A Graphical Interface to Genome Multidatabases

Wang Chiew Tan
Ke Wang
National University of Singapore

Limsoon Wong
BioInformatics Centre & Institute of Systems Science

Formulating queries to access multiple databases can be a formidable task especially when many terms from various databases and complex constraints are involved. To specify a multidatabase query, the user usually has to search through documents for exact database terms and learn the multidatabase language. This report presents QUICK (QUery Interface to CPL-Kleisli), a graphical user interface to multiple databases. CPL (Collection Programming Language) is a high-level multidatabase language built on top of an open query system Kleisli. QUICK allows users to handle overwhelming information from different data sources in an intuitive and uniform manner. The query specification is reduced to specifying user's terms in his/her own world, selecting paths and specifying constraints in a graph. QUICK is able to automatically generate a CPL query that corresponds to the user's intent. Additional graphical functions are provided for the user to fine-tune the query generated.

A multidatabase system is a distributed system that acts as a front end to many autonomous DBMSs and a global layer above the autonomous DBMSs through a global schema or a multidatabase language. The global user can access information from multiple sources in the multidatabase system in a single straightforward request. However, the multitude of information available in multidatabase systems often impedes the user from quickly formulating a query. One reason is that the user often has to search through numerous manuals or documents for exact database terms in order to precisely specify a query. For example, it is difficult to be sure that employee identification number is termed "emp_id" and not "emp-id" or "employee_id" in the multidatabase. Although this problem also exists for single database systems, the magnitude of information in multidatabase environments makes it a more immediate problem. For instance, the schema documentation of GDB (Pearson, 1991) (Genome Data Base), a collection of databases providing human genome information, is well over 300 pages. In addition, multidatabase users are usually occasional users, in the sense that they use their home databases most of the time and access multidatabases only occasionally. As such, it is unreasonable to require multidatabase users to provide exact database terms. Some form of help should be given to reduce the user's effort in query specification.

Traditional textual query formulation requires syntactic and semantic knowledge of the language. A large number of graphical user interfaces exists for single database systems which make query specification more user-friendly. However, the issue of graphical user interfaces is not well-addressed in multidatabase systems, see Section RELATED WORK. In this paper, we present a prototype system QUICK (QUery Interface to CPL-Kleisli) to address this issue. The CPL (Wong, 1995) (Collection Programming Language) is a high level multidatabase language, built on top of an open query system called Kleisli (Wong, 1995), which can handle nested relations and structured files. QUICK is a graphical query interface which translates graphical specifications into CPL. The purpose of QUICK is to minimize the effort of end users in formulating queries for multidatabase systems. QUICK allows fast query formulation even with sporadic users having neither sufficient knowledge of query languages, nor extensive prior knowledge of database struc-

Manuscript originally submitted June 18, 1996; Revised December 27, 1996; Accepted January 17, 1997 for publication.

