K2/Kleisli and GUS: Experiments in integrated access to genomic data sources

by S. B. Davidson
J. Crabtree
B. P. Brunk
J. Schug
V. Tannen
G. C. Overton
C. J. Stoeckert, Jr.

The integrated access to heterogeneous data sources is a major challenge for the biomedical community. Several solution strategies have been explored: link-driven federation of databases, view integration, and warehousing. In this paper we report on our experiences with two systems that were developed at the University of Pennsylvania: K2, a view integration implementation, and GUS, a data warehouse. Although the view integration and the warehouse approaches each have advantages, there is no clear "winner." Therefore, in selecting the best strategy for a particular application, users must consider the data characteristics, the performance guarantees required, and the programming resources available. Our experiences also point to some practical tips on how database updates should be published, and how XML can be used to facilitate the processing of updates in a warehousing environment.

With the recent completion of a rough draft of the human genome, with the finished sequence for Drosophila, *C. elegans*, and yeast (among others), and with numerous other sequencing projects in progress, vast amounts of genomic data have become available for further refinement and analysis. Moving past DNA (deoxyribonucleic acid) sequences, researchers are interested in the corresponding protein sequences, their structure, and function. Beyond sequences, researchers wish to understand the "space" and "time" dimensions of genes, as for example, what genes are expressed in which tissues and during what stages of development. While these and other questions can only be answered exactly by direct experimentation, very often insights can be gained by accessing the tremendous amount of genomic information that is available on line.

Suppose a researcher seeks to discover genes involved in a multigenic neurological disorder such as bipolar schizophrenia. High-throughput analysis of gene expression patterns yields expression profiles of several thousands of potentially involved genes. Analysis of the profiles reveals hundreds of genes that represent candidate genes for the disorder. Experimentally analyzing all these candidates is prohibitively expensive. The researcher must therefore prioritize the candidates and start the search with the most promising ones. Thus, the researcher would access databases such as GenBank,[1] SWISS-PROT,[2] and OMIM[3] in order to determine, by genetic mapping, which of these genes are located in human chromosomal regions associated with schizophrenia, or which of these genes are located in human chromosomal regions syntenic to those in model organisms (e.g., mouse or rat) where related nonhuman neurological diseases have been mapped. Note that because GenBank, EMBL,[4] and DDBJ[5] form a consortium and exchange data regularly, EMBL or DDBJ could have been used instead of GenBank in the query above.

Unfortunately, although most genomic information that researchers wish to access is available on line, it does not all reside in one database and in one location. Rather, it is spread over multiple data sources using a variety of data models and data formats, presenting a variety of languages and interfaces for data

0018-8670/01/$5.00 © 2001 IBM

retrieval. Many of the data sources are not implemented using conventional database management systems (such as relational databases), but use formatted files with specialized GUIs (graphical user interfaces) and retrieval packages (e.g., SRS[6] and ACeDB[7]). These formats have been adopted in preference to database management systems (DBMSs) for several reasons. The data are complex and not easy to represent using the relational data model. Typical structures include sequential data (lists) and deeply nested structures (trees). This complexity would argue for the use of object-oriented database systems, but these have not met with success because of the constant need for database restructuring.[8] As new experimental techniques are discovered, for example, new data structures are needed to record details particular to those techniques. Furthermore, formatted files are easily accessed from languages such as Perl and C, and a number of useful software packages exist, for a variety of platforms, that work with these files.

As an example of the genomic data that are available on line, consider the SWISS-PROT entry shown in Figure 1. Each line begins with a two-character code, which indicates the type of data contained in the line. Each database entry is identified by an accession number AC and is timestamped by up to three dates, DT: the create date is mandatory, whereas the last sequence update and last annotation update appear only if the sequence or annotation has been modified since the database entry was created. The sequence SQ (list of amino acids) appears at the end of the entry. Citation information (bibliographical references) are lines beginning with R. Taxonomic data OC contain a description of the biological source of the protein. Database references DR contain explicit links to entries in other databases: EMBL (annotated nucleotide sequence database); HSSP (homology derived secondary structure of proteins); WORMPEP (predicted proteins from the *Caenorhabditis elegans* genome sequencing project); INTERPRO, PFAM, PRINTS, PROSITE (databases of protein families and domains) among others. Annotation information—which is obtained from publications reporting new sequence data, review articles, and external experts—is mainly found in the feature table FT, keyword lines KW, and comment lines CC (not shown). Note that the bibliographical references are *nested* structures; there are two references, and the RP, RC, RA, and RL fields are repeated for each reference. Similarly, the feature table can be thought of as a nested structure in which each line contains a start and end position (e.g., 14 to 21) and a type of feature (e.g., NP_BIND). The entry is designed to be easily read by a human and structured enough to be machine parsed. However, several lines still contain a certain amount of structure that could be separated out during parsing. For example, the author list is a string, which could be parsed into a list of strings so as to be able to index into the individual authors. Similarly, the taxonomic data are strings spread over several lines that could again be parsed into a list.

The heterogeneity of the data sources, together with their frequently unconventional implementation, makes accessing genomic data across multiple data sources extremely difficult. Researchers—like the one studying bipolar schizophrenia—are faced with the problem of integrated access to heterogeneous data sources. What software should they use to gain access to the data? How accessible is this software relative to their (often limited) computer expertise? Should they access the data where the data are, or should they import the data into their own specialized database? What are the trade-offs between these approaches? In the next section we describe three approaches to integrating access to heterogeneous data sources.

Three approaches

Over the past ten years, a variety of techniques have been developed within the genomic community to provide integrated access to multiple, heterogeneous data sources. Several *link-driven federations* have been created, in which users start by extracting entries of interest in one data source and then hop to other related data sources via Web links that have been explicitly created by the developers of the system. SRS,[6] LinkDB,[9] and GeneCards[10] are examples of this approach. Systems implementing a *view integration* approach have also emerged within the community, such as K2/Kleisli[11,12] and OPM.[13] In this approach, the schemas of a collection of underlying data sources are merged to form a global schema in some common model (such as the relational, complex value, or object-oriented model). Users query this global schema using a high-level query language, such as SQL,[14] OQL,[15] or CPL.[16] The system then determines what portion of the global query can be answered by which data source, ships local queries off to the appropriate data source, and combines the answers from the various data sources to produce an answer to the global query. These view integration systems can also be used to create an instantiation of the global schema, commonly referred to as a *warehouse*. Once this instantiation has been set

Figure 1 Sample SWISS-PROT entry

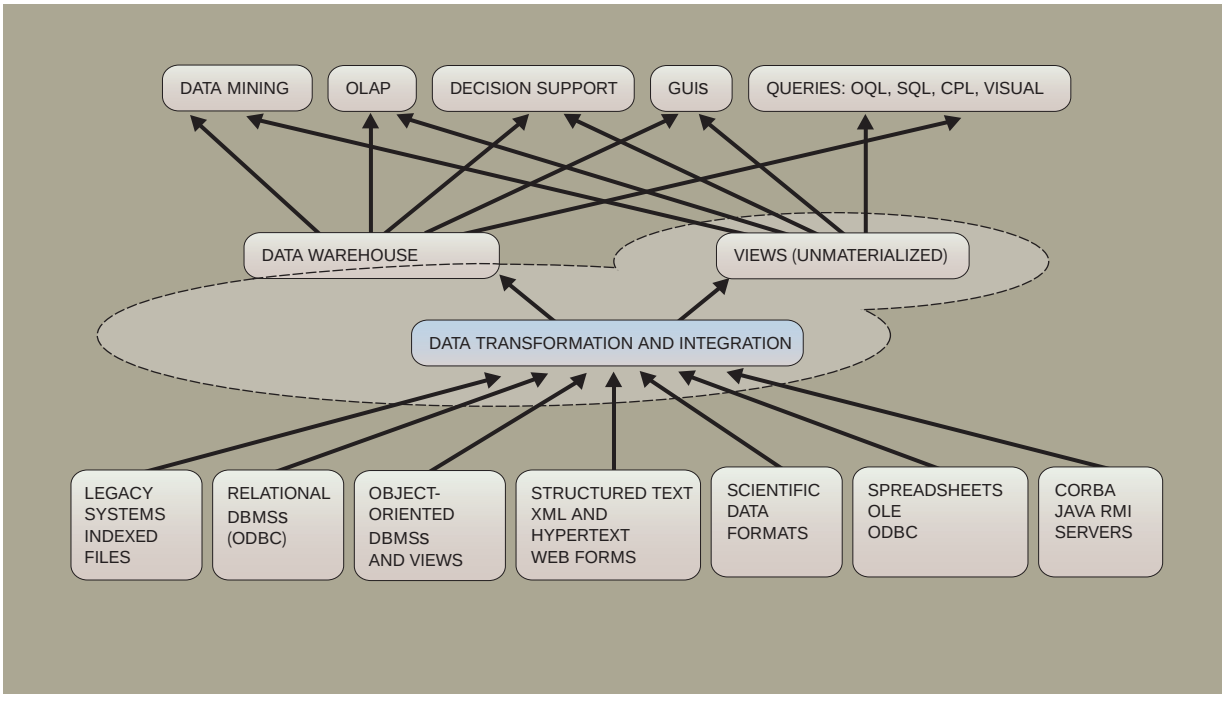
```
ID  EF1A_CAEEL     STANDARD;      PRT;   463 AA.
AC  P53013;
DT  01-OCT-1996 (Rel. 34, Created)
DT  01-OCT-1996 (Rel. 34, Last sequence update)
DT  15-DEC-1998 (Rel. 37, Last annotation update)
DE  ELONGATION FACTOR 1-ALPHA (EF-1-ALPHA).
GN  (EFT-3 OR F31E3.5) AND RO3G5.1.
OS  Caenorhabditis elegans.
OC  Eukaryota; Metazoa; Nematoda; Chromadorea; Rhabditida; Rhabditoidea;
OC  Rhabditidae; Peloderinae; Caenorhabditis.
RN  [1]
RP  SEQUENCE FROM N.A. (EFT-3).
RC  STRAIN=BRISTOL N2;
RA  Favello A.;
RL  Submitted (NOV-1995) to the EMBL/GenBank/DDBJ databases.
RN  [2]
RP  SEQUENCE FROM N.A. (RO3G5.1).
RC  STRAIN=BRISTOL N2;
RA  Waterston R.;
RL  Submitted (MAR-1996) to the EMBL/GenBank/DDBJ databases.
DR  EMBL; U51994; AAA96068.1; -.
DR  EMBL; U40935; AAA81688.1; -.
DR  HSSP; P07157; 1AIP.
DR  WORMPEP; F31E3.5; CE01270.
DR  WORMPEP; RO3G5.1; CE01270.
DR  INTERPRO; IPR000795; -.
DR  PFAM; PF00009; GTP_EFTU; 1.
DR  PRINTS; PR00315; ELONGATNFCT.
DR  PROSITE; PS00301; EFACTOR_GTP; 1.
KW  Elongation factor; Protein biosynthesis; GTP-binding;
KW  Multigene family.
FT  NP_BIND      14     21       GTP (BY SIMILARITY).
FT  NP_BIND      91     95       GTP (BY SIMILARITY).
FT  NP_BIND     153    156       GTP (BY SIMILARITY).
SQ  SEQUENCE   463 AA;  50668 MW;  12544AF1F17E15B7 CRC64;
          MGKEKVHINI VVIGHVDSGK STTTGHLIYK CGGIDKRTIE KFEKEAQEMG KGSFKYAWVL
          DKLKAERERG ITIDIALWKF ETAKYYITII DAPGHRDFIK NMITGTSQAD CAVLVVACGT
          GEFEAGISKN GQTREHALLA QTLGVKQLIV ACNKMDSTEP PFSEARFTEI TNEVSGFIKK
          IGYNPKAVPF VPISGFNGDN MLEVSSNMPW FKGWAVERKE GNASGKTLLE ALDSIIPPQR
          PTDRPLRLPL QDVYKIGGIG TVPVGRVETG IIKPGMVVTF APQNVTTEVK SVEMHHESLP
          EAVPGDNVGF NVKNVSVKDI RRGSVCSDSK QDPAKEARTF HAQVIIMNHP GQISNGYTPV
          LDCHTAHIAC KFNELKEKVD RRTGKKVEDF PKFLKSGDAG IVELIPTKPL CVESFTDYAP
          LGRFAVRDMR QTVAVGVIKS VEKSDGSSGK VTKSAQKAAP KKK
```

