Challenges in Integrating Biological Data Sources *

S.B. Davidson, C. Overton and P. Buneman
Dept. of Computer and Information Science & Dept. of Genetics
University of Pennsylvania
Philadelphia, PA 19104
Email: {*susan,coverton,peter*}*@central.cis.upenn.edu*

July 12, 1995

1 Introduction

In the 1985 National Academy of Sciences report, "Models for Biomedical Research: A New Perspective," Morowitz et al [9] argue that biological research had reached a point where "new generalizations and higher order biological laws are being approached but may be obscured by the simple mass of data." The authors go on to propose the creation of a "Matrix of Biological Knowledge" (now called the Biomatrix) in which data, information and knowledge are structured and stored to provide an integrated view of biology. During the intervening ten years, the rate at which data has been generated has, if anything, exceeded the expectations of that report, but fortunately the data is mostly available online, overcoming at least one of the major hurdles in creating the Biomatrix.

However, despite substantial efforts at creating unified databases for genomic data (see for example [25]), we have to ask in what sense the ultimate goal of the creating the Biomatrix as a unified database for biological knowledge is attainable. We see a number of very serious obstacles to this:

- As new experimental techniques are developed and as "new generalizations and ... laws" are discovered, radical changes to the structure (schema) of the database are required. Coping with the evolution of structure is a major challenge to database and knowledge base technology.
- The scope of such a database is unclear. Even if it were possible, for example, to construct a satisfactory database for genomic data, this represents only a fraction of biological data.

*This research was supported in part by DOE DE-FG02-94-ER-61923, NSF BIR94-02292 PRIME, ARO AASERT DAAH04-93-G0129, and ARPA N00014-94-1-1086.

And even if a complete Biomatrix could be developed, how would this relate to other, non-biological, data sources?

- The problem of scale is horrendous. How are small databases associated with individual research groups to be tied in with the structure of some large integrated database? The administrative problems alone appear unsurmountable.

Thus, while the Biomatrix calls for a massive integration of data on a huge scale in both volume and complexity, significant technical challenges still remain for the smaller scale data integration problems that arise frequently within bioinformatics. The problem of integration has several components, not all of which have been met by current database technology. First, there is a collection of inter-related data in heterogeneous data sources that are connected over the internet. Some of these data sources are “standard databases” with well understood query interfaces (e.g. relational databases and to some extent object-oriented databases). Others are in data exchange/interchange formats with more restricted, navigational interfaces (e.g. ACeDB and Entrez). Still others are in flat files with some structure that can be parsed to retrieve data (e.g. ASN.1 exchange format data and GenBank flat files). In addition, certain application programs such as BLAST and FASTA can be viewed as a kind of data source query that ideally should be merged with more traditional queries. Second, an increasing number of users wish to perform complex, “bulk” queries against distributed data. Until recently many users have been happy merely to browse through data sources, finding related data in a rather haphazard manner. While some users will continue to be satisfied with special-purpose analysis packages or GUIs, or will only want browsing capabilities, advanced analysis and data source interrogation capabilities require more powerful query facilities. Furthermore, users' expectations of how fast data should be retrieved have risen with technological advances in networking. Ultimately the query interface should be supported using a high-level, well-defined language that is capable of optimizing ad-hoc queries to provide answers in a timely manner. Third, updates to data are strictly locally controlled. It is doubtful that resource maintainers will relinquish local autonomy and allow enough flexibility in the transactional capabilities of their systems to implement global updates; the current mode of operation are informal update paths throughout a federation of data sources.