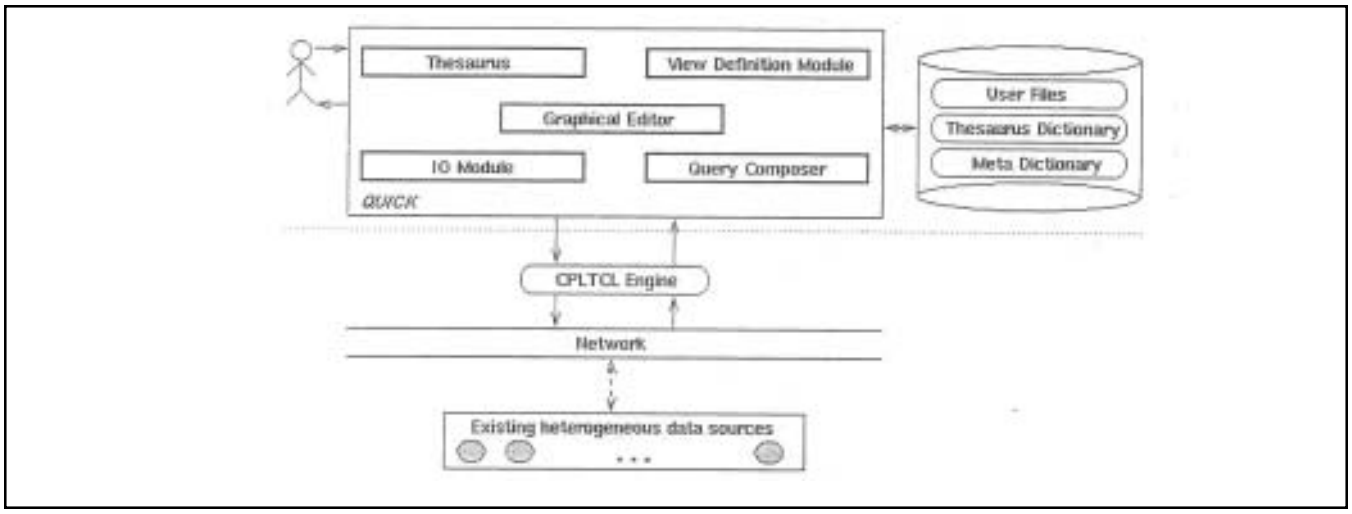

Figure 1: Overall Architecture of the System

tures. QUICK is written in Tcl 7.4/Tk 4.0 and it can be executed in any unix environment with X-Windows system and Tcl 7.4/Tk 4.0 installed. To run a query, CPLTCL, a variant of CPL for interfacing with TCL, is required to be installed.

Figure 1 shows the overall architecture of the system. QUICK is running on the Engine module CPLTCL that executes the query sent by QUICK. To replace CPLTCL by another multidatabase language, only the Query Composer and Meta Dictionary need to be replaced. The Thesaurus Dictionary provides a synonym mapping between user terms and database terms. The Meta Dictionary contains the schema information about views and frequently used predicates between views in the form of graphs. We consider general predicates that are not necessarily join. The Meta Dictionary can only be modified by the DBA. The editing of graphs during a particular user session have no effect on the Meta Dictionary. Instead, the editing result is saved onto a separate user file kept in the Data Store. The user can retrieve this file in a later session. Within the QUICK, there are five main modules. The Thesaurus is responsible for extracting corresponding database terms for the user specified terms. The View Definition is responsible for extracting the subgraph that contains the database terms returned by the Thesaurus. The Graphical Editor is the core module which supports essential graphical functions. The IO module is responsible for accessing session files in Data Store. Finally, the Query Composer generates a well-formed CPL query from a graphical specification.

There are three layers in the use of this system — the Thesaurus layer, Graph layer, and CPL layer. A user can enter the system at any of these layers. An expert user may like to enter the system at the lowest CPL layer by directly formulating a query in CPL but in the comfort of the graphical environment. A naive user may like to enter from the Thesaurus layer or Graph layer. The Thesaurus layer is good for users with minimal knowledge of databases and when only user terms are known. To map user terms to corresponding database terms, an interactive confirmation by the user may be needed and certain context information such as description of database terms and containing views and databases will be available to help with the confirmation. Based on confirmed database terms, the View Definition module retrieves the relevant portion view and schema definitions from the Meta Dictionary and presents it to the user in the form of a subgraph. Then the user proceeds to formulate queries using graphical interface functions provided by QUICK. The user can also choose to skip the Thesaurus layer and work with the entire graph or retrieve a subgraph by manually removing the irrelevant nodes — the Graph Layer. This is suitable for users with some knowledge of the underlying databases.

Though the use of QUICK in this paper is based on the Genome databases, QUICK is a generic interface for general multidatabase applications. For a new application running on CPL, only new Meta Dictionary and the Thesaurus Dictionary need to be created (by the DBA); for a new application running on a multidatabase language other than CPL, the Query Composer also needs to be substituted. The rest of this paper is organized as follows. In the next section, we review related interface work on multidatabase systems. In the section BIOLOGICAL DATA SOURCES, we describe the example biological databases used in this paper. In the section AN APPLICATION WITH QUICK, we show how a multidatabase query can be formulated with our prototype system QUICK using some genome databases. The conclusion is given in the last section.

Related Work

Most multidatabase research projects have emphasized on schema conflict resolution, query optimization, query processing and concurrency control. Query formulation for multidatabase systems has been predominantly textual. For

example, see Grant (1993) or Krish (1991). Most aim to perform query transformation and optimization, but have much neglected the graphical user interface aspect of multidatabase systems.

Graphical user interface, which has evolved from textual languages to the WIMP (Windows, Icons, Menus, Pointing devices) metaphor, is a mature research area for single database systems. A large number of graphical user interfaces already exist, mostly based on the popular relational databases (for example, SUPER (Spaccapietra and Tari, 1991) and CANDID [Schneider, 1991]). There has been a growing number of graphical user interfaces for object-oriented databases, e.g., Jun and Yoo (1994); Mong (1994). Most of this interface work focuses on schema editing operations and navigations, and translation from a graphical specification into a query program is mostly on a single database system based on the relational model.

The multidatabase graphical user interface in Towell (1995) uses the object-oriented approach to structure its databases. Standard retrieval functions are available for accessing objects. However, the user is still responsible to explicitly specify which databases, object classes and operations to invoke. Each instance of a class or attribute is returned in a separate window, which could be a problem if there are many instances to be returned. In our approach, the thesaurus helps naive users to immediately scope down to relevant views and relationships. Join and selection conditions can be added dynamically and the results are returned in a nested relation in one window.