up, the global query can be answered using information in the warehouse rather than shipping off local queries to various data sources. We refer to this approach for integrated access to heterogeneous data sources as *warehousing*.

Figure 2 illustrates the connection between the view integration and the warehousing strategies. At the bottom of the figure multiple, heterogeneous data sources are shown. Above the data sources a software layer is shown that provides access to the underlying data sources. The dotted lines show that the software layer can then be used either for view integration access or as the tool for creating a data warehouse. In either the view integration or the warehouse strategy, a global query of the underlying data sources can be embedded in a variety of application programs and GUIs, shown at the top of the figure.

There are obviously trade-offs[17] between the link-driven federation, view integration, and warehouse approaches. The link-driven federation approach is very useful for the nonexpert user, since it relies almost entirely on a point-and-click interface. Most

Figure 2 View and warehouse integration



of the federation approaches also offer a limited retrieval language that can be quickly learned by nontechnical users. SRS, for example, allows users to specify logical combinations of regular expressions to index into fields of interest. There is also tremendous value in the linked connections. The approach is therefore very helpful for laboratories with little in-house computer expertise. However, the approach does not scale well. When a new data source is added to the federation, connections between its entries and entries of all existing federation data sources must be added, a task of complexity commonly referred to as the "N^2" problem. Furthermore, it is often the case that if a user is interested in a join between two data sources in the federation, he or she must manually perform the join by clicking on each entry in the first data source and following all connections to the second data source. (SRS avoids this by providing a linking operator that retrieves linked entries to a set of entries.) In contrast, a join can be expressed in a single high-level query in the view integration or warehouse strategies. In general the query languages supporting view integration or warehouse approaches are much more powerful languages and allow arbitrary restructuring of the retrieved data.

In the view integration or warehouse strategy, the user sees a global schema of the underlying data. To be useful, the schema should give the user the ability to connect various pieces of information. This can be done either by explicitly providing linking tables (as in the link-driven federation approach, but using some form of identifiers rather than hyperlinks), or by providing software to compute matches between data (such as homology or similarity above some threshold for sequence entries). The schema itself can be thought of as a set of integration queries over the union of the schemas of the underlying data sources, perhaps with additional linking tables provided by the integrator.

The remainder of this paper is organized as follows. We start by describing the K2/Kleisli view integration system for bioinformatics applications developed at the Penn Center for Bioinformatics (PCBI), used at SmithKline Beecham, and used to build the TAMBIS[18] system at the University of Manchester. We then discuss various practical issues associated with the warehousing approach before describing a particular warehouse, GUS (the Genomics Unified Schema). GUS forms the basis for several organism

and tissue specific research projects at the University of Pennsylvania and collaborating institutions; in particular, view applications have been developed in support of a *Plasmodium falciparum* database, a mouse and human gene index, and a database of genes expressed in the developing endocrine pancreas. The section on performance provides results of some performance studies involving K2/Kleisli and GUS. The section on conclusions contains final comments.

K2 relies on a set of data drivers, each of which handles a single type of data source.

K2/Kleisli

K2 is the latest incarnation of a distributed query system that we have been developing over the past seven years at the University of Pennsylvania. K2 is based on many of the same principles that guided the design of Kleisli,[12,19,20] its conceptual predecessor. Like Kleisli, the K2 system uses a complex value model of data. This model is one in which the "collection" types, i.e., sets, lists, and multisets (bags), may be arbitrarily nested along with record and variant (tagged union) types. Kleisli uses as its language the Collection Programming Language[21] (CPL), which was developed specifically for querying and transforming complex value data. Although equivalent in expressive power to SQL when restricted to querying and producing relational data, CPL uses a "comprehension"-style syntax,[16] which is quite different in style from SQL. This departure from the *de facto* query language standard has apparently made CPL less accessible to database professionals. Consequently the decision was made in K2 to support the more recent industry-standard query language OQL.[22] OQL uses the "select-from-where"-style syntax of SQL, but its semantics is that of comprehensions, just like CPL. K2 supports full OQL extended with variant (disjoint union) types.

The complex value data model of K2 also incorporates a new data type, that of "dictionaries." A dictionary is a function with an explicit *finite* definition domain. This allows the representation[22] of object-oriented classes as dictionaries whose domains model the class extents, i.e., sets of object identities. Dictionaries also allow a direct representation of Web-based data sources that allow the retrieval of information through HTML forms. K2 also differs from Kleisli in its implementation language; whereas Kleisli was written using Standard ML,[23] K2 is implemented primarily in Java** and makes use of several of the standard protocols and application programming interfaces (APIs) that are part of the Java platform,[24] including RMI[25] and JDBC**.[26]

The architecture of K2 is similar to that of a number of other view integration systems. K2 relies on a set of *data drivers*, each of which handles the low-level details of communicating with a single class of underlying data sources (e.g., Sybase** relational databases, Perl/shell scripts, the BLAST[27] 2.x family of similarity search programs, etc.). A data driver accepts queries expressed in the query language of its underlying data source. It transmits each such query to the source for evaluation and then converts the query result into K2's internal complex value representation. For data sources that support it, this is done on a tuple-by-tuple or object-by-object basis analogous to the demand-driven tuple processing paradigm of relational databases. Data drivers are also responsible for providing K2 with data source meta-data (i.e., types and schemas), which are used to type check queries.

Once a user's OQL query has been type checked, K2 must decompose it into subqueries that can be answered by the underlying data sources. Furthermore, it must rewrite the OQL query fragments, where necessary, into queries that the data sources can understand (since most will not support OQL directly). Both of these tasks are handled by the system's *query optimization module*, which is an extensible rule-based optimizer. The K2 optimizer performs query "deforestation," i.e., the elimination of intermediate collection results. Further, it performs rewrites to group operations by data source. Cost-based optimization, which we are also investigating, is an alternative commonly used in commercial relational database systems. However, the distributed environment in which the system must run does not lend itself well to accurate cost estimation. Note that any part of the query that the K2 optimizer is unable to ship to the underlying data sources will be executed by the K2 run-time system itself.

To illustrate how K2 is used, consider the following scenario in which GUS is queried in combination with

Figure 3 A sample K2 type that represents a simplified PubMed entry

```
simplified-pubmed-entry ::=
    ( abstract: <0: null, 1:string>,
      uid: <0: null, 1:long>,
      pmid: <0: null, 1:long>
      cit: (title:<0:unit, 1:{ <name:string, iso-jta:string, isbn:string, ...> } >,
            from: <journal: ..., book: ..., proc: ...>,
            authors: (names: <std: [ ... ], ml: [ string ], str: [ string ]>,
                      affil: ... )
      mesh: <0: null,
             1:{ (mp: bool, term: string,
                  qual: <0: null, 1:{ (mp: bool, subh: string) }>) }>,
      substance: <0: null,
                  1:{ (type: <nameonly: null, cas: null, ec: null>,
                              cit: <0: null, 1:string>,
                              name: string) }>, ... )

        () ::= record, [] ::= list, {} ::= set, <> ::= variant (tagged union),
     <0: null, 1: string> ::= a variant that represents an optional string value
          "..." ::= a portion of the type that has been omitted for brevity
```

the National Center for Biotechnology Information's (NCBI's) PubMed[28] database. GUS, which will be discussed in detail in a later section, contains expressed sequence tag (EST) assemblies that represent genes.

By using the mapping and expression data integrated by GUS, a scientist has identified a set of EST assemblies that represent candidate genes for an inherited disorder. Some of these assemblies correspond to known genes and some do not. In order to gain further insight into the function of the "unknown" genes, the investigator wants to find publications that reference the known genes and that mention a specific protein or substance known to be affected by the disorder. Annotated bibliographic data of this kind are not part of GUS, but can be found in NCBI's PubMed[28] database. Figure 3 shows a simplified version of the K2 type that describes an entry in PubMed as provided by the NCBI Network Entrez service.