Various approaches to integration are appearing throughout the bioinformatics community and it is unlikely that there will be a single satisfactory strategy. In this report, we examine the technical challenges to integration, critique the available tools and resources, and compare the cost and advantages of various methodologies. We begin by analyzing the basic steps in strict and complete integration: 1) transformation of the various schemas to a common data model; 2) matching of semantically related schema objects; 3) schema integration; 4) transformation of data to the federated database on demand; and 5) matching of semantically equivalent data. Some progress has been made on generic problems such as (1) and (3) within the wider database community, but issues of semantics (steps (2) and (5)) have only been dealt with any degree of success by domain experts within the biological community. We then look at the solution space of integration strategies as defined by two axes, the “tightness” of federation and the “degree” of instantiation, discuss where various solutions fall on this plane, and examine their cost and advantages/disadvantages. Finally, we examine technical challenges that are not adequately addressed by these approaches, but are essential elements of a long-term solution: managing optimizations to provide timely response, and dealing with updates at both the instance and schema level.

2 Steps in Integration

The objective of integration is to make data distributed over a number of distinct, heterogeneous databases accessible via a single interface. Having constructed this interface, the system must be capable of efficiently executing queries and of reacting to changes in the underlying resources, whether they be updates to the schema or to the data. The cost of integration therefore has two components: the cost of initially constructing the interface, and the cost of maintaining it. This section focuses on the first component, construction of the single database interface; the other component will be discussed in later sections.

Rather than discussing integration in the abstract, we present a simple example to make the steps concrete.

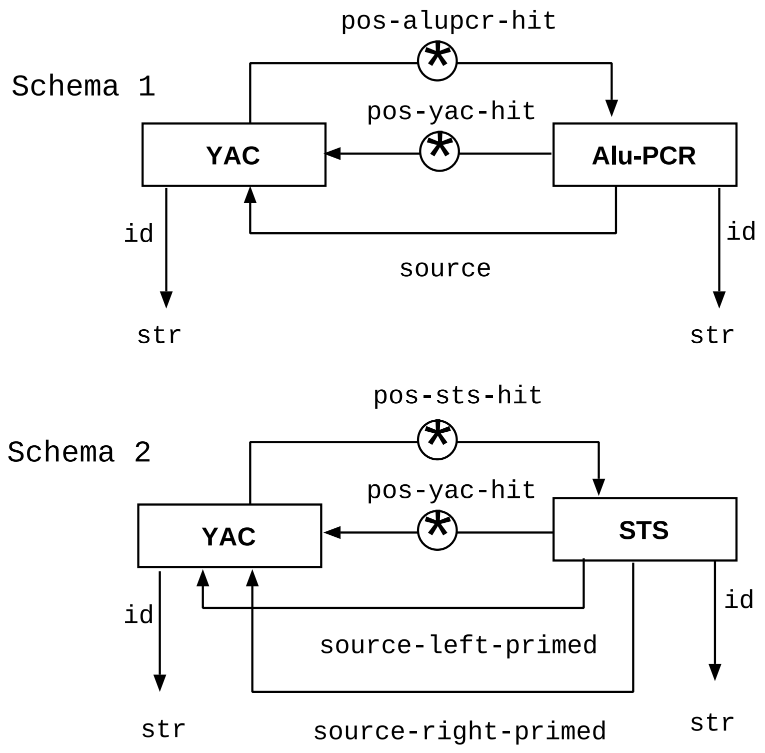

Figure 1: Schemas for YAC/STS and YAC/Alu-PCR databases

Example 1: Figure 1 shows the schemas of two databases representing YAC/STS and YAC/Alu-PCR physical mapping data.[1] The first schema has two classes: *YAC* and *Alu-PCR*.

[1]The graphical notation used here is inspired by [1].

The YAC class has two attributes: *id*, representing the unique ID of a YAC, and *pos-alupcr-hit*, a set-valued attribute (indicated by the "star" node on the edge) denoting YACs positive by hybridization to an AluPCR probe. The *Alu-PCR* class has three attributes representing its unique ID, the set of YACs for which it is positive (*pos-yac-hit*), and the particular YAC which served as the source for generating the AluPCR probe.

The second schema also has two classes, this time *YAC* and *STS*. The *YAC* class has attributes representing its unique ID and positive experimental hits with STSs, an analog of the YAC and Alu-PCR relationship. The *STS* class has four attributes representing its unique ID, the YACs for which it is positive, and the "left" and "right" ends of the YAC insert junction used to obtain sequence for PCR primer selection. ("Left" and "right" are arbitrary designations here.)

