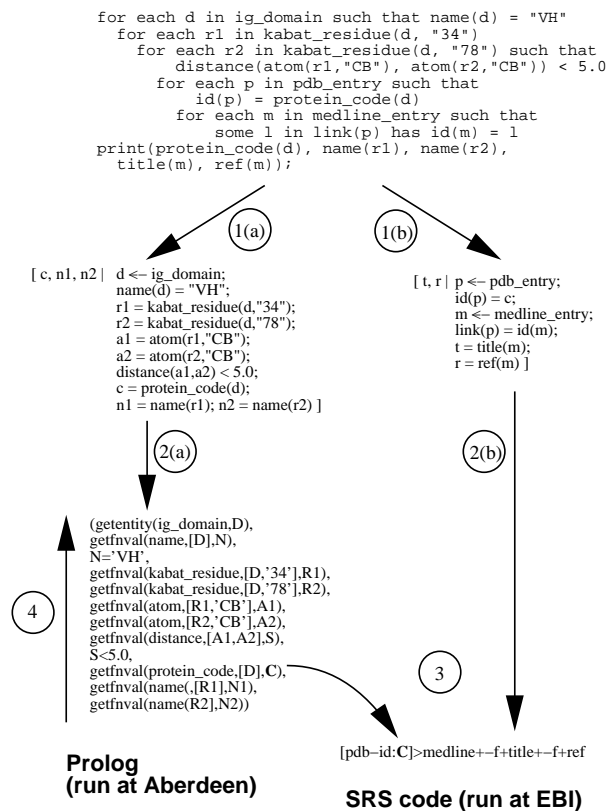


Figure 3: An example query that accesses local and remote data. This query finds all heavy chain variable domains (“VH” domains) and examines the amino acid residues at positions “34” and “78” in the structural framework of the domain. The separation between their $C\beta$ atoms is calculated and if this is less than 5.0 Angstroms a call is made to the FTP server at the EBI to find the titles and full references of all related articles. This query is processed as follows: 1. The Daplex query is translated to a list comprehension, which is split into two parts, (a) referring to local data and (b) remote data. 2. List comprehension (a) is translated to Prolog code including calls to the P/FDM primitive routines. List comprehension (b) is translated to SRS code. 3. The Prolog code is run at Aberdeen, providing a value for the variable C. For each value found, the value is added to the SRS code, which is then run at EBI. 4. Prolog backtracks to find other values for the variable C, and new SRS commands are sent to EBI.

proteins belonging to the family. Therefore, it would be useful to know for each residue in our structural database whether it is at one of these “invariant” positions.

First, we need to define an attribute called *at_invariant_position* on the class *residue*. This function will take an amino acid residue as its argument and return a boolean value indicating whether a given residue is always found at that particular position in related proteins. This information can be found by examining the HSSP file [21] corresponding to the Protein Data Bank (PDB) entry for the protein being examined.

Part of an HSSP file contains the amino acid sequence of a protein with a known structure, together with the sequences of those proteins (often with no known structure) which are similar to the first. In this part of the data file, each row represents a position in the protein chain, and each column represents a different protein chain. Thus, one of the columns contains the one-letter codes of the residues in the protein with known structure (these codes describe its “sequence”), and each of the following columns contains the sequence of one of the similar proteins. Amino acid residues at corresponding positions in different protein are listed on the same line. Each line also contains other information including the PDB identifier of the residue at that position in the known structure and the variability of residue occurrences at that position. Invariant positions are identified as those with a variability of zero. Thus, the code providing a value for the function *at_invariant_position* can find a value by:

- finding the PDB code of the protein in which the residue occurs and the residue identifier from the local database,
- calling out to the FTP server at EBI to retrieve the HSSP entry for the protein with the given PDB code, and then
- processing this file locally to find whether that position is indeed invariant.

An alternative approach would be to extract all information about invariant positions from HSSP files in a batch operation and to add this information to the local database. This would have the advantage that one would not need Internet access to use this attribute, making it possible to use these data on a stand-alone workstation. Even if a networked workstation was being used it would be faster to retrieve a value from a local database than to retrieve a data file from a remote site and then process this to identify values of interest. On the other hand, there are

also disadvantages with this approach. A minor disadvantage is that this approach would require a little extra storage space on the local machine. More significantly, to keep up-to-date, the local database would need to be updated whenever the HSSP files are recalculated as new sequences become known. In contrast, by accessing a central collection of data we can ensure that the most up-to-date data are used to provide an answer to our query.