Biological Data Structures

In this paper, the query specification by QUICK will be demonstrated through an example based on two biological data sources, namely, GDB (Genome Database) and GenBank (Burks, et al., 1992). GDB is a Sybase relational database that supports biomedical research, clinical practice and professional and scientific education by providing human gene mapping information. It is an international collaboration sponsored by biomedical funding agencies worldwide. GenBank is a genetic sequence database which is a collection of all known DNA sequences. We will be accessing GenBank via Network Entrez which is a retrieval program for a specially formatted text file that contains all information of GenBank in a certain release. An average record in this source has over 12 levels of nesting and over 150 different kinds of subobjects. These databases are part of the Human Genome Project whose primary aim is to identify all genes in the human genome and to sequence 3 billion bases of DNA that comprise the human genome. These biological sources are chosen for two reasons. First, it is an open research problem to integrate these genome databases well (see Goodman [1995]). Second, databases in the Human Genome Projects are among the most complex and diverse information sources in the world. The study on such applications will have general implications on a total solution.

Important genomic data exist in a number of distributed databases, e.g., GDB is hosted at The John Hopkins University at Baltimore, Maryland, whereas, GenBank is located at National Center for Biotechnology, National Institute of Health at Washington, D. C. They also exist in a number of different formats. For example, GDB is a Sybase relational database and GenBank is a data source consisting of structured files in the ASN.1 (Abstract Syntax Notation) format. These are just two of the databases in the Human Genome Project. As genomic data are generally complex structures in its natural form, many of them are best represented as free text or structured text files. There is a need to integrate these structured files with traditional databases.

Genomic data are not only diverse in type, but also large in size. A typical genetic database can consist of hundreds of tables and thousands of database terms. GDB version 5.5 contains close to 400 tables and approximately 1,300 database terms. A typical genetic query usually requires joins spanning relations in several distributed databases. Currently, there are many special-purpose HTML forms available on the internet such as the Entrez Browser (http://www3.ncbi.nlm.nih.gov/Entrez/index.html), Query Forms for searching data in GDB (http://gdbwww.gdb.org/gdb/shortcuts.html). As these interface tools only put certain aspects of the data online, they do not allow flexible access to all the data available. Although powerful query languages such as CPL offer flexible access to these data sources, they often require strong syntax and semantic knowledge of the language before a query can be formulated. QUICK is one step towards a more flexible and friendly interface to multiple sources in such applications.

An Application with QUICK

Since CPL is the target query language of QUICK, we give a brief introduction to it.

CPL uses the comprehension syntax (Buneman, et al., 1994). A CPL query has the form

{ *e* | *GF_1*, ..., *GF_n* }

called a *comprehension*. *GF_i* is of the form `y <- R` or is a condition such as "y.#name = z.#name". *e* is an expression for the result to be returned. A CPL query can be read in a way similar to tuple calculus: "The set of all e such that *GF_1*, ..., *GF_n*."

For example, in the expression

```
{(#name:m, #matric_no:n) |
    (#name:\m,#matric_no:\n,...) <- STUDENT};
```

`\m` is a simple pattern that matches anything and binds it to "`m`". Subsequent references to "`m`" will use this binding. The same goes for `\n`. "`...`" matches anything. # marks labels or

attribute names. Thus, the above expression matches each tuple in STUDENT partially.

A "primitive" is analogous to the concept of a function in programming languages. For example, the primitive below adds one to its argument, and the statement "addone(8);" invokes the primitive and the returned result will be 9.

```
primitive addone == \x => x + 1.
```

In CPL, the construct for sending a request *e* to a server *N* has the form `process e using N.` Also, comments are indicated by "!".

A Real Application

Consider the above four views in Figure 2 derived from databases GDB and GenBank. View GDB-locus contains the locus summary information. View GDB-object_genbank_eref contains the class description of genes. View GDB-locus_cyto_location contains the information about chromosomes. View GENBANK-entrez_summary contains the GenBank summary. The first three views are related through identifiers locus_id and object_id, and the second and fourth views are related through attributes genbank_ref and accession. Relating attributes share a common domain and are links for specifying across-view queries. An example of a CPL definition of a view, created by the DBA in the Meta Dictionary, is given below. In our convention, each view name has the form "DBname-Viewname". So, GDB-locus means the view locus derived from database GDB. It is also possible that a view is derived from more than one database. GENBANK-entrez_summary is one such example. It contains information from both GDB and GenBank, but the main information (summary) comes from GenBank. Therefore, we still call it GENBANK-entrez_summary. By leaving out "DBname", the user will be completely unaware of the location of each view. However, for clarity, we include database names as part of node names in the display.