Network Entrez provides a C language API to PubMed and several other databases, including GenBank, and uses ASN.1[29] to represent data and types. ASN.1 is a complex value data model much like that used by K2, and so the translation between the two is straightforward. A data driver for Network Entrez has been developed using its ASN.1/C API. The data driver appears in K2's OQL interface as a user-defined function that can be passed commands written using an *ad hoc* syntax. For example, the following OQL statement retrieves all the substances (e.g., proteins, enzymes, etc.) associated with a single PubMed reference:

```
K2> entrez("-g 20296074 -d m -r
Entrez-back.getmle.data.E.substance.E.name");
```

The result of executing the query would be echoed back as:

```
list("Heparin",
     "Complement 3d",
     "N-acetylheparin",
     "Glycoproteins",
     "Complement 9",
     "clusterin")

K2: optimized query in 0.0020 seconds.
K2: total elapsed time for request was 1.162
seconds.
```

In the preceding query, `entrez` is the OQL function that represents the Entrez data driver; "-g 20296074" specifies the Entrez ID of the PubMed reference; "-d m" specifies the PubMed/MEDLINE section of Entrez; and the "-r" flag gives an optional path expression that specifies which part(s) of the ASN.1 entry should be returned to K2. This syntax is difficult to remember, so we can use OQL's `define` directive to create a function that represents a very simple view on PubMed. In the following, "| |" is OQL's string concatenation operator:

```
define get-medline-substances(pmid) as
entrez(
  "-g " ||
  pmid || " -d m -r
  Entrez-back.getmle.data.E.substance.E.name"
  );
```

Combining these functions with the data on genes in the GUS data warehouse, we can list the references and substances associated with an EST assembly (predicted gene) with the following OQL view functions. The first, `GUS-transcript-seqs`, takes as input a GUS EST assembly ID and returns the accession numbers of the individual sequences (both ESTs and mRNAs) that make up the assembly:

```
define GUS-transcript-seqs(rnaId) as
select enaseq.source_id
from GUS_RNASequence rs,
     GUS_NAFeature naf,
     GUS_AssemblySequence aseq,
     GUS_ExternalNASequence enaseq
where rs.rna_id = variant(1: rnaId)
and rs.na_feature_id = naf.na_feature_id
and naf.na_sequence_id =
     aseq.assembly_na_sequence_id
and aseq.na_sequence_id =
     enaseq.na_sequence_id;
```

The second function, `GUS-transcript-pubmed-refs`, joins the relevant tables in GUS with PubMed, using the two functions that we have just defined. It calls `get-medline-substances` on each sequence in the EST assembly and returns a collection of records (the OQL "struct" clause), each of which contains a PubMed ID (`pmid`) and a list of substances. The function uses an additional mapping function, `accn-to-ids`, which takes as input the accession number of a sequence and returns the PubMed IDs of all the publications that reference that sequence. This function is implemented by a Perl script. Finally, note the use of OQL's "flatten" command to transform a nested collection (e.g., a set of sets) into a nonnested collection:

```
define GUS-transcript-pubmed-refs(rnaId) as
  select struct(pmid: pmid,
    substances: get-medline-substances(pmid))
  from flatten
    (select accn-to-ids("m " || accn)
    from GUS-transcript-seqs(rnaId)
    accn) pmid;
```

We can now call `GUS-transcript-pubmed-refs` on an assembly ID to get associated PubMed IDs and substances:

```
K2> GUS-transcript-pubmed-refs(101005);

bag((pmid: 9530155,
     substances: list("Neurotensin",
                      "Azacitidine",
                      "neuromedin N",
                      "Peptide Fragments")))
```

The preceding example can trivially be modified to retrieve MEDLINE abstracts, or to list only those EST assemblies linked to publications that also mention "neurotensin" (for example). More generally, we can incorporate data from any of the other sources for which we have data drivers. These include: metabolic and/or signaling pathway information from KEGG[30] and EcoCyc;[31] DNA sequence-related data from GenBank, GSDB,[32] and dbEST;[33] organism-specific data from MGD[34] and GDB;[35] sequence similarity searches using BLAST;[27] and data from any of the databases indexed by SRS.[6] As in our example, the data sources may include scripts and application programs (e.g., BLAST), as long as an appropriate data driver is available.

Thus far we have not mentioned the object-oriented capabilities of K2. An interesting aspect of the system is that integrated views may be defined not only by OQL functions (as in our simple example), but also by user-defined object-oriented classes. A new language, K2MDL, lets users describe new classes by specifying how their extents and attributes are computed from the underlying data sources. View classes so defined can then be queried using OQL, and K2 will compose the OQL queries with the K2MDL views, producing multisource queries in its internal language; the resulting queries are then normalized and decomposed by the optimizer. In using K2 to define views, the user has a range of options. At the highest level of abstraction, the user may define virtual classes that span several underlying databases. At the lowest, it is possible to use K2 to access the underlying data sources directly (as in our call to the "entrez" driver). Either way, a user or K2 system administrator is free to provide integrated views only for selected parts of the underlying data sources, such as those that are best understood or most frequently used. Those parts of the underlying data sources that have yet to be integrated in some view may still be queried and joined directly.

K2MDL combines ODL[15] (Object Definition Language) and OQL syntax. It defines the schema of a class in the view using ODL, and defines how the extent of the class and how the attributes of an object

are computed using OQL. The approach is related to O2Views.[36]

K2 is implemented as a multithreaded server that can handle multiple client connections. Clients communicate with the server using either RMI-IIOP (Remote Method Invocation, Internet Inter-Orb Protocol) or an *ad hoc* socket protocol. We have implemented a client that provides interactive command line access to the system (as shown in our examples), in addition to a set of client libraries that simplify accessing K2 from any Web application that uses Java servlets.[25]

Additional information on the system and further examples of parameterized queries can be obtained from the K2 Web site.[37]

Issues in warehousing

The view integration and warehouse strategies share the need for a common data model—relational, object-oriented, complex object, etc.—in which to represent the underlying data sources. They also share the need for an appropriate query language—SQL, OQL,[15] CPL[16]—in which to express the integrating queries. The difference between the approaches is whether there exists a physical copy of the integrated, global view. If so, then we have a warehouse. Otherwise, the view is not materialized and we are dealing with view integration. In the view integration approach, for each set of search parameters, the integration queries are specialized to these parameters and when executed result in the much smaller relevant part of the integrated view as needed to satisfy the query. This points to some advantages of the view integration approach: it has low initial cost, low maintenance cost, and the query result is always up-to-date.

The main advantage of the warehouse approach is that system performance tends to be much better (this is illustrated in the section on performance). Indeed, query optimization can be performed locally and communication latency to access various data sources is eliminated. System reliability is also better since there are fewer dependencies on network connectivity or the availability of the underlying data sources. (Data sources may go down, or become overloaded and temporarily unable to answer queries.) It is also easier to enforce any interdatabase constraints.[38,39] Another advantage of warehousing is that while the underlying data sources may contain errors, often the only feasible way for the integrated view to have correct data is to keep a separate cleansed copy. Furthermore, the researcher may have additional information—or *annotations*—to add to the integrated view, which is either entered manually or with the help of a software package guided by a human. The "added-value" of corrected and annotated data stored in the data warehouse is significant.

However, a warehouse requires maintenance as the underlying data sources change, and this raises a number of practical problems:

1. How can we detect that the underlying data sources have changed?
2. How can we automate the refresh process?
3. How can we track the origins or "provenance" of data?

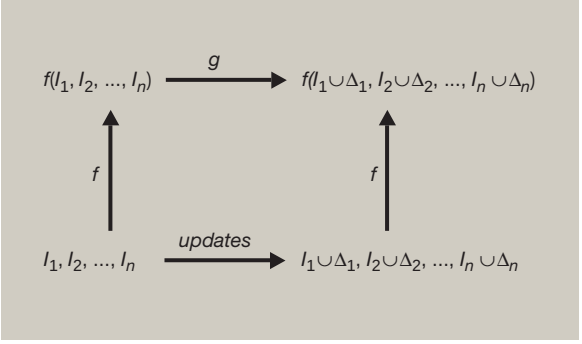
Detecting change in a data source. Part of the problem of change detection is deciding whether a push or a pull technology should be used. In a *push* technology, users register queries with the underlying data source and request explicit notification when a change occurs that matches the query; this is also known in the database literature as "continuous" (or "continual") queries.[40–43] In a *pull* technology, the user periodically polls the underlying data source to see if a change of interest has occurred. Note that a push technology requires the underlying data source to be capable of processing some form of triggers, and to be willing and able to send such notification.

Genomic data sources are just beginning to offer push capabilities. For example, SWISS-PROT offers a service called "Swiss-Shop" that allows keyword-based and sequence/pattern-based requests.[4] When new sequence entries are entered that are relevant to the request, they are mailed electronically to the requesting user. This occurs at weekly intervals as the new update files are generated. Most of the other major genomic data sources, however, do not yet offer push services.[44] Warehouse developers within the genomics community will therefore probably have to rely on pull technologies in many cases.

Another aspect of change detection is finding out exactly how the underlying data source has changed. In the context of genomic databases, this is complicated by the fact that updates are typically propagated in one of three ways:

1. Producing periodic new versions that can be downloaded by the user community

Figure 4 View maintenance problem



2. Timestamping data entries so that users can infer what changes have occurred since they last accessed the data
3. Keeping a list of additions and corrections; each element of the list is a complete entry. The list of additions can be downloaded by the user community.

None of these methods precisely describes the minimal changes that have been made to the data.

As an example, suppose that a warehouse stores a portion of SWISS-PROT in a *normalized*[45] relational database, which requires that all fields in a table be single-valued facts about the key of the table. In the resulting relational schema, an entry is split over about 15 tables. As pointed out in the introduction, the bibliographic reference field (RN) in an entry is not a single-valued fact about the key of an entry (AC) since there may be many references per entry. References must therefore be split off as a separate table. Furthermore, since a reference could be related to several different entries, it is not enough to include AC as a *foreign key* in the publication table referencing some tuple in the entry table; a separate table denoting a many-to-many relationship between publications and entries must be created, and the order in which the reference appears in the entry must be maintained in an attribute. The same reasoning can be applied to authors of a reference, keywords, features, and so on.

Now suppose that an update to a SWISS-PROT entry occurs. The warehouse maintainer can detect that an update has occurred since SWISS-PROT publishes a list of entries that have been modified since the last release. However, they do not say exactly how the entry has changed. The actual change may be very small; for example, adding an extra author to a reference consumes only a few characters of the new entry. If the entry is relevant to the warehouse (i.e., it is selected by the integration query), the addition of an author will only affect a few tables in the warehouse rather than all 15 tables that represent SWISS-PROT.