Suppose we want to combine these two databases into a single database containing information about both STS and AluPCR results. A suitable schema is shown in figure 2, where the "plus" node indicates a variant. Variant types are common in programming languages as "discriminated unions" where they are used to express the possibility that a data item may take on one of a number of mutually exclusive forms. They occur frequently in these examples, but they are not properly represented in database management systems – even in object oriented systems. In this example the *YAC* classes from both the source databases are mapped to a single class *YAC* in the target database. The *pos-alupcr-hit* and *pos-sts-hit* attributes of the *YAC* classes are mapped to a single attribute *pos-probe-hit* which takes a value that is either an *Alu-PCR* or an *STS*, depending on the type of experiment. A more difficult mapping involves simplifying the representations for Alu-PCR and STS probe source. Rather than having different relationships for the two types of probes, a new relationship, *source*, is introduced that maps the probe to its YAC source. This is sufficient to specify the source of an Alu-PCR probe, but for the STS probes, a new string attribute, *primer-set-end*, is added that indicates which YAC-insert junction was used to develop the STS. Note that this is a more general representation than the original STS source information. To resolve this difference in representation a straightforward embedding of data will not be sufficient; we will need to do some more sophisticated structural transformations on the data.

From this example, the general task of database integration can be seen to consist of five conceptual tasks:

- Data model transformation
- Semantic schema matching
- Schema integration
- Data transformation
- Semantic data matching

We expand on what we mean by each of these.

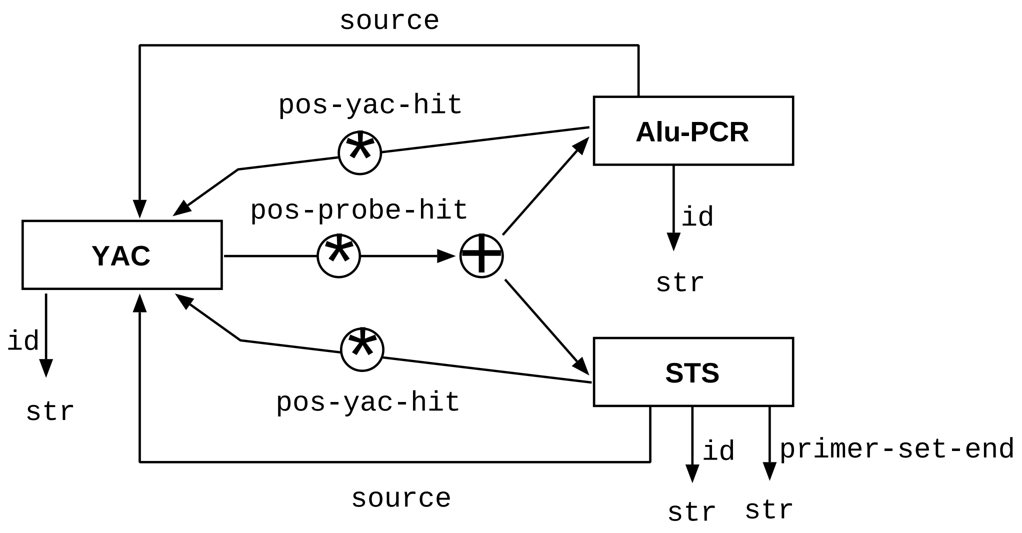

Figure 2: An integrated schema of YAC/STS and YAC/Alu-PCR results.

Data model transformation. Bearing in mind that we are extracting portions of the underlying databases for integration, the first step in integration is to represent each of the underlying schemas in a common model. Obviously, this transformation should preserve the relevant information. Furthermore, it should be sufficiently simple and expressive to allow data to be represented in multiple ways so that conflicts between alternative representations can be resolved. Thus the common data model must be at least as expressive as any of the individual data models. It is, of course, possible to represent almost any data structure within the relational data model with sufficiently elaborate encoding. By "expressive" we mean a data model in which the transformation is natural and obvious. For example, we might want to modify the schema for a YAC/STS experiment result as shown in figure 3 prior to integrating it with the YAC/Alu-PCR database, and the common model must be able to support both representations.