The following shows an example of a CPL definition for the first view (GDB-locus) in Figure 2. The rest of the views, also defined in CPL, are kept in the Meta Dictionary and are automatically referenced by QUICK through involved database terms. That is, the user does not need to input them at all.

```
primitive GDB-locus ==
  { (#locus:
        (#locus_id: x.#locus_id,
         #locus_loc_summary:
            x.#locus_loc_summary,
          #locus_name: x.#locus_name,
                        #locus_symbol:
x.#locus_symbol)) |
     \x <- process "select * from locus l
where 1=1" using gdb };
```

Suppose that we wish to answer the following query:

View Name	Attributes	Attribute Description
GDB-locus	locus_id	Unique internal identifier for a locus.
	locus_loc_summary	Summary listing of the cytogenetic location for the locus.
	locus_name	HGM-approved name given to describe this locus.
	locus_symbol	HGM-approved symbol given to describe this locus.
GDB-object_genbank_eref	genbank_ref	GenBank accession number, that is, an external identifier.
	object_class_key	Number key indicating the type of database object. For locus object, this number is 1.
	object_id	Unique internal identifier for this object.
GDB_locus_cyto_location	loc_cyto_band_end	Cytogenetic band at the qter end of the locus location.
	loc_cyto_band_start	Cytogenetic band at the pter end of the locus location.
	loc_cyto_chrom_num	Chromosome number on which the locus is located.
	locus_id	Unique internal identifier for a locus.
GENBANK_entrez_summary	accession	Accession number.
	uid	Internal identifier used by GenBank.
	title	Name for this accession.

Figure 2: The views of each node in the subgraph of Figure 4

Figure 3: Part of chromosome 20 showing the p13 band.

Retrieve the GenBank summary and locus summary information about human genes on chromosome 20p13. The steps to specify the query using QUICK are described below.

Step 1. Specify the user terms.

The user enters terms in his/her own world through the Thesaurus module. For the above query, user terms are entered through the following SQL-like statement

```
SELECT summary, locus, human genes, chromosome
WHERE chromosome=20 and band start=p13
```

where "locus", "human genes", "summary", "chromosome", "band start" are user terms, which are mentioned, explicitly or implicitly, in the query either as the data to be retrieved or as the constraint of such data. If the user further knows that this information is contained in databases "GDB" and "GenBank", he/she may enter these database names in a "WITHIN" sub-statement. Both "WHERE" and "WITHIN" are optional.

There are important differences between a standard SQL statement and the above SQL-like statement. First, all terms in the above statement are user terms, not database terms. Second, the above statement does not have the "FROM" sub-statement because the user is not required to know the views or nodes containing the required data. In other words, through the above statement the user specifies what is wanted in his/her own terms as if there were a universal relation containing all data items. It is the job of the Thesaurus modules to map the user terms to database terms, with some degree of interaction with the user, and to decide and retrieve views containing the required data which are nodes in graphs contained in the Meta Dictionary. For this example, the Thesaurus module will retrieve (i) the node GDB-locus (containing the locus summary information) due to user terms "locus" and "summary", (ii) the node GDB-locus_cyto_location (contain-