Given an old and new entry, it is possible to use various DIFF algorithms to calculate minimal changes. For example, the "acediff" utility will do this for ACe databases. IBM's XMLTreeDiff[46] is a similar utility for data exported in XML (Extensible Markup Language). For data sources that export data in XML (and we believe that this will soon happen for many of the major data sources), algorithms for *ordered trees*[47–51] can be used. However, it is not clear that the order of fields is important in the XML representation of a SWISS-PROT or GenBank entry, nor is it easy to represent updates using positional information. In Reference 52 we therefore advocate the use of a model in which value-based keys are used at every level of nesting, and in this case the DIFF algorithm becomes a simple, efficient top-down algorithm.

Automating the refresh process. To automate the refresh process, the portions of the warehouse defined by integrating queries must be updated (commonly called *view maintenance*) and any derived data or annotations based on the integrating query data recomputed.

The problem of view maintenance has received much attention from the database community, and is illustrated in Figure 4. In this figure, f represents an integration query that takes as input underlying data source instances $I_1, I_2, \ldots I_n$ producing the warehouse $f(I_1, I_2, \ldots I_n)$. The underlying data source instances are then updated, producing new underlying data source instances $I_1 \cup \Delta_1, I_2 \cup \Delta_2, \ldots I_n \cup \Delta_n$. Note that Δ_i may be a combination of insertions and deletions and that $\cup$ is used to denote the incorporation of the insertions and deletions into the data source instance I_i. Furthermore, it is common to represent the modification of a value by the deletion of the old value followed by the insertion of a new value. Thus the expression $I_i \cup \Delta_i$ represents all insertions, deletions as well as modifications that have been made to the i'th data source.

To produce the updated warehouse $f(I_1 \cup \Delta_1, I_2 \cup \Delta_2, \ldots I_n \cup \Delta_n)$, it is always possible to re-ex-

ecute the integration query. However, this is very expensive so the problem is to find a query g that takes as input the updates $\Delta_1, \Delta_2, \ldots, \Delta_n$, and possibly the original instances $I_1, I_2, \ldots I_n$ or the existing warehouse $f(I_1, I_2, \ldots I_n)$, and updates the warehouse to produce the new state. When g can be written without requiring the original instances and only takes as input $\Delta_1, \Delta_2, \ldots, \Delta_n, f(I_1, I_2, \ldots I_n)$, the view is said to be *self-maintainable*.

For example, suppose that we have input (relational) data sources $R(A, B) = \{(a1, b1), (a2, b2)\}$ and $S(B, C) = \{(b1, c1), (b3, c3)\}$, and a view V defined as $f(R, S) = R \bowtie S = \{(a1, b1, c1)\}$. Updates $\Delta_1 = \{(a3, b3)\}$ and $\Delta_2 = \{(b2, c2)\}$ occur to the base relations. Then V can be updated by calculating $V \cup (\Delta_1 \bowtie S) \cup (R \bowtie \Delta_2) \cup (\Delta_1 \bowtie \Delta_2)$, which (assuming that V is large and Δ_1, Δ_2 are small) is more efficient than recalculating the entire view. The view is not self-maintainable, however, since we need to access both R and S to calculate the changes to V. On the other hand, the following view V' is self-maintainable: $f'(R) = \sigma_{A=a1}(R) = \{(a1, b1)\}$. When an update occurs (such as $\Delta'_1 = \{(a1, b3), (a3, b3)\}$), the updated view can be calculated by simply inserting the filtered update ($\{a1, b3)\}$) to the view. More complex view definitions can also be made self-maintainable by reasoning about functional dependencies and foreign key constraints, and storing auxiliary information at the warehouse (see References 53–55 for details).

View maintenance has been extensively studied in the context of relational databases[56–65] (see Reference 66 for a survey), and less extensively studied in the context of object-oriented[67,68] databases, nested[69] relational databases, models allowing multisets,[70] and semistructured[71–73,52] databases. However, the problem of recomputing corrections and annotations has not been studied.

Data provenance. Data provenance[74–77] addresses the problem of tracking the origins of a data item. The data may be the result of a global query, where components of data originate from different underlying data sources. Alternatively, the data may be derived from the warehouse data using various data mining algorithms. For example, suppose that one form of annotation in our warehouse is to assign function to sequences based on similarity (e.g., using BLAST searches). This annotation could then be transitively inherited by other sequences. If the original annotation is determined to be incorrect through experimentation, all subsequent annotations would also have to be undone. It is therefore important to track the origins of the annotation by keeping detailed information about the origin of the annotation. This is discussed in the next section.

Note that data provenance is related to the problem of recomputing annotations. In our example, if some sequence annotation is changed, then any subsequent annotations based on it will need to be redone. Knowing the provenance of data could help determine which annotations need to be recomputed.

Data warehousing in GUS

To take advantage of the benefits of data cleansing and annotation that are available with data warehousing, we have developed a schema called the Genomics Unified Schema (GUS) to integrate and add value to data obtained from several major sequence databases. The databases that are included in GUS thus far are GenBank/EMBL/DDBJ, dbEST, and SWISS-PROT, and contain annotated nucleotide (DNA, RNA [ribonucleic acid]) and amino acid (protein) sequences. GUS uses a relational data model where tables hold the nucleotide and amino acid sequences along with associated annotation.

GUS uses the central dogma of biology (DNA → RNA → protein) as its organizational principle. Sequence-centric entries from the external databases are mirrored within GUS, and also transformed into gene-centric entities. Thus, GUS tables hold the conceptual entities that the sequences and their annotation ultimately represent (i.e., genes), the RNA derived from those genes, and the proteins derived from those RNAs. The incoming sequence annotation may be experimentally determined or predicted via a variety of algorithms, although they are all stored in GUS as features localized as spans (intervals) or points on the underlying sequence(s) (see the FT "fields" in Figure 1). During the transformation into a gene-centric database, data cleansing occurs to identify erroneous annotation and misidentified sequences. Ontologies are used to structure the annotations, in particular those referring to organisms (see the OC field in Figure 1). Additional computational annotation is then generated based on the newly integrated sequences (e.g., gene/protein function.) We are also just beginning the process of manual annotation and curation, which will become increasingly important as time goes by.

Data provenance. The ability to track where data came from is extremely important in GUS. In addi-

tion to the common data warehouse concerns of tracking the origins of data from external sources, we are also concerned with tracking the history of computationally and manually derived data generated in the course of warehouse construction. Genome sequencing efforts, such as the Human Genome Project, have led to the availability of large amounts of nucleotide sequence for which there is little or no annotation that is experimentally verified. Instead, predictions of gene content, gene identity, and gene function are made based on a variety of computational approaches; many of these algorithms train on data sets which themselves contain predictions. Thus one prediction is often dependent upon earlier predictions, and errors can easily be spread and compounded. A similar situation exists for EST (expressed sequence tag) sequencing projects, which identify genes expressed in various types of cells and organisms. The genes represented by the ESTs are identified through computational analysis of individual or (as in the case of GUS) assembled of ESTs. These predictions may be confirmed, altered, or discarded as new information becomes available in the form of experimental results, literature searches, or new sequence data. Thus when we annotate genomic sequences (for gene content) and genes (for gene identity and function) we must record enough information to allow both users and our professional annotators to evaluate whether the annotation is justified given the available evidence.

The information that GUS tracks for computationally derived annotation is: (1) the algorithm used; (2) the algorithm implementation (software version); (3) the algorithm invocation (run-time information); and (4) the algorithm parameters (values passed to the program at run time). A table for each of these *algorithm*-associated types of information is present in GUS. The algorithm tables are not only used for tracking annotation history, but also for tracking data integration. That is, the algorithm tables together with tables describing the *external* data sources are used to record the loading of data from external sources, allowing us to precisely track the source of each tuple, including which script was used to load it. In addition to the algorithm tables, an `Evidence` table is used to relate specific annotations to the facts they are based on. *Fact* tables hold the data generated when algorithms are run against GUS entries.

One example of a fact table is `Similarity`, which stores the results of similarity searches between any two sets of sequences in the database. These similarities could then become the evidence that allows us to predict the cellular role of a particular protein. Algorithm information associated with each tuple in the `Similarity` fact table includes the database index used in the search, thereby providing the ability to identify which version of the sequence database was searched.

It should be noted that the algorithm and evidence tables are also used to track manual annotation. In this case, the `Evidence` table will point to the tuple in GUS on which the annotator based a decision to make an assignment (e.g., confidence in a prediction, or a controlled vocabulary term) or to change a relationship (e.g., merge genes, or split an EST assembly). The algorithm tables capture the annotation software used and (more importantly) who performed the annotation and when.

In GUS, the complete history of any annotation can be retrieved.

Finally, any updates to the database as a result of the annotation process are tracked in *version* tables that record not only the altered tuple, but the algorithm that caused that tuple to be altered, the database transaction, and time at which it was versioned. Thus, the history of any annotation—or more generally, of any tuple—in GUS can be retrieved. This complete annotation history is useful for reconstructing past database states, both for archival purposes and also to aid in identifying and rectifying potential problems with the automated annotation process. It also allows the system to respond more gracefully to user requests for entries that are no longer in the database; instead of simply saying "entry not found," the system can tell the user exactly when and why it was retired from active service.

Not surprisingly, the GUS schema[78] is quite large (over 180 tables, most of which are versioned). As mentioned in the section on issues in warehousing, the compact representation of a SWISS-PROT entry (shown in Figure 1) when translated to a normalized relational model results in each entry being split over about 15 tables; the same holds true for

EMBL-format GenBank and dbEST entries. As a result, just mirroring the external databases in GUS takes about 50 tables. Various tables related to integration and annotation make up the remainder.

Since the schema is large and fairly unintuitive, a Perl-based object layer has been added on top of the relational implementation. Note that this is similar to the strategy used in OPM,[13] in which users can view and query the database as if it were an object-oriented database while the actual implementation is relational.[79] For example, using the object layer a user can view an entry structured as it is in SWISS-PROT rather than as it is stored in relational form; thus the features of an entry can be listed with a simple Perl method invocation, which the object layer translates into a join over the appropriate tables. The object layer is also crucial in tracking data provenance, as it transparently handles a number of versioning and bookkeeping tasks without the need for direct user intervention.