These approaches have their advantages and disadvantages and we might expect different users or sites to prefer different approaches depending on their individual needs. By adhering to the principles of data independence [8] this can be accommodated; either approach can be followed and the query or application program can be used without change. A compromise approach would be to cache data retrieved for the remote site on the local machine so that if a piece of information is accessed again during a session then a locally stored value can be used without the need to call out across the Internet for a second time.

Rather than just printing the results of a database query to the screen, users of our system can define *action methods* [15] that can, for example, produce a display effect based on object instances and values found by a database search. Recently, we have implemented an interface between P/FDM and the molecular display program RasMol (written by Roger Sayle) that allows display commands to be sent from the P/FDM database to RasMol.

As an example of this, suppose we want to highlight on a molecular graphics display those residues in a protein that are conserved in all other members of that protein’s family. We might want to do this during a homology modelling exercise to help identify regions of the molecule that are likely to be structurally conserved.

We can now write a Daplex command to highlight invariant positions in crambin (PDB code “1CRN”):

```
for each p in protein such that
  protein_code(p) = "1CRN"
  highlight_invariant_positions(p);
```

In this example, *highlight_invariant_positions* is an action method that uses values from the derived function (or “method”) *at_invariant_position* to construct commands to automatically select and highlight a set of residues within RasMol. Figure 4 shows the structure of crambin displayed by RasMol before and after this query is run. Figure 5 shows how this query is processed.

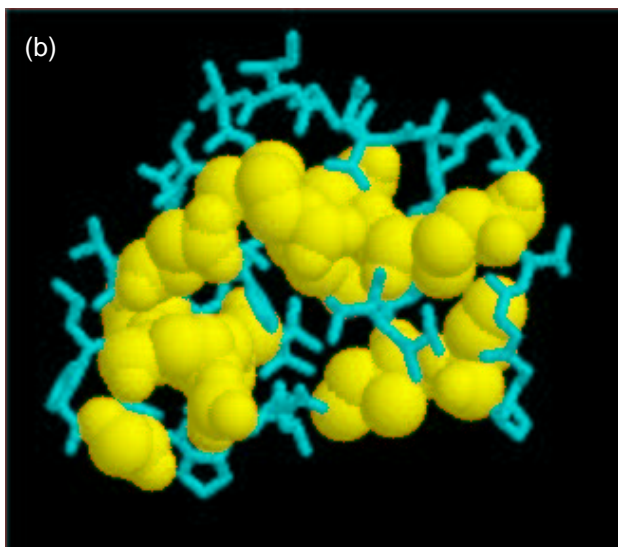
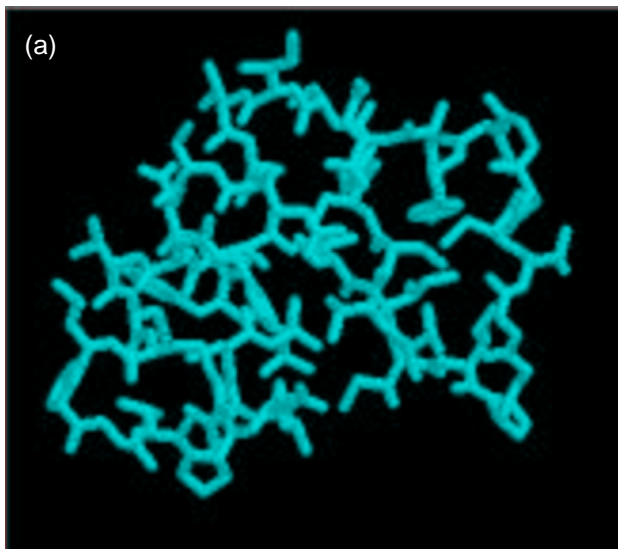


Figure 4: Molecular structure of crambin (display produced using RasMol). (a) stick representation of crambin. (b) stick representation of crambin but with atoms in the 12 invariant residues highlighted as spheres.

for each p in protein such that protein_code(p) = "1CRN"
highlight_invariant_positions(p);