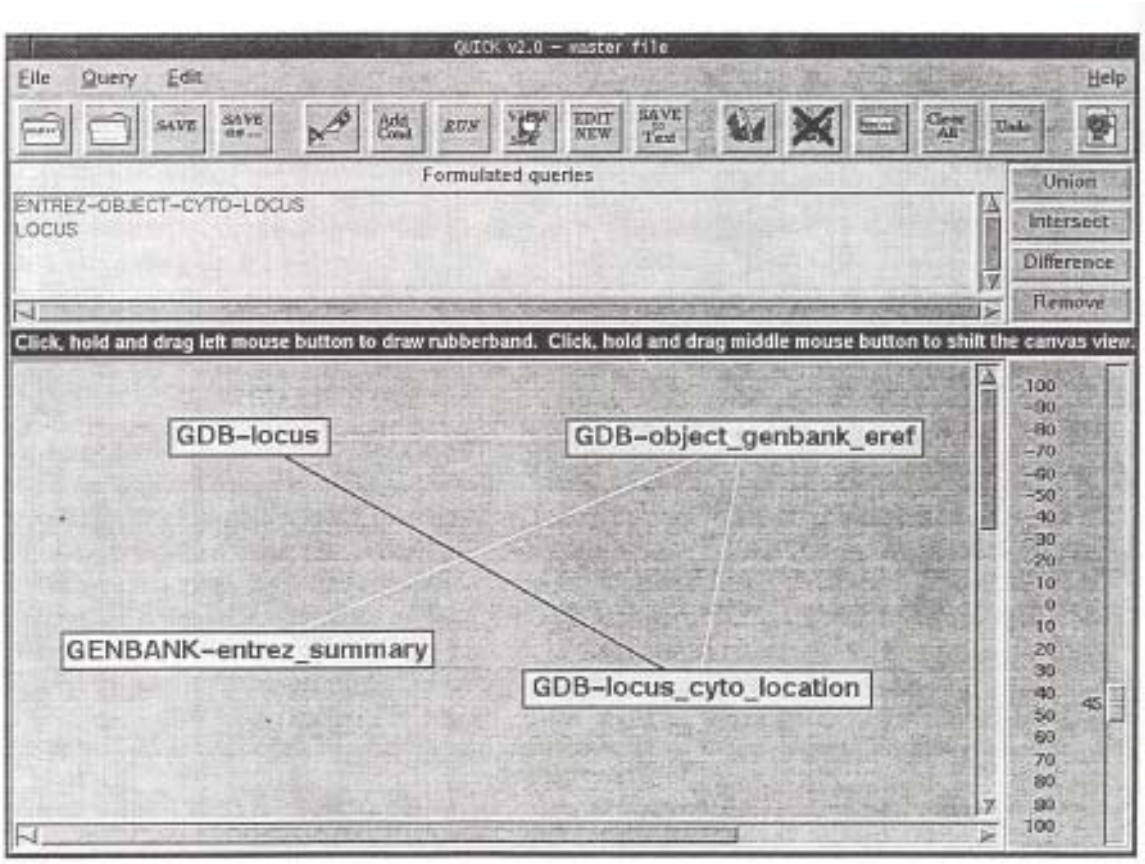

Figure 4: A display of subgraph in Graphical Editor

Selected edge	Selected predicates with definitions in CPL
GDB-locus, GDB-object_genbank_eref	JOIN_locus_genbank primitive JOIN_locus_genbank == (\L,\R) => L.#locus.#locus_id= R.#object_genbank_eref.#object_id andalso R.#object_genbank_eref.#object_class_key=1;
GDB-object_genbank_eref, GDB-locus_cyto_location	JOIN_object_cyto primitive JOIN_object_cyto == (\L,\R) => L.#object_genbank_eref.#object_id= R.#locus_cyto_location.#locus_id andalso L.#object_genbank_eref.#object_class_key=1;
GENBANK-entrez_summary, GDB-object_genbank_eref	JOIN_entrez_summary_object_genbank primitive JOIN-entrez_summary_object_genbank == (\L, \R) => L.#entrez_summary.#accession= R.#object_genbank_eref.#genbank_ref;

Figure 5: The Selected Edges and Selected Predicates

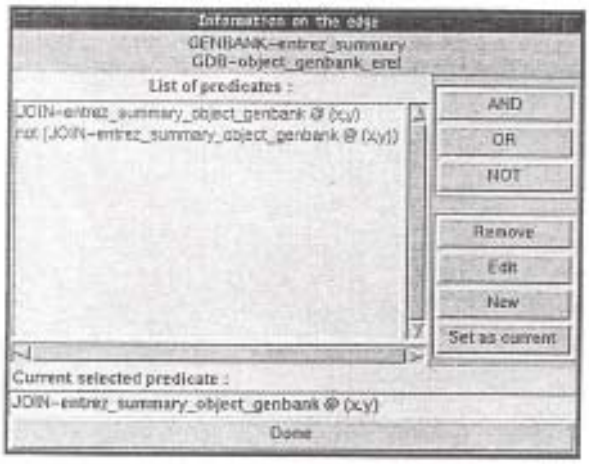

Figure 6: Information on the edge, by double clicking on the edge

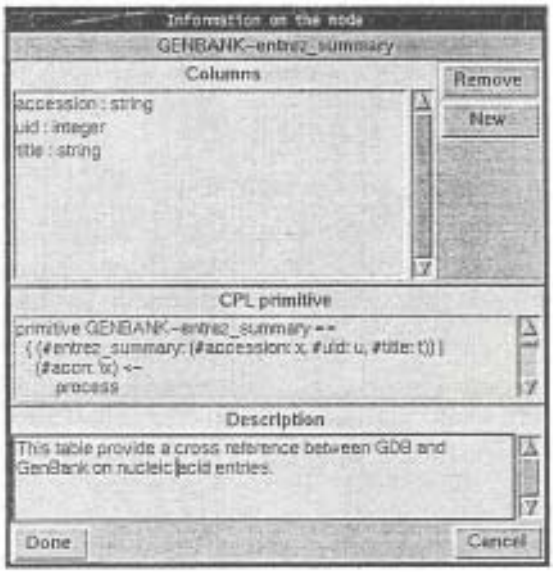

Figure 7: Information on the view, by double clicking on the node

ing the cytogenetic location information for locus objects) due to user terms "locus", "band start" and "chromosome", (iii) the node GDB-object_genbank_eref (containing an attribute for restriction to human genes) due to user term "human genes". Since node GENBANK-entrez_ summary provides a cross reference between GDB and GenBank on nucleic acid entries and contains information about locus, it is also returned. Corresponding database terms in these nodes and their information are presented to the user for confirmation. The confirmed database terms are then passed to the View Definition module which will extract a subgraph to be displayed on the Graphical Editor.