Various models have been used for the common data model, ranging from relational and extended entity-relationship models to semantic, functional, and object-oriented models. In [27] the various requirements on a model for transforming heterogeneous databases are examined, and the authors conclude that a model supporting complex data-structures (sets, records and variants), object-identity and specialization and generalization relations between object classes is desirable.

By and large, the transformation between models is a purely syntactic process, however there are interesting questions about information loss and equivalence of representation.

Semantic schema matching: Before schemas can be merged, the correspondences among concepts with semantic overlap must be established across the participating data sources. This is typically done by resolving names between the schemas (whether homonyms or synonyms), or creating specializations and generalizations of concepts to show similarities among terms.

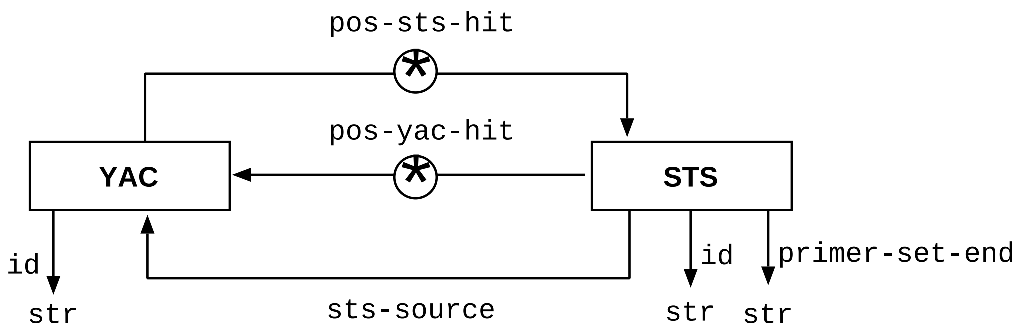

Figure 3: A modified schema for a YAC/STS experiment results

For example, in the YAC-Probe-Source example, we determined that there is a natural correspondence between YAC/STS and YAC/Alu-PCR experiments; however, a different approach to integration than the one we proposed might determine that Alu-PCR and STS experiments are both specializations of some superclass of YAC/probe experiments.

Semantic matching at the schema level remains one of the thorniest problem in data source integration. Unfortunately, there are no proven technologies to automate this process, which currently demands a labor intensive effort on the part of domain experts. Nonetheless, the process can be facilitated by improving the level of annotation of the participating databases and establishing standardized thesauri of biological concepts (see working group reports for MIMBD94). If done correctly, these steps in themselves may create a foundation for automated methods of semantic matching.

Schema integration: Having determined similarities between concepts in the underlying schemas, a common global schema is created from the underlying schemas. This may be as simple as taking the union of the underlying schemas, or may require further modifications of representation in the underlying schemas and in the integrated schema itself.

First, concepts that overlap semantically are frequently represented in different ways in different schemas since the specifics of concept representation typically depend on application requirements. Thus, before correspondences can be established between schemas it may be necessary to resolve conflicts of representation in the underlying schemas.

Returning to the YAC-Probe-Source example, it is necessary to perform a structural modification on the database of YAC/STSs to replace the two STS primer source attributes with a single source attribute, *sts-source*, and the string-valued attribute *primer-set-end* as previously described. This yields an intermediate database with the schema shown in Figure 3.

Second, the way in which the integrated database itself will be used may be radically different from the ways in which the underlying databases were designed and it may therefore be necessary to add, modify or delete concepts. As an example, it may be necessary to add explicit linking tables between concepts from different schemas that are related at some level.