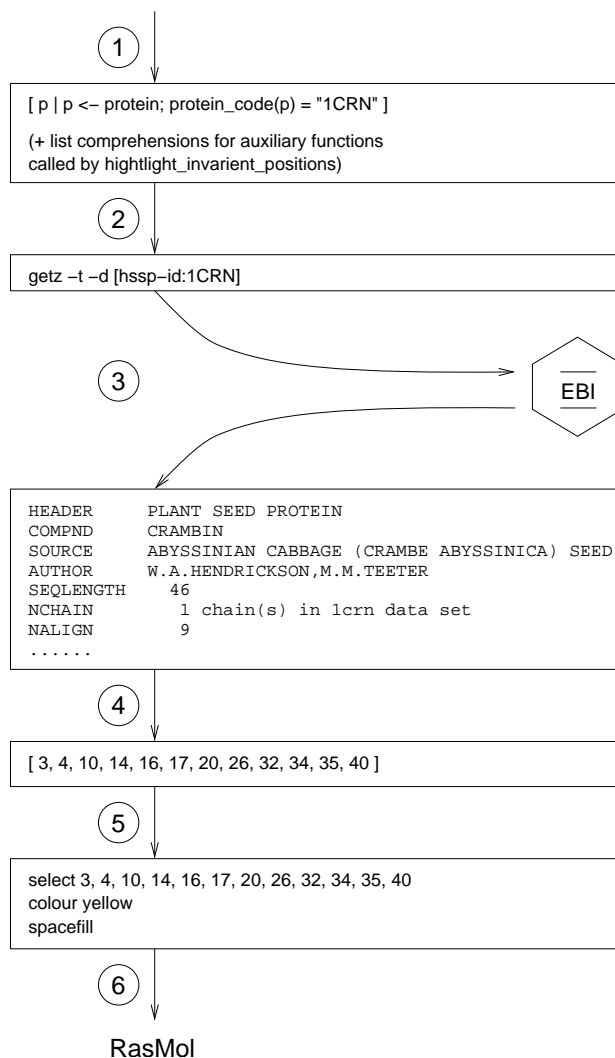


Figure 5: Highlighting invariant positions in crambin. 1. A Daplex query, typed by the user, is translated to a list comprehension. 2. An SRS command is generated. 3. The SRS command is sent to an FTP server at EBI, and an HSSP file is retrieved. 4. The HSSP file is processed locally to find invariant positions. 5. Display commands for the molecular display program RasMol are generated. 6. These are sent automatically to RasMol.

5 Discussion

The Daplex language, which we use, was originally designed for multi-database systems and query optimisation. It was designed by Shipman [22] working at CCA (Boston) in collaboration with MIT, and was adopted for a research programme (MULTIBASE [18]) into distributed heterogeneous database systems. Daplex was used to formulate the overall query, which was then translated down into subqueries in the languages of the component database systems (SQL or CODASYL at that time). However, the interest in this died out, probably because relational systems were not generally in commercial use at that time, and where they were in use as a distributed system, it was as a homogeneous system, tightly coupled, using the manufacturer's software.

The new factor today is the explosive growth of database systems in molecular biology on a variety of platforms, in different countries. It is unrealistic to expect these systems all to adopt the same platform. Currently, SRS itself "mirrors" copies of genome databases! However, this is not essential in principle, and it does not affect our argument. We use SRS mainly for the external identifiers and indexes it provides as a way into remote databases, not because these are mirrored. As the databases get bigger and more frequently updated this becomes less attractive. Thus a multi-database solution is needed, as presented in this paper.

A particularly interesting feature of Daplex is its use of variables which are, effectively, bound to object identifiers, ranging over objects or records of a particular type (or class). It also expresses computation of values or selection conditions in terms of functions over these objects. This makes it particularly suitable for work on object-oriented databases, or as a replacement for SQL when working with an object-oriented programming language [9]. Thus the combination of Smalltalk/Daplex could well be a replacement for C/SQL which was so popular in the 1980's!

In the context of multi-database work, it means that part of a Daplex query will be executed to create a set of external object identifiers. These are then sent across to another database along with a Daplex subquery translated into the target query language for that database, and parameterised with respect to the supplied object identifier. This sub-query will return a set of result values, possibly including more object identifiers. These object identifiers play the role of keys in relational systems in identifying the required data. They are usually supported by some form of index tables at each site, thus they can be used very

effectively to locate other objects either at the originating site or at other sites. This is the essence of our system.

Because of this use of object identifiers it is particularly straightforward to link across to external databases which themselves have an object schema. Recently genome databases have been moving to use object schemas. As noted earlier, there is much interest in the OPM model [5], which is being looked at both by Brookhaven for protein structure data and by John Hopkins for GDB-6. There is also interest in the ACEDB model [6]. The OPM model has a query language more like SQL in style, but still based on object identifiers. It uses relational storage, like object-relational databases, but every class has a table with a mandatory column for the object identifier. This object identifier may be compound (composed of several attributes) like a compound key in a relation. The authors explain that "since OPM is intended for a community of scientists that currently use relational DBMSs and are likely to move to object-oriented DDMSs in the future, we are developing OPM-based tools ... these tools will automatically generate DBMS methods...". This shows the value of an object schema for migrating data between different storage representations. It also shows the general applicability of automatic code generation from conceptual models to fit target implementations. Our work very much follows this strategy. One difference in the data model used in OPM is *protocols*, which are used to model experimental procedures. However, these are not so appropriate for other kinds of data, e.g. protein structure data, and are not used in the OPM schema for the Protein Data Bank [1].