Step 2. Edit the retrieved subgraph.

The subgraph retrieved is displayed in Figure 4. A list of pre-defined predicates is associated with an edge. The CPL definitions of views for nodes and definitions of predicates for edges can be examined by clicking on the edge or node, such as in Figure 6 and Figure 7. Frequently used predicates on edges are maintained in the Meta Dictionary. However, predicates can be added or deleted and unwanted nodes can be deleted during a user session. Such updates are local to the separately stored session file; the underlying graphs in the Meta Dictionary remain unchanged. From the displayed subgraph the user will select edges and predicates on edges to compose the query. For the above query, three selected predicates are shown in Figure 5. When the graphical editing is

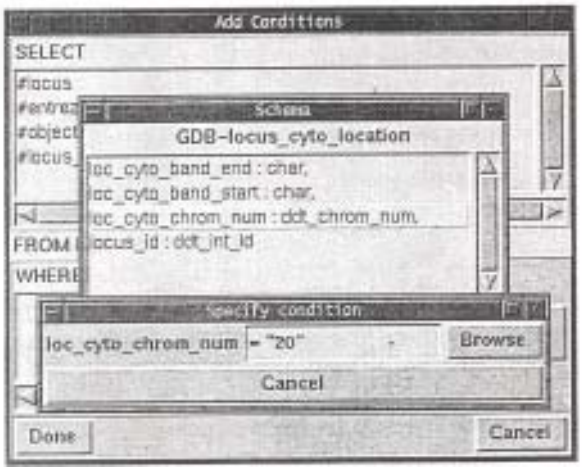

Figure 8: Adding the first condition

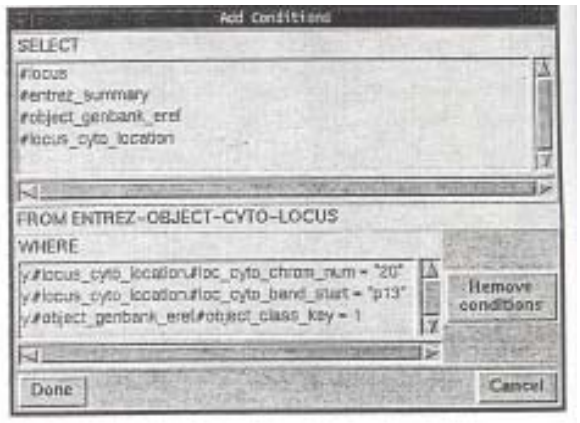

Figure 9: All conditions added

completed, the query is formulated with the click of a button. A name can be given to a saved query for later references; in our case, ENTREZ-OBJECT-CYTO-LOCUS.

Step 3. Specifying additional conditions.

The additional conditions that the chromosome is no. 20, that cytogenetic band at the pter end of the locus location is "p13", and that only human genes are of interest, are specified by selecting the "Add Condition" function in the "Query" menu on the upper-left part of the window. This window displays the query in the familiar SQL format. For example, Figure 9 shows the selection of four attributes from the intermediate view ENTREZ-OBJECT-CYTO-LOCUS formulated in Step 2 with additional conditions appended to the WHERE portion of the SQL window. The resulting CPL query, which is generated automatically by QUICK, is shown below.

```
! Connects to databases GDB and Entrez (in
GENBANK) with 1 connection
! each.
connect-to-gdb(1);
connect-to-entrez(1);

! CPL definitions of nodes and predicates
involved are produced here
! but not shown.

! The first intermediate view between
GENBANK-entrez_summary and
! GDB-object_genbank_eref generated by the
Query Composer.
primitive ENTREZ-OBJECT == {
     ( #entrez_summary: x.#entrez_summary,
                    #object_genbank_eref:
y.#object_genbank_eref ) |
     \x <- GENBANK-entrez_summary,
     \y <- GDB-object_genbank_eref,
     JOIN-entrez_summary_object_genbank(x,y)
};

! The second intermediate view between
ENTREZ-OBJECT and
! GDB-locus_cyto_location generated by the
Query Composer.
 primitive ENTREZ-OBJECT-CYTO == {
     ( #entrez_summary: x.#entrez_summary,
                    #object_genbank_eref:
x.#object_genbank_eref,
                    #locus_cyto_location:
y.#locus_cyto_location ) |
     \x <- ENTREZ-OBJECT,
     \y <- GDB-locus_cyto_location,
     JOIN_object_cyto(x,y)
};

! The third intermediate view (also the final
view) between
! ENTREZ-OBJECT-CYTO and GDB-locus generated by
the Query Composer.
! The last two conditions are user specified
conditions through
! graphical manipulations.

primitive ENTREZ-OBJECT-CYTO-LOCUS == {
     ( #locus: x.#locus,
       #entrez_summary: y.#entrez_summary,
                    #object_genbank_eref:
y.#object_genbank_eref,
                    #locus_cyto_location:
y.#locus_cyto_location ) |
     \x <- GDB-locus,
     \y <- ENTREZ-OBJECT-CYTO,
     JOIN_locus_genbank(x,y),
   y.#locus_cyto_location.#loc_cyto_band_start
= "p13",
   y.#locus_cyto_location.#loc_cyto_chrom_num
= "20"
};

! The invocation statement.
ENTREZ-OBJECT-CYTO-LOCUS;
```