Continuing with the YAC-Probe-Source example, the integration is fairly simple: We merely have to associate the classes and attributes of the two source databases with those in the integrated database, so that the *YAC* classes and *id* attributes, and also the *Alu-PCR* and *STS* attributes, are associated.

Data transformation: In structurally transforming the schemas, it is important to keep in mind that these changes must eventually be reflected in terms of the *data* in the underlying databases. Although it would seem that this should follow directly from the structural schema modifications, there are in general many possible interpretations of a schema modification. For example, in a data model supporting classes of objects and optional attributes of classes, suppose we changed an attribute of an existing class from being optional to being required. There are a number of ways that such a schema modification can be reflected on the underlying data: we could insert a default value for the attribute wherever it is omitted, or we could simply delete any objects from the class for which the attribute is missing. It is therefore important that the meaning of the schema modification be represented explicitly as a data transformation. See [17] for more details on database transformations.

Semantic data matching: Semantic matching at the data level is, if anything, more difficult than semantic schema matching, having the least formal methodological support and requiring the most domain knowledge. In semantic data matching, the relationships between data items that are equivalent across data sources or are attributes of one another are made explicit. Finding which protein sequence entries in PIR or loci in GDB correspond to nucleic acid sequence entries in GenBank are prime examples of semantic data matching. As with semantic schema matching, the creation of links between semantically related data items can be greatly aided by a comprehensive thesaurus of terms accompanying highly structured data sources. Often though, the formation of links across data sources at the data level requires deep domain knowledge and the use of heuristic techniques to "guess" at the links. Surprisingly, considerable progress has been made in this area as evidenced by the abstracts in this year's meeting [20]. We attribute the rapid pace in creating linking tables for biological data sources to the pragmatic approaches taken by bioinformatics systems developers in attacking integration at its most fundamental level; many of the day-to-day questions biologists pose can be answered when linking tables are available regardless of the completeness of the rest of the integration process.

3 Dimensions of Integration

A number of examples of integrated databases within the biological community have emerged over the past few years. These solutions can be roughly classified as points in a plane defined by two axes: 1. Loosely to tightly federated; 2. Instantiation to virtual (view).

Degree of Federation. The first axis deals primarily with how many of the tasks of DB integration have been performed, and in how much detail. A *tightly* federated database is one in which the schemas have been transformed into a common model, and as much semantic schema matching and schema integration has been performed as possible. Data transformations and semantic data matching has also been performed. In a *loosely* federated database, there is a common data model but the integrated schema may be nothing more than the union of the underlying schemas. Furthermore, while there is a common language for querying against the federation the user may have to be aware of where the data is coming from since no naming conflicts have been resolved.

The advantages of a tight federation from the user's point of view are numerous, basically addressing all end-user and application developer desiderata: a single data representation (schema) uniting all relevant data sources enormously reduces the task of navigating the semantic tar pits of the multiple databases; and the system presents a single, uniform query language for the software developer and end-user upon which simple views and graphical interfaces can easily be built. The disadvantages are that it is very costly to build, requiring all of the steps of integration (the most costly of which are the semantic schema and data matching steps), and it is costly to maintain. Every time a schema evolves (which happens quite rapidly in the context of biological databases), the integration must be modified. Another disadvantage is that it is inflexible. As mentioned earlier, the representation of concepts within a database depends on their intended usage. Thus providing different views may involve writing several different transformations of the underlying data sources.

Instantiated versus View. The second axis deals with whether there exists a physical copy of the integrated database, or whether the integration describes how to translate queries against a view into queries against the underlying data sources.

The advantages to an instantiated implementation is that system performance will, in general, be much better than is possible under a distributed environment. First of all, query optimization can be performed locally (assuming the system is a good one); second, inter-data source communication time will be eliminated. System reliability should also be significantly higher since there are fewer dependencies on network connectivity and the individual data sources (again, assuming that the integrated system is reliable). It is also less costly to implement any inter-database constraints that have been determined in the integration [26, 30]. The disadvantages of an instantiated implementation is not just the initial cost, but even more the cost of maintenance. Every time an update is made in one of the underlying databases it must be reflected in the integrated instance. This may involve a complex query resulting in an update rather than just a simple update, if the integrated schema is complex. This is related to the problem of optimizing queries against views involving joins, and whether a "materialized view" is cost effective [31]; this will be discussed in more detail in section 5.