Update management. GUS (the schema) has been instantiated to create a comprehensive public resource of human and mouse genome annotation.[80] Since new sequences that are relevant to this database appear daily in the external data sources, our goal is to complete a cycle of annotation with the most current information available every two to three months. Although ideally GUS should be completely up-to-date with respect to the external data sources, it currently seems acceptable to provide a resource that lags by a few months. For example, SWISS-PROT is a highly curated version of the protein portion of GenBank, and is not completely up-to-date;[81] however, it is extensively used due to the high quality of data and annotations.

To update the database, the latest versions of external databases are downloaded along with any subsequent updates and new entries (including both new entries and modifications of existing entries). Entries from the external databases are then classified as unmodified, new, or modified based on date and version information associated with entries. Unmodified entries can be ignored, and new entries simply inserted into GUS. Modified entries, which can be detected by examining the appropriate entry field (e.g., DT line in SWISS-PROT with "Last annotation update" or an increment in the version of a GenBank accession), are more problematic. Since the complete entry is given rather than the minimal updates, the actual differences must be identified during the entry loading process. Note that since we are mirroring relevant portions of the source databases (essentially "selecting" entries of interest), calculating how GUS should be updated merely entails filtering the updates according to our selection criteria rather than a more complex function (recall the discussion of automating the refresh process in the section on issues in warehousing).

To detect exactly what components of a modified entry have been changed, the object layer is used to implement a simple diff algorithm (recall the discussion of change detection in the section on issues in warehousing). Using the accession number of the entry (which is assumed to be a key for the entry), the object layer retrieves the top level tuple for the entry. It then navigates down the nested structure of the entry by traversing relationships. Note that each table representing a component of the entry uses an internal identifier, which is used to index into the next level of relationship tables. Only when a change in value is detected is a change actually made to the database. As noted earlier, version tables are used to track all changes.

The new and modified entries are then put through an annotation pipeline, which integrates protein and DNA sequences and their annotations, and transforms the sequence-based entries into gene and protein-based entries. By annotation pipeline we mean a series of computational analyses where the output of an analysis is used as input for the next analysis in the series (see for example Reference 82). Further annotation on the integrated sequences is then performed, including functional role prediction. Updates to the annotation produced by this pipeline need to be managed in the same way as the updates from external databases.

Updates to entries from external databases may of course impact the old integrated sequences and the validity of their annotation. Thus, in each annotation cycle the latest entries are used and the annotation (from computational analysis) is freshly generated, but the old integrated sequences and their annotation must also be kept. It is important to maintain stable identifiers for genes, RNAs, and proteins so that data analyses can be compared from different update cycles of GUS. Therefore, we compare the integrated sequences generated between update cycles to pass on identifiers whenever possible at the high level of genes, RNAs, and proteins. Manual annotation tied to these identifiers can then be reapplied.

To ensure that a consistent version of the database is always seen by the public, our current working model is to lock down a "production" version of the database for public consumption while doing the updates in a "development" database. When the entries in the development database are complete, the new version is released to the public. This paradigm will need to change in the near future as we begin manual annotation of GUS entries, since entry selection for manual annotation is guided by user interest unlike the computational analyses that are driven by new or updated entries. As a result, "collisions" may occur between computational and manual annotations because they are on different schedules. Changes in the schemas of external data sources sometimes also occur. Relational views are used to represent the annotation associated with the sequence entries providing us with the ability to easily incorporate certain kinds of schema changes, e.g., the addition or renaming of an attribute.

Early days' experience. After several initial revisions the GUS schema has stabilized, and current versions of SWISS-PROT (protein) and dbEST (EST) have been loaded; loading the newest version of GenBank is in progress. We are currently working on the integration protocols to fully integrate the DNA (GenBank) and protein (SWISS-PROT) entries. These data (GenBank and EST) have been used to create a gene index that attempts to assign mRNA sequences (both literal and those computed via assembly of ESTs) to single genes along with their genomic sequences and gene predictions. The database can be queried and viewed at http://www.allgenes.org.

A number of significant gains have been realized from building the GUS warehouse. First, since we structure the data using ontologies of biological terms on entry into the database, we can query the data in much more powerful ways than are possible in the individual source databases. Second, given this structure, new sequences representing mRNA molecules are much more readily (and reliably) identified for inclusion in our gene index. A third gain is the ease with which we can predict cellular roles and in particular track the evidence for those roles given the presence of proteins (SWISS-PROT and GenBank nonredundant protein databases) in GUS.

We have also found the GUS warehouse to be an effective vehicle for delivering specialized databases. Prior to developing GUS, we were maintaining several different databases, using a variety of database management systems and data models, each of which was designed for a specific function now covered in GUS. These functions[83] include genome annotation (GAIA), gene integration (EpoDB), and generation of a gene index (DoTS); in addition, we were becoming involved in maintaining organism-specific information (such as the mouse and human gene index), a *Plasmodium falciparum* database,[84] and a database of genes expressed in the developing endocrine pancreas.[85] Each of these databases is now (or will become) a function-specific or organism-specific view of GUS.

The issue of update and annotation management has not been completely solved, and further research will continue on better ways to integrate the annotation between cycles using workflow approaches. This becomes especially important in maintaining personnel-intensive manual annotation and will be aided by our ability to track the history of the annotations as described previously.

Performance

As we stated previously, performance is often an overriding concern in choosing a warehouse-based solution over a view-based one. Exactly how much faster a warehouse query will run compared to an equivalent query in a view integration system depends on many factors, including the nature of the query and the number and types of source databases that must be accessed. Table 1 presents some relevant performance data gathered in mid-1995 using an earlier version of the K2/Kleisli system discussed in the section on K2/Kleisli. The table illustrates the performance gains that can be realized by using a relational warehouse (first row). It also demonstrates that the performance of a view integration system can vary widely depending on the join evaluation strategies available to the system (e.g., semijoin versus nested loop join, rows 2–3). The join evaluation strategies available depend in turn on the capabilities of the underlying data sources; the last query, using GDB and Entrez (row 4), could not make use of a semijoin, because the Entrez data driver/system is nonrelational and does not support the operation. Note also that several of the queries failed to complete in this case due to network timeouts (to which the Entrez driver is more susceptible).

The table entries are running times in seconds for several implementations of the following query: "Retrieve the official HUGO (human genome organization) names, accession numbers, and amino acid sequences of all known human genes mapped to

Table 1 Running times of a query on several different implementations of a database

Chromosome:	1	2	3	4	5	6	7	8	9
GSDB LJ[a] (isql)	20	20	18	19	19	23	21	20	17
GDB-GSDB SJ[b] (Kleisli)	147	106	215	150	97	93	138	75	75
GDB-GSDB NL[c] (Kleisli)	1771	1508	2135	1769	1298	3033	1531	1124	1131
GDB-Entrez NL[c] (Kleisli)	—[d]	—[d]	—[d]	1113	420	1943	342	558	848

a LJ = Local Join ("warehouse" approach)
b SJ = Semijoin (relational sources only)
c NL = Nested Loop iteration
d Query failed to complete

chromosome *c*." The query requires both mapping data (from GDB) and sequence data (from GSDB or Entrez/GenBank). At the time these experiments were conducted, GSDB contained identical copies of the relevant tables from GDB, allowing us to treat GSDB as a warehouse with respect to this type of query. This is shown in the first row of the table, where the Sybase command-line interface "isql" was used to run the requisite SQL queries against GSDB. The remaining three rows represent different view evaluation strategies using exactly two source databases and Kleisli as the view system. Note that only the data for the first nine chromosomes are shown and that all times except those in the last row are averages of at least five repetitions.

By way of comparison, using a more recent data set, the GUS system supports a similar query, namely "return all EST assemblies that can be localized to chromosome *c* by radiation hybrid mapping." In our current development system—an Oracle** 8i database running on a 4-processor Linux** machine—this query takes on average 24 seconds. Note that this query translates to a six-table join in which two of the tables each contain more than 3 million rows, whereas the original GDB-GSDB query involved substantially fewer data. For this and other frequently used queries, we have created materialized views in GUS to improve their performance. Using the appropriate materialized view for this query reduces it to a three-table join and allows it to run in less than a second, and we expect further performance improvements when the system is properly tuned for Oracle.[86]

Conclusions

Both the K2/Kleisli implementation of the view integration approach and GUS warehouse have proven useful for genomic applications within the Center for Bioinformatics at the University of Pennsylvania. Kleisli was used for some time to implement several Web-based, parameterized queries that were written for specific user groups. Users could query views that integrated many important on-line data sources (such as GenBank, GSDB, dbEST, GDB, SRS-indexed databases, KEGG and EcoCyc) and application programs (such as BLAST). After the user supplied values for parameters, the data sources and application programs were then accessed. The set of available Web-based queries grew as users mailed in requests, although few people actually wrote CPL queries themselves. K2 has now supplanted Kleisli on our Web site, and the list of queries has been somewhat modified. K2/Kleisli has also been extremely successful within SmithKline Beecham, and has formed the basis for the TAMBIS[18] system at the University of Manchester. K2/Kleisli and other view integration implementations seem most useful in a curiosity-driven/browsing type of environment where network delays and temporary data source unavailability can be tolerated.

GUS, on the other hand, has grown in importance as we have moved toward projects involving "production strength" support for function- and organism-specific applications, in particular those involving annotation. In a production strength system, we need much more control over the data to guarantee the data's correctness. Furthermore, the added annotation is tied to the state of the input data sources, and we have found it easier to mirror that state within the warehouse than to reconstruct it based on timestamps and version information.

It should be noted that K2 was not used to populate GUS. There are three main reasons for this. First, it was felt that although OQL is an excellent query language, it is not well-suited to the task of large-scale data restructuring. For this type of work it is better to rely either on a high-level declarative transformation language like TSL/WOL[87] (and, as an aside, on its ability to execute the resulting transformations effi-

ciently) or on a general-purpose programming/scripting language. Second, unlike SQL, OQL does not have an explicit insert/update syntax; in OQL updates must be performed by method calls that affect the state of the database as "side effects." Third, the data are not highly restructured or integrated when initially mirrored in GUS; most of the integration is performed in the subsequent annotation pipeline. Overall, since the manner in which GUS was to be populated was well understood and relatively straightforward, it was most efficient to write Perl scripts to perform the task. K2 will, however, be used as we incorporate databases that we cannot or do not wish to warehouse locally, for example PubMed (as illustrated in the section on K2/Kleisli) as well as a specialized database of mouse neuroanatomy that we plan to integrate in the near future.