Notice that the Query Composer processes selected edges in an order that may be indepedent of the user's selection order. Also, predicates defined for the edges can be more general than simply natural join predicates shown in the example here. For example, we can define "is-blast-homolog-of" as a predicate which executes the NCBI BLAST[1] (National Center for Biotechnology Information Basic Local Alignment Search Tool) program to check whether a given gene is homologous[2] to another. Although predicates are general and there are many ways in which the selected edges can be processed, the query result is always unique due to the way we structure the graph. The CPL engine also comes with an optimizer which will automatically avoid materializing intermediate views as well as migrating joins, selections and projections on GDB to the remote server (see Ngu, et al. (1993); Buneman, etal. (1995)). Therefore, QUICK only needs to generate queries by processing the selected edges in any order.

Step 4. Getting the result.

The generated query can be executed with the click of a button. The result is returned in a window as a nested relation. An example of the result is shown below.

```
{ ...
   (locus: (locus_name: "centromere protein B
(80kD)",
            locus_symbol: "CENPB",
            locus_id: 58,
            locus_loc_summary: "20p13"),
  entrez_summary:
           (accession: "X55039",
            uid: 29860,
            title: "Human hCENP-B gene for cen-
tromere
                    autoantigen B (CENP-B)"),
  object_genbank_eref:
           (genbank_ref: "X55039",
            object_class_key: 1,
            object_genbank_id: 66951,
            object_id: 58),
  locus_cyto_location:
           (loc_cyto_band_start: "p13",
            loc_cyto_band_end: "",
            locus_id: 58,
            loc_cyto_chrom_num: "20"))
  ... }
```

In the above example, the user's effort of specifying a query is reduced to three steps, that is, specify user terms, select nodes or edges at the confirm of their on-line definitions, and specify additional conditions. Thesaurus and browsing capabilities help the user to quickly and easily recall exact terms. Any (generic) joins between nodes will be depicted graphically. Most predicates needed for join conditions will probably be already available or it can be easily incorporated into the graph. With QUICK, minimal knowledge of underlying databases and CPL are required because the process of transacting a graphical specification to a CPL query is automated.

Other than the features shown above, a great deal of effort is made towards a user-friendly graphical editor. Here are a few of them. The top menu bar of Figure 4 contains options "File", "Query", "Edit", and "Help". QUICK allows to save the current session and retrieve a saved session for further editing work in a later time. The "File" menu contains commands for saving and retrieving a session file. After the user finishes the graphical specification of the query, he/she may choose to translate the specification to a named CPL query, or edit the query manually within QUICK, or run the query. These functions are contained in the "Query" menu. Named queries can be further manipulated using the set functions "Union", "Intersect" and "Difference" displayed on the right side of Figure 4. The menu "Edit" contains additional graphical functions. Among them, "Abstract" and "Unabstract" allow the user to collect a set of nodes together under one node or reverse the operation, thus enabling the user to hide away irrelevant portions for a while and put them back on the screen when required; "Zoom In" and Zoom Out" zoomed in or out a portion of a graph in a separate window; "Clear All Selection" cancels selection of nodes, edges, and predicates; "Undo" undoes the most recent graph operation. Nodes and edges can be moved around or deleted and the graph can be scaled in size to a level comfortable to the user. The "Help" option will bring up the help information on various topics in the form of hypertext links. Finally, most frequently used functions in "File", "Query", "Edit" menus are explicitly displayed below these menus so that only a single click is needed to use these functions.