Before discussing particular approaches that have been taken, it is important to distinguish between *examples* (or solutions) and *methodology.* That is, an example can be implemented using a particular methodology and be "tightly integrated and instantiated"; however, it may be possible to use that same methodology and implement a "loosely integrated and view" solution. The more flexible a methodology is within this solution space, the more useful it will be since it will admit a greater range of solutions. It is also typically the case that a methodology which admits a tightly integrated solution can also be used to implement a loosely integrated solution, and that one which admits a view implementation can also be used to implement an instantiated implementation – but not vice versa. Thus a methodology for tightly integrated views is extremely flexible.

Much of the research within the database community has been to develop methodologies for performing one or more of the integration tasks. For example, Multibase [11] is a methodology for implementing view integrations, and suggests a limited family of schema transformations for integrating schemas. The underlying model is functional, and the query language is Daplex. A solution using Multibase could range from fairly loose to very tight integration, but always a view implementation. Furthermore, since there is no discussion in [11] of the first step of integration, translation from a given model into the functional model, it is immediately applicable only for functional databases which support Daplex.

The federated architecture suggested in [14] and later in [18] is a methodology which always yields a loosely federated system with a view implementation for relational systems. The focus in [14] is on how to gain and grant access permissions for data in a distributed system, while that in [18] is how to extend an SQL-like language to facilitate expressions over multiple sources with common names. There also exist a variety of methodologies which address only the schema transformation and integration aspects of integration [22, 24, 7, 28, 4]. Techniques that address primarily implementing schema transformations can be found in [22, 3, 21, 28, 2]. A discussion of transformations between ER-model and the relational model can be found in [19].

Before turning to examples of integration of biological databases, we should also remark that the cost of integration is also influenced by whether it is an *autonomous* or *cooperative* integration effort. Database researchers generally assume that each of the individual data sources in a federation are developing autonomously; however, cooperative development, where individual data sources agree on the structure of overlapping schema fragments, a common thesaurus and unique identifiers linked across sources, can substantially reduce the barriers to integration even if heterogeneous data models are chosen. If the developers further agree on a common data model, then the cost of integration can be almost fully absorbed into the implementation and maintenance cost of the individual data sources. While the issue of "autonomous versus cooperative" strategies addresses a sociological rather than technical barrier, it provides an important mechanism for practical data source coordination and integration that can substantially affect the overall cost of integration.

4 Sample Solutions and Cost

Tight/Materialized/Autonomous Integration: The Integrated Genome Database (IGD) developed by Ritter [25] represents the most demanding approach to integration: it is an example of a tightly federated, fully materialized database, where each of the twenty or so individual resources is fully autonomous with respect to the integration. IGD uses an ACeDB schema and front-end. The underlying database can either be in the ACeDB management system or (as we understand it) in Sybase.

IGD has been favorably received. It provides a common schema for the underlying databases, a popular graphical user interface (ACeDB), and the ACeDB query facility. Since most updates to ACeDB databases are through text files rather than through the transaction management systems expected in full blown DBMSs, daily bulk updates are efficient because little constraint checking is enforced.

The integration physically resides at one site, can be queried without remote database accesses and thus should provide rapid access to data. However, one can imagine a scenario in which queries against a virtual IGD schema are translated into queries against the underlying data sources from which IGD was constructed.

Several issues concerning the long-term viability of IGD in its present form arise. First, IGD may well be pushing ACeDB past its design limits with respect to the database size and performance. Anecdotal evidence suggests that ACeDB databases approaching 1GB are unreliable and require subverting the generality of the system name space to be implemented. The second and more pressing issue is the overall cost of system maintenance. It is not clear what tools have been built in the IGD project to