Implicit in the problems encountered in creating and maintaining view integration systems is the lack of standards for, and cooperation between the underlying data sources. These kinds of problems largely motivated a standardization drive based on the Common Object Request Broker Architecture** (CORBA**) proposed by the Object Management Group[88] (OMG). The OMG is a consortium whose mission is to foster interoperability and portability of enterprise applications via cooperative creation and promulgation of object-oriented standards. Foremost of all OMG standards is CORBA, which describes how a network of component systems should behave in order to interoperate in a distributed environment. The Life Sciences Research Task Force (LSR) has been formed within the OMG to address requirements and specifications of software components for life sciences using CORBA. Currently, the LSR has reached consensus on specifications for biomolecular sequence analyses and genome maps, and it is now up to individual data source owners or third parties to modify their data sources or to provide wrappers to their data sources so that they conform to these specifications. We believe, however, that the standardization of all genomic data sources is an unrealistic goal given their diversity, autonomy, and rapid schema changes. This is evidenced by the fact that interest in CORBA seems to have waned over the past year and to have been superseded by XML.[89]

As a universal data exchange format, XML may well supplant existing formats such as EMBL and ASN.1 for biological data,[90] and as such will simplify some of the lower-level driver technology that is part of K2/Kleisli and other view integration systems. There is an abundance of freely available parsers and other software for XML, and heavy industry backing of XML. The question is whether it will do more than function as an exchange format. It may, in fact, become a basis for view integration systems by using one of the query languages[91,92] developed for semistructured data or XML. Before it becomes a good basis for an integration system, however, we believe that several things must happen.

1. Some kind of schema technology must be developed for XML. DTDs function as only a rough schema for XML. For example, there are no base types other than PCDATA (so the integer 198 cannot be distinguished from the string "198"), no ability to specify keys or other constraints on the schema, and the reliance on order makes representing tuples (in which the order of attributes is unimportant) tricky. The recent XMLSchema[93] proposal addresses many of these problems by providing a means for defining the structure, content, and semantics of XML documents.

2. An agreement must be reached on the use of terms, or there must be a mechanism to map between terms. The discussions in this paper have sidestepped one of the most difficult parts of data and software integration: semantic integration. Semantic integration focuses on the development of shared ontologies between domains of users, and on the resolution of naming conflicts (synonyms and homonyms). In the TAMBIS project, although Kleisli was used for the low-level (syntactic) integration, a major effort of the project was to develop an ontology through which researchers could navigate to find information of interest. The view layer K2MDL in K2 aids in semantic integration by providing a means for mapping between concepts in different databases, and has proven extremely useful in the integration projects for SmithKline Beecham. For XML to be useful in data integration, either the usage of tag labels must be uniform across the community, or a semantic layer must be available.

3. A standard for XML storage must be adopted. Several storage techniques are currently being explored based on relational and object-oriented technologies; new technologies are also being considered. However, there is no agreement on what is best. Warehouse developers must currently therefore provide their own mapping layer to store the result of an integration query.

These issues are of current interest in the database research community, and we expect to see some preliminary solutions in the near future.

Because it is likely that warehouses like GUS will continue to be developed, we believe that a number of practical steps should be taken by both the producers of primary data and the developers of warehouses.

1. *Develop "keyed" XML data interchange formats.* There are currently two ways of identifying a node in an XML tree. The first is to use ID attributes, which are globally unique PCDATA strings. The second is to use the ordering of subnodes to identify a position. For example, in a DOM[94] tree representation of an XML document the address of a node is given by the element position of each node on the path from the root to the given node. However, element positions change as new data are added; for example, inserting a new element after the third element changes the position of every element following it in the list. Since positions may change as updates occur, it is therefore desirable to use value-based rather than position-based identifiers. For example, in the relational model, tuples are identified by *keys*, that is, sets of attributes whose values uniquely identify a tuple and which cannot be modified. In a hierarchically structured model (as with K2/Kleisli or XML), the analog is to require that keys be specified at each level of nesting so that each node in the tree can be described by a unique path of keys from the root to that node. Note that this is different from IDs, which must be globally unique within a document rather than locally unique. The XMLSchema[93] is addressing this, and several other proposals[95] are also being made that may have some impact.

2. *Publish minimal changes.* Intuitively, rather than just publishing "Entry 90158697 has been modified" it would also be useful to know how it has been modified, by using statements such as "Feature X has been added to entry 90158697," where X is the value of the feature added. Given keyed hierarchical data, the changes to an entry can simply be represented as the set of paths that represent insertions, the set of paths that represent deletions, and the set of paths that represent modifications of values in the old entry. Since not everyone will want such detailed information, users should probably continue to have the choice of obtaining the newest version of a database or of obtaining the entries that have been modified.

3. *Keep track of where the data came from.* Since data in secondary databases are derived from primary sources, it is important to keep track of *where* the data came from and *why* the data are there. At a minimum, this implies that detailed information should be maintained about the version of the primary data source from which the data were extracted, the date on which the information was extracted, and information about the query that extracted the data. For data that were obtained through analysis packages based on data from primary sources, even more information should be maintained: the analysis performed, the input parameters, the name of the person performing the analysis, etc.

An additional problem that we have not discussed but which has implications for creating warehouses is the ownership of data. Over the past ten years, data originally in the public domain have, for a variety of reasons, had increasingly restrictive licenses placed on their use. While databases such as GenBank and PubMed are still in the public domain, other databases such as SWISS-PROT, which is itself a value-added secondary database, are placing restrictions on the use of their data. In the case of SWISS-PROT, the restrictions are intended as a funding mechanism aimed at commercial uses rather than at nonprofit uses.[96] However, if a nonprofit user integrates part of SWISS-PROT into a specialized database that can then be used by commercial users, the nonprofit user must provide a list of commercial users so that the licensing can be checked.[97] Even though such restrictions are quite reasonable given the high cost of producing high-quality data, they may present a significant barrier to instantiating those portions of a warehouse integration that draw data from primary sources with restrictive licenses.

Acknowledgments

K2 is the work of Jonathan Crabtree, Scott Harker, and Val Tannen. GUS has been designed and developed by Brian Brunk, Jonathan Crabtree, Chris Overton, Jonathan Schug, Chris Stoeckert, and the entire Computational Biology and Informatics Laboratory (CBIL). We are thankful to Peter Buneman and members of the Database and Bioinformatics group in the Department of Computer and Information Science, as well as the members of CBIL at

the Center for Bioinformatics for their input on various aspects of this work.

This research was supported in part by DOE DE-FG02-94-ER-61923 Sub 1, DOE DE-FG02-00-ER-62893, NIH R01-HG01539, NSF DBI99-75206, NSF IIS90-17444, ARO DAAG55-98-1-0031, and a grant from SmithKline Beecham.

**Trademark or registered trademark of Sun Microsystems, Inc., Sybase, Inc., Oracle Corporation, Linus Torvalds, or Object Management Group.