Conclusion

A preliminary prototype interface to multiple autonomous heterogeneous databases, called QUICK, is implemented. QUICK translates a graphical specification into well-formed CPL primitives that can be run on the CPL-Kleisli system. Graphical functions are supported to allow the generated query to be enhanced or modified to suit specific needs. A practical application of this prototype on heterogeneous genome data sources is presented. QUICK is one step towards integrating and querying heterogeneous genome databases.

Due to the nature of interface work, QUICK does not have much formal theory. However, several principles of designing a user-friendly interface have been addressed : It reduces the need for the user to specify the exact database terms in order to formulate a multidatabase query. It removes the need for the user to fully grasp a multidatabase language. Also, a graphical representation and manipulation, transpar-

ent of its distributed nature, is used.

Many experiments show that QUICK has been fairly efficient in helping the user to formulate a correct query quickly. A few reasons contribute to its efficiency. First, the minimum need for the users to find out the exact database terms of data items in databases proves to be useful especially in the case of the biological data sources which often contain large and complex data. The second reason is that automatically generated queries remove the need for the user to specifiy the query using the underlying multidatabase query language almost entirely. This means less syntax errors and debugging for the user. Help and explanations are also easily available in the interface. No formal training needs to be given for the user to be proficient in using QUICK.

As the future work, we plan to conduct more intensive and broad experiments with QUICK on large and real world applications. We also intend to study and extend the expressiveness of queries that can be formulated in QUICK. The ultimate goal of QUICK is to provide a user-friendly interface for querying large heterogeneous databases such as genome databases.

Endnotes

[1] BLAST is a heuristic search algorithm employed by BLAST programs. The BLAST programs are tailored for sequence similarity seaching—for example to identify homologs to a query sequence.

[2] Two sequences are homologous if they are "similar."

References

Buneman, Peter, Susan Davidson, Kyle Hart, Chris Overton, and Limsoon Wong (1995). A Data Transformation System for Biological Data Sources. In *Proceedings of 21st International Conference on Very Large Data Bases*, 158-169, Zurich, Switzerland.

Buneman,Peter, Leonid Libkin, Dan Suciu, Val Tannen, and Limsoon Wong (1994). Comprehension Syntax. *SIGMOD Record*, 23(1):87-96.

Burks, C., M.J. Cinkosky, W.M. Fischer, P. Gilna, J.E Hayden, G.M. Keen, M.Kelly, D. Kristofferson, and J.D. Lawrence (1992). GenBank. *Nucleic Acids Research*, 20 Supplement:2065-2069..

Goodman, Nathan (1995). Research Problems in Genome Databases. In *PODS*, May.

Grant, J., W. Litwin, N. Roussopoulos, and T. Sellis. (1993). Query Languages for Relational Multidatabases. *The Int'l Journal on Very Large Data Bases*, 2(2):153-171, April.

Yong S. Jun and Suk I. Yoo. (1994). A Graph-based Graphical User Interface for Object-Oriented Databases. In *Proc. Of the 1994 Int'l Conf. On Object-Oriented Information System of Data*, 238-251, London, England.

Krishnamurthy, R., W. Litwin, and W. Kent (1991). Interoperability of Heterogeneous Databases with Schematic Discrepancies. In *Proc. First Int'l Workshop on Interoperability in Multidatabase Systems*, 144-151, 1991.

Monk, Simon (1994). A Graphical User Interface for Schema Evolution in Object-Oriented Database. In *Proc. of the 2nd Int'l Workshop on Interfaces to Database Systems*, 185-196, Lancaster University.

Ngu, Anne, Lingling Yan, and Limsoon Wong (1993). Heterogeneous Query Optimization using Maximal Subqueries. In *Proceedings of 3rd International Symposium on Database Systems for Advanced Applications*, 413-420, Taejon, Korea, April.

Pearson, P.L. (1991). The genome data base (GDB), a human genome mapping repositry. *Nucleic Acids Research*, 19:2237-2239.

Schneider , M. and C. Trepied (1991). Extensions for the graphical query language CANDID. In *Proc. of IFIP 2nd Working Conf. On Visual Database Systems*, 185-199, North Holland, Netherlands.

Spaccapietra, Stefano and Zahir Tari (1991). Super: A comprehensive approach to Database Visual Interfaces. In *Proc. of IFIP 2nd Working Conf. On Visual Database Systems*, 365-380, North-Holland, Netherlands.

Towell , Elizabeth R. and William D. Haseman (1995). Implementation of an Interface to Multiple Databases. *Journal of Database Management*, 13-21, Spring .

Wong, Limsoon (1995). *The Collection Programming Language Reference Manual. The Kleisli Query System Reference Manual.* Institute of Systems Science, Heng Mui Keng Terrace,