We are unusual in using the Prolog language for most of our system. Prolog is very good for parsing query languages, for pattern-matching and for planning how to solve a complex query with data at different sites [13] [16]. Prolog term structures incorporating shared variables with initially unknown values serve to represent the query plan, and this enables us to transform it and execute it in an efficient manner. The Prolog system thus plays the role of a *mediator* as proposed by Wiederhold [23]. The mediator shows its intelligence in how it breaks the query into various parts for evaluation at each site, and how it transforms the parts to make use of different indexing features at the sites. Currently, much of the information about sites is built into the mediator, but we intend to make this more table-driven and extensible.

Buneman *et al.* have also written a mediator based on list comprehensions, but in a functional language,

and are using it mainly for sequence integration, and have used this with biological data [3]. They use a schema based on abstract data types with nested components, which suits sequence files, but also maps onto relational databases.

The functional data model [22] originated in the MULTIBASE project to integrate distributed data. Our research has shown how it can also be used with objects. We have also shown how the functional data model can provide an object view of flat databank files. In this work, it has been necessary to map flat files onto functional data model entity classes and attributes in order to integrate data from existing established databanks with local FDM objects. Note that we do **not** convert entire flat files into object format! Instead we use the flat file's own local indexing mechanism for efficient selective access. However, flat files are often well suited to particular kinds of data searches, and we have chosen to use a flat file representation in other work where it was necessary to search antibody sequence data efficiently [17].

We have developed a system where data access requests are expressed in terms of meaningful concepts and properties described in a conceptual schema. In answering these queries, commands are generated and sent across the Internet. Thus we can directly make use of a shared central collection of data and existing indexes and search tools developed for use with those databanks. However, the details of how this is done is hidden from the user below the level of the conceptual schema. The biologist (end-user) does not have to worry about how the data are stored or where it is located. Also, the local site does not have to take up space and time mirroring other large databases.

6 Conclusions

We have developed a system that provides uniform access to heterogeneous databases through a single high-level query language based on an object model. This model gives us interoperability with many other models including OPM and ODMG. The system is being used to relate antibody structure data held locally to sequence data held remotely through cross-reference indexes maintained by SRS.

Our system was developed independently of the proposals made by Peter Karp at the MIMBD'95 meeting in Cambridge (1995). However, it has remarkable similarity. Karp proposes a system with a mediator component that forms a plan to break the query into sub-queries that are translated against the

different databases and sent to them by using appropriate drivers. Our system uses a mediator component written in Prolog, to perform this task. Its architecture, based on the notion of data independence, makes it easy to generate sub-queries against a variety of target databases. We had been doing this for some time with in-house databases, but are now using the FTP link to SRS to query remote databases.

One refinement of Karp's schema is that we strongly believe in an object-based conceptual model, although we are very happy to have target databases in flat-file or relational format, provided they are accessible selectively through external identifiers (keys). This is the basis on which MULTIBASE was developed, using the Daplex language, which we continue to use. We differ from MULTIBASE in that our underlying architecture is in Prolog and uses a theoretically well-founded intermediate form (list comprehensions or ZF expressions) as the basis for optimisation [13] [3].

This work raises interesting applications of distributed query optimisation and ordering of query execution.

The work we still have to do, as in Karp's proposal, is to make the system extensible, so that it can reason about which databases to use. This is relatively easy to do through a knowledge base of databases expressed in Prolog. However, we would plan to generalise concepts such as "swissprot_entry" so that they are more generic and mapped to alternative databases. Ultimately we anticipate a "facilitator" architecture which allows descriptors and interfaces to new biomolecular databases to be added, but this may need extensions and changes to SRS.

Acknowledgements

We would like to thank Thure Etzold (EMBL) and Jeroen Coppieters (EBI) for discussions and help with access to SRS and the FTP server at EBI, and John Fothergill (Molecular and Cell Biology, University of Aberdeen) for useful discussions.