Cited references and notes

1. D. Benson, I. Karsch-Mizrachi, D. Lipman, J. Ostell, B. A. Rapp, and D. Wheeler, "GenBank," *Nucleic Acids Research* **28**, No. 1, 15–18 (2000).
2. A. Bairoch and R. Apweiler, "The SWISS-PROT Protein Sequence Database and Its Supplement TrEMBL in 2000," *Nucleic Acids Research* **28**, 45–48 (2000).
3. V. A. McKusick, *Mendelian Inheritance in Man, Catalogs of Human Genes and Genetic Disorders*, 12th edition, Johns Hopkins University Press, Baltimore, MD (1998).
4. W. Baker, A. van den Broek, E. Camon, P. Hingamp, P. Sterk, G. Stoesser, and M. A. Tuli, "The EMBL Nucleotide Sequence Database," *Nucleic Acids Research* **28**, No. 1, 19–23 (2000).
5. Y. Tateno, S. Miyazaki, M. Ota, H. Sugawara, and T. Gojobori, "DNA Data Bank of Japan (DDBJ) in Collaboration with Mass Sequencing Teams," *Nucleic Acids Research* **28**, No. 1, 24–26 (2000).
6. T. Etzold and P. Argos, "SRS: An Indexing and Retrieval Tool for Flat File Data Libraries," *Computer Applications of Biosciences* **9**, 49–57 (1993), http://srs.ebi.ac.uk.
7. J. Thierry-Mieg and R. Durbin, "ACeDB—A C. elegans Database: Syntactic Definitions for the ACeDB Data Base Manager," 1992, http://genome.cornell.edu/acedocs/syntax.html.
8. N. Goodman, S. Rozen, and L. Stein, "Requirements for a Deductive Query Language in the MapBase Genome-Mapping Database," *Proceedings of Workshop on Programming with Logic Databases*, Vancouver, BC, October 1993, pp. 18–32.
9. W. Fujibuchi, S. Goto, H. Migimatsu, I. Uchiyama, A. Ogiwara, Y. Akiyama, and M. Kanehisa, "DBGET/LinkDB: An Integrated Database Retrieval System," *Proceedings of the Pacific Symposium on Biocomputing* (1998), pp. 683–694, http://www.genome.ad.jp/dbget/.
10. M. Rebhan, V. Chalifa-Caspi, J. Prilusky, and D. Lancet, *GeneCards: Encyclopedia for Genes, Proteins and Diseases*, Technical Report, Weizmann Institute of Science, Bioinformatics Unit and Genome Center, Rehovot, Israel (1997), http://nciarray.nci.nih.gov/cards/.
11. L. Wong, "Kleisli, a Functional Query System," *Journal of Functional Programming* **10**, No. 1, 19–56 (2000).
12. P. Buneman, S. B. Davidson, K. Hart, C. Overton, and L. Wong, "A Data Transformation System for Biological Data Sources," *Proceedings of the International Conference on Very Large Data Bases* (*VLDB*), September 1995, Morgan Kaufmann Publishers, San Francisco, CA (1995), pp. 158–169.
13. I.-M. A. Chen and V. M. Markowitz, "An Overview of the Object-Protocol Model (OPM) and OPM Data Management Tools," *Information Systems* **20**, No. 5, 393–418 (1995).
14. J. Melton and A. Simon, *Understanding the New SQL: A Complete Guide*, Morgan Kaufmann Publishers, San Francisco, CA (1993).
15. *The Object Database Standard: ODMG 2.0*, R. G. G. Cattell, Editor, Morgan Kaufmann Publishers, San Mateo, CA (1997).
16. P. Buneman, L. Libkin, D. Suciu, V. Tannen, and L. Wong, "Comprehension Syntax," *SIGMOD Record* **23**, No. 1, 87–96 (March 1994).
17. S. B. Davidson, C. Overton, and P. Buneman, "Challenges in Integrating Biological Data Sources," *Journal of Computational Biology* **2**, No. 4, 557–572 (1995).
18. N. W. Paton, R. Stevens, P. G. Baker, C. A. Goble, S. Bechhofer, and A. Brass, "Query Processing in the TAMBIS Bioinformatics Source Integration System," *Proceedings of the 11th International Conference on Scientific and Statistical Databases*, IEEE Press, New York (1999), pp. 138–147.
19. P. Buneman, J. Crabtree, S. B. Davidson, C. Overton, V. Tannen, and L. Wong, "BioKleisli," *Bioinformatics*, S. Letovsky, Editor, Kluwer Academic Publishers, New York (1998).
20. S. Davidson, C. Overton, V. Tannen, and L. Wong, "Biokleisli: A Digital Library for Biomedical Researchers," *Journal of Digital Libraries* **1**, No. 1, 36–53 (1996).
21. K. Hart, L. Wong, C. Overton, and P. Buneman, "Using a Query Language to Integrate Biological Data," *Abstracts of 1st Meeting on the Interconnection of Molecular Biology Databases*, Stanford, CA, August 1994.
22. R. G. G. Cattell, D. K. Barry, D. Bartels, M. Berler, J. Eastman, S. Gamerman, D. Jordan, A. Springer, H. Strickland, and D. Wade, *The Object Database Standard: ODMG 2.0*, Morgan Kaufmann Publishers, San Francisco, CA (1997).
23. R. Milner, M. Tofte, and R. Harper, *The Definition of Standard ML*, MIT Press, Boston, MA (1990).
24. See http://www.javasoft.com/j2se/.
25. See http://www.javasoft.com/products/.
26. See http://java.sun.com/products/jdbc/.
27. S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic Local Alignment Search Tool," *Journal of Molecular Biology* **215,** 403–410 (1990).
28. See http://www.ncbi.nlm.nih.gov/entrez/query/static/overview.html.
29. ISO, *Standard 8824. Information Processing Systems. Open Systems Interconnection. Specification of Abstraction Syntax Notation One* (*ASN.1*), 1987.
30. M. Kanehisa and S. Goto, "KEGG: Kyoto Encyclopedia of Genes and Genomes," *Nucleic Acids Research* **28**, No. 1, 29–34 (2000).
31. P. D. Karp, M. Riley, M. Saier, I. T. Paulsen, S. M. Paley, and A. Pellegrini-Toole, "The EcoCyc and MetaCyc Databases," *Nucleic Acids Research* **28**, No. 1, 56–59 (2000).
32. C. Harger, G. Chen, A. Farmer, W. Huang, J. Inman, D. Kiphart, F. Schilkey, M. P. Skupski, and J. Weller, "The Genome Sequence DataBase," *Nucleic Acids Research* **28**, No. 1, 31–32 (2000).
33. M. S. Boguski, T. M. Lowe, and C. M. Tolstoshev, "dbEST—Database for 'Expressed Sequence Tags,' " *Nature Genetics* **4**, No. 4, 332–333 (August 1993).
34. J. A. Blake, J. T. Eppig, J. E. Richardson, M. T. Davisson, and the Mouse Genome Database Group, "The Mouse Genome Database (MGD): Expanding Genetic and Genomic Resources for the Laboratory Mouse," *Nucleic Acids Research* **28**, No. 1, 108–111 (2000).
35. P. Pearson, N. Matheson, N. Flescher, and R. J. Robbins, "The GDB Human Genome Data Base Anno 1992," *Nucleic Acids Research* **20**, 2201–2206 (1992).
36. C. Souza dos Santos, S. Abiteboul, and C. Delobel, "Virtual Schemas and Bases," *Proceedings of the 4th International Conference on Extending Database Technology*, Cambridge, UK,

March 1994, *Lecture Notes in Computer Science* **779**, Springer-Verlag, Berlin (1994).
37. See http://www.cbil.upenn.edu/K2.
38. M. Rusinkiewicz, A. Sheth, and G. Karabatis, "Specifying Interdatabase Dependencies in a Multidatabase Environment," *IEEE Computer* **24**, No. 12, 46–53 (1991).
39. G. Wiederhold, "Mediators in the Architecture of Future Information Systems," *IEEE Computer* **25**, No. 3, 38–49 (1992).
40. J. Chen, D. DeWitt, F. Tian, and Y. Wang, "Niagaracq: A Scalable Continuous Query System for Internet Databases," *Proceedings of the ACM SIGMOD International Conference*, Dallas, TX, May 2000, ACM, New York (2000), pp. 379–390.
41. L. Liu, C. Pu, R. Barga, and T. Zhou, "Differential Evaluation of Continual Queries," *Proceedings of the International Conference on Distributed Computing Systems* (*ICDCS*), ACM, New York (1996), pp. 458–465.
42. L. Liu, C. Pu, and W. Tang, "Continual Queries for Internet Scale Event-Driven Information Delivery," *IEEE Transactions on Knowledge and Data Engineering* **11**, No. 4, 610–628 (1999).
43. L. Liu, C. Pu, R. Barga, and T. Zhou, "Continuous Queries over Append-Only Databases," *Proceedings of the ACM SIGMOD Conference on the Management of Data*, ACM, New York (1992), pp. 321–330.
44. Push capabilities may be more common in the private sector. For example, DoubleTwist (http://www.doubletwist.com) has software agents that notify users when relevant entries are added to their database(s).
45. J. D. Ullman, *Principles of Database and Knowledgebase Systems I*, Computer Science Press, Rockville, MD 20850 (1989).
46. See http://www.alphaWorks.ibm.com/formula/xmltreediff.
47. S. M. Selkow, "The Tree-to-Tree Editing Problem," *Information Processing Letters* **6**, No. 6, 184–186 (1977).
48. K. C. Tai, "The Tree-to-Tree Correction Problem," *Journal of the Association for Computing Machinery* **26**, No. 3, 422–433 (1979).
49. K. Zhang and D. Shasha, "Simple Fast Algorithms for the Editing Distance Between Trees and Related Problems," *SIAM Journal on Computing* **18**, No. 6, 1245–1262 (1989).
50. K. Zhang, R. Statman, and D. Shasha, "On the Editing Distance Between Unordered Labeled Trees," *Information Processing Letters* **42**, No. 3, 133–139 (1992).
51. S. Chawathe and H. Garcia, "Meaningful Change Detection in Structured Data," *Proceedings of the ACM SIGMOD Conference on Management of Data*, May 1997, ACM, New York (1997), pp. 26–37.
52. H. Liefke and S. Davidson, "View Maintenance for Hierarchical Semistructured Data," *Proceedings of the International Conference on Data Warehousing and Knowledge Discovery* (*DaWaK'00*), London, England, September 2000, Springer-Verlag, Berlin (2000).
53. D. Quass, A. Gupta, I. S. Mumick, and J. Widom, "Making Views Self-Maintainable for Data Warehousing," *Proceedings of the IEEE Conference on Parallel and Distributed Information Systems* (*PDIS*), Miami Beach, FL, December 1996, IEEE, New York (1996), pp. 158–169.
54. K. A. Ross, D. Srivastava, and S. Sudarshan, "Materialized View Maintenance and Integrity Constraint Checking: Trading Space for Time," *Proceedings of ACM SIGMOD International Conference on Management of Data*, Montreal, Canada, June 1996, ACM, New York (1996), pp. 447–458.
55. N. Huyn, "Multiple-View Self-Maintenance in Data Warehousing Environments," *Proceedings of the International Conference on Very Large Databases* (*VLDB*), Athens, Greece, 1997, Morgan Kaufmann Publishers, San Francisco, CA (1997), pp. 26–35.
56. O. Shmueli and A. Itai, "Maintenance of Views," *Proceedings of ACM SIGMOD International Conference on Management of Data*, Boston, MA, June 1984, ACM, New York (1984), pp. 240–255.
57. J. A. Blakeley, P.-A. Larson, and F. Tomba, "Efficiently Updating Materialized Views," *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1986, ACM, New York (1986), pp. 61–71.
58. J. A. Blakeley, N. Coburn, and P. A. Larson, "Updating Derived Relations: Detecting Irrelevant and Autonomously Computable Updates," *ACM Transactions on Database Systems* **14**, No. 3, 369–400 (September 1989).
59. A. Gupta, I. S. Mumick, and V. S. Subrahmanian, "Maintaining Views Incrementally," *Proceedings of the ACM SIGMOD Conference*, Washington, DC, May 1993, ACM, New York (1993), pp. 157–166.
60. S. Ceri and J. Widom, "Deriving Production Rules for Incremental View Maintenance," *Proceedings of the International Conference on Very Large Data Bases* (*VLDB*), Barcelona, Spain, Morgan Kaufmann Publishers, San Francisco, CA (1991), pp. 577–589.
61. R. Paige, "Applications of Finite Differencing to Database Integrity Control and Query/Transaction Optimization," *Proceedings of Advances in Database Theory*, New York (1984), pp. 170–209.
62. X. Qian and G. Wiederhold, "Incremental Recomputation of Active Relational Expressions," *IEEE Transactions on Knowledge and Data Engineering* **3**, No. 3, 337–341 (1991).
63. T. Griffin, L. Libkin, and H. Tricket, "An Improved Algorithm for the Incremental Recomputation of Active Relational Expressions," *IEEE Transactions on Knowledge and Data Engineering* **9**, No. 3, 508–511 (1997).
64. D. Quass, "Maintenance Expressions for Views with Aggregation," *Workshop on Materialized Views: Techniques and Applications*, Montreal, Canada, June 1996, ACM, New York (1996), pp. 110–118.
65. A. Gupta, V. Harinarayan, and D. Quass, "Generalized Projections: A Powerful Approach to Aggregation," *Proceedings of International Conference on Very Large Databases* (*VLDB*), Zurich, Switzerland, September 1995, Morgan Kaufmann Publishers, San Francisco, CA (1995), pp. 358–369.
66. A. Gupta and I. S. Mumick, "Maintenance of Materialized Views: Problems, Techniques, and Applications," *IEEE Data Engineering Bulletin* **18**, No. 2, 3–18 (June 1995).
67. D. Gluche, T. Grust, C. Mainberger, and M. H. Scholl, "Incremental Updates for Materialized OQL Views," *Proceedings of the 5th International Conference on Deductive and Object-Oriented Databases*, December 1997, Montreux, Switzerland, *Lecture Notes in Computer Science* **1341**, Springer-Verlag, Berlin (1997).
68. H. A. Kuno and E. A. Rundensteiner, "Incremental Maintenance of Materialized Object-Oriented Views in Multiview: Strategies and Performance Evaluation," *IEEE Transactions on Knowledge and Data Engineering* **10**, No. 5, 768–792 (1998).
69. A. Kawaguchi, D. F. Lieuwen, I. S. Mumick, and K. A. Ross, "Implementing Incremental View Maintenance in Nested Data Models," *Proceedings of International Workshop on Database Programming Languages*, Estes Park, CO, August 1997, *Lecture Notes in Computer Science* **1369**, Springer-Verlag, Berlin (1997), pp. 202–221.
70. T. Griffin and L. Libkin, "Incremental Maintenance of Views with Duplicates," *Proceedings of the ACM SIGMOD Confer-*

ence, San Jose, CA, May 1995, ACM, New York (1995), pp. 328–339.
71. D. Suciu, "Query Decomposition and View Maintenance for Query Languages for Unstructured Data," *Proceedings of International Conference on Very Large Databases* (*VLDB*), Bombay, India, September 1995, Morgan Kaufmann Publishers, San Francisco, CA (1995), pp. 227–238.
72. S. Abiteboul, J. McHugh, M. Rys, V. Vassalos, and J. Wiener, "Incremental Maintenance for Materialized Views over Semistructured Data," *Proceedings of the International Conference on Very Large Databases* (*VLDB*), New York, August 1998, Morgan Kaufmann Publishers, San Francisco, CA (1998), pp. 38–49.
73. Y. Zhuge and H. Garcia-Molina, "Graph Structured Views and their Incremental Maintenance," *Proceedings of the 14th International Conference on Data Engineering* (*ICDE*), Orlando, FL, February 1998, IEEE, New York (1998), pp. 116–125.
74. Y. Cui, J. Widom, and J. Wiener, "Tracing the Lineage of View Data in a Data Warehousing Environment," *ACM Transactions on Database Systems* **25**, No. 2, 179–227 (2000).
75. A. Woodruff and M. Stonebraker, "Supporting Fine-Grained Data Lineage in a Database Visualization Environment," *Proceedings of the Thirteenth International Conference on Data Engineering*, IEEE, New York (1997), pp. 91–102.
76. T. Lee, S. Bressan, and S. Madnick, "Source Attribution for Querying Against Semi-Structured Documents," *Proceedings of the International Conference on Information and Knowledge Management—Workshop on Web Information and Data Management*, ACM, New York (1998).
77. P. A. Bernstein and T. Bergstraesser, "Meta-Data Support for Data Transformations Using Microsoft Repository," *IEEE Data Engineering Bulletin* **22**, No. 1, 9–14 (1999).
78. See http://www.allgenes.org/cgi-bin/schemaBrowser.pl for the complete schema.
79. This approach was used in GDB (see http://gizmo.lbl.gov/opm.html).
80. See http://www.allgenes.org.
81. A new version is released quarterly, but it is difficult to determine how much of a lag there is from when the sequences first appear in GenBank.
82. L. C. Bailey, Jr., S. Fischer, J. Schug, J. Crabtree, M. Gibson, and G. C. Overton, "Gaia: Framework Annotation of Genomic Sequence," *Genome Research* **8**, No. 3, 234–250 (1998).
83. For descriptions of these databases, see http://www.cbil.upenn.edu.
84. See http://www.plasmodb.org.
85. See http://www.cbil.upenn.edu/EPConDB.
86. Our current production GUS database runs under Sybase 11.9.2 and we are in the process of adding support for Oracle.
87. S. B. Davidson and A. Kosky, "WOL: A Language for Database Transformations and Constraints," *Proceedings of the International Conference of Data Engineering*, April 1997, IEEE, New York (1997), pp. 55–65.
88. See http://www.omg.org.
89. See http://www.w3.org.
90. See for example http://www.ebi.ac.uk/microarray/MGED/.
91. A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu, "A Query Language for XML," *Proceedings of the International World Wide Web Conference* (*WWW8*), Toronto, 1999, http://www8.org.
92. J. Robie, J. Lapp, and D. Schach, "XML Query Language (XQL)," *W3C Query Languages Workshop* (*QL'98*), Boston, MA, December 1998, http://www.w3.org/.
93. World Wide Web Consortium (W3C), *XML Schema Part 0: Primer*, 2000, http://www.w3.org/TR/xmlschema-0/.
94. World Wide Web Consortium (W3C), *Document Object Model (DOM) Level 1 Specification*, 1998, http://www.w3.org/TR/WD-DOM-971009.
95. P. Buneman, S. Davidson, W. Fan, C. Hara, and W.-C. Tan, "Keys for XML." To appear in the 10th International World Wide Web Conference, May 2001.
96. See http://www.isb-sib.ch/announce.
97. The SWISS-PROT copyright notice also states for nonprofit users: "There are no restrictions on its use by non-profit institutions as long as its content is in no way modified."

Accepted for publication November 20, 2000.

Susan B. Davidson *Center for Bioinformatics and Department of Computer and Information Science, University of Pennsylvania, 200 South 33rd Street, Philadelphia, Pennsylvania 19104-6389 (electronic mail: susan@cis.upenn.edu).* Dr. Davidson received the B.A. degree in mathematics from Cornell University, Ithaca, NY, in 1978, and the Ph.D. degree in electrical engineering and computer science from Princeton University, Princeton, NJ, in 1982. Dr. Davidson is a professor in the Department of Computer and Information Science, where she has been since 1982. Her research interests include database systems, database modeling, distributed systems, bioinformatics, and real-time systems. Dr. Davidson is also Interim Director of the Center for Bioinformatics, a multischool center spanning the Schools of Medicine, Engineering and Applied Science, and Arts and Sciences. The center pulls together researchers from biomedicine, statistics, mathematics, and computer science, and is known for its pioneering work in database integration, genomic schema development, visualization tools, and annotation systems.

Jonathan Crabtree *Center for Bioinformatics, University of Pennsylvania, 1315 Blockley Hall, Philadelphia, Pennsylvania 19104-6021 (electronic mail: crabtree@pcbi.upenn.edu).* Mr. Crabtree is a senior programmer/analyst at the University of Pennsylvania's Center for Bioinformatics. He received his undergraduate degree from Williams College in 1993, followed by an M.S.E. degree in computer and information science from the University of Pennsylvania in 1996, where he is also currently enrolled as a part-time Ph.D. student. Over the past five years he has worked on a variety of research projects, ranging from genome annotation and database integration to comparative sequence analysis and interactive genomic visualization.

Brian P. Brunk *Center for Bioinformatics, University of Pennsylvania, 1316 Blockley Hall, Philadelphia, Pennsylvania 19104-6021 (electronic mail: brunkb@pcbi.upenn.edu).* Dr. Brunk is a senior IT project manager in the Center for Bioinformatics directing the Genomics Unified Schema (GUS) project, which involves database integration and analysis of disparate biological databases. He received his Ph.D. degree in 1988 from the University of Virginia, Department of Biology, working with Dr. Paul Adler studying the genetic basis of pattern formation in the fruit fly (D. melanogaster). Dr. Brunk then did a postdoctoral fellowship with Dr. Charles Emerson (lastly at the University of Pennsylvania, Department of Cell and Developmental Biology) before joining the Center for Bioinformatics under the direction of Dr. G. Christian Overton. His current research interests include utilizing database integration and data mining to represent and understand biological genetic networks.

Jonathan Schug *Center for Bioinformatics, University of Pennsylvania, 1315 Blockley Hall, Philadelphia, Pennsylvania 19104-6021 (electronic mail: jschug@pcbi.upenn.edu).* Mr. Schug is a staff programmer at the Center for Bioinformatics, and a Ph.D. degree candidate in computer and information science at the University of Pennsylvania. His research interests include the control of gene expression, including DNA sequence analysis, using information-theoretic and linguistic approaches, and modeling of genetic circuits using hybrid systems.

Val Tannen *Department of Computer and Information Science, University of Pennsylvania, 200 South 33rd Street, Philadelphia, Pennsylvania 19104-6389 (electronic mail: val@cis.upenn.edu).* Dr. Tannen received his Ph.D. degree in computer science from MIT in 1987, and has been a University of Pennsylvania faculty member since then. His research interests include information integration, query optimization, programming languages, logic in computer science, and parallel processing.

G. Christian Overton *Center for Bioinformatics and Department of Genetics, University of Pennsylvania, 1312 Blockley Hall, Philadelphia, Pennsylvania 19104-6021.* Dr. Overton was an associate professor in the Department of Genetics of the University of Pennsylvania School of Medicine, and the founding director of the Center for Bioinformatics at the University of Pennsylvania School of Medicine. He received a B.S. degree in physics and mathematics from the University of New Mexico, a Ph.D. degree in biophysics from the Johns Hopkins University, and subsequently an M.S.E. degree in computer science from the University of Pennsylvania. Prior to joining the University of Pennsylvania faculty, he spent five years as part of the artificial intelligence research group at the Unisys Center for Advanced Information Technology. His research interests included the implementation of databases for genome informatics and gene expression, and the development of database technology for the evolution, transformation, and integration of databases. He died unexpectedly on June 1, 2000.

Christian J. Stoeckert, Jr. *University of Pennsylvania, 1313 Blockley Hall, Philadelphia, Pennsylvania 19104-6021 (electronic mail: stoeckrt@pcbi.upenn.edu).* Dr. Stoeckert is a research associate professor in the Department of Genetics of the University of Pennsylvania School of Medicine. He received his Ph.D. degree in 1982 at Johns Hopkins University in biophysics, and did his postdoctoral training with Sherman Weissman in the Department of Human Genetics at the Yale University School of Medicine. He has been an assistant professor in the Department of Anatomy at the University of Pennsylvania and an associate member of the Stoke's Research Institute at the Children's Hospital of Philadelphia. His research interests include databases, gene annotation, gene expression analysis, and ontologies.