References

- [1] Abola, E.E. and Prilusky, J.: 3DBbase v1.7 – A Relational Database for the PDB archives (1995) <http://www.pdb.bnl.gov/3DBbase/3DBbase.html>
- [2] Bernstein, F.C., Koetzle, T.F., Williams, G.J.B., Mayer, E.F., Jr., Bryce, M.D., Rodgers, J.R., Kennard, O., Shimanouchi, T. and Tasumi,

- M.: The Protein Data Bank: A Computer-based Archival File for Macromolecular Structures. *Mol. Biol.* **112** (1977) 535–542
- [3] Buneman, P., Davidson, S.B., Hart, K., Overton, C. and Wong, L.: A Data Transformation System for Biological Data Sources. In Dayal, U., Gray, P.M.D. and Nishio, S. (eds.) *Proceedings VLDB'95, Morgan-Kaufmann* (1995) 158–169
- [4] Buneman, P., Libkin, L., Suciu, D., Tannen, V. and Wong, L.: *Comprehension Syntax*. SIGMOD Record **23** (1994) 87–96
- [5] Chen, I.A. and Markowitz, V.M.: An Overview of the Object-Protocol Model (OPM) and OPM Data Management Tools, Information Systems, Pergamon Press (1995)
- [6] Durbin, R. and Thierry-Mieg, J.: *Syntactic Definitions for the ACEDB Data Base Manager* (1992)
- [7] Etzold, T. and Argos, P.: SRS an indexing and retrieval tool for flat file data libraries. *CABIOS* **9** (1993) 49–57
- [8] Gray, P.M.D. and Kemp, G.J.L.: Object-Oriented Systems and Data Independence. In Patel, D., Sun, Y. and Patel, S. (eds.) *1994 International Conference on Object Oriented Information Systems (OOIS'94)*, Springer-Verlag (1994) 3–24
- [9] Gray, P.M.D., Kulkarni, K.G. and Paton, N.W.: *Object-oriented databases: A semantic data model approach*. Prentice Hall International (UK) Ltd., Hemel Hempstead (1992)
- [10] Gray, P.M.D., Paton, N.W., Kemp, G.J.L. and Fothergill, J.E.: An object-oriented database for protein structure analysis. *Protein Engineering* **3** (1990) 235–243
- [11] Huysmans, M., Richelle, J. and Wodak, S.J.: SESAM: a relational database for structure and sequence of macromolecules. *Proteins: Structure, Function and Genetics* **11** (1991) 59–76
- [12] Islam, S.A. and Sternberg, M.J.E.: A relational database of protein structures designed for flexible enquiries about conformation. *Protein Engineering* **2** (1989) 431–442
- [13] Jiao, Z. and Gray, P.M.D.: Optimisation of methods in a navigational query language. In Delobel, C., Kifer, M. and Masunaga, Y. (eds.) *Proc. 2nd International Conference on Deductive and Object-Oriented Databases*, Springer-Verlag (1991) 22–42
- [14] Karp, P.D.: A Vision of DB Interoperation. Meeting on the Interconnection of Molecular Biology Databases (1995)
- [15] Kemp, G.J.L.: Protein modelling: a design application of an object-oriented database. In Gero, J.S. (ed.) *Artificial intelligence in design '91*, Butterworth-Heinemann Ltd. (1991) 387–406
- [16] Kemp, G.J.L., Iriarte, J.J. and Gray, P.M.D.: Efficient Access to FDM Objects Stored in a Relational Database. In Bowers, D.S. (ed.) *Directions in Databases: Proceedings of the Twelfth British National Conference on Databases (BNCOD 12)*, Springer-Verlag, New York (1994) 170–186
- [17] Kemp, G.J.L., Jiao, Z., Gray, P.M.D. and Fothergill, J.E.: Combining Computation with Database Access in Biomolecular Computing. In Litwin, W. and Risch, T. (eds.) *Applications of Databases: Proceedings of the First International Conference, ADB-94*, Springer-Verlag, New York (1994) 317–335
- [18] Landers, T. and Rosenberg, R.L.: An Overview of MULTIBASE. In Schneider, H.-J. (ed.) *Distributed Data Bases*, North-Holland Publishing Company (1982)
- [19] Paton, N.W. and Gray, P.M.D.: Optimising and Executing DAPLEX Queries using Prolog. *The Computer Journal* **33** (1990) 547–555
- [20] Rieche, B. and Dittrich, K.R.: A Federated DBMS-Based Integrated Environment for Molecular Biology. In *Proceedings of the Seventh International Conference on Scientific and Statistical Database Management* (1994)
- [21] Sander, C. and Schneider, R.: Database of Homology-Derived Protein Structures and the Structural Meaning of Sequence Alignment. *Proteins: Structure, Function and Genetics* **9** (1991) 56–68
- [22] Shipman D.W.: The Functional Data Model and the Data Language DAPLEX. *ACM Transactions on Database Systems* **6** (1981) 140–173
- [23] Wiederhold, G.: Mediators in the Architecture of Future Information Systems. *IEEE Computer* **25** (1992) 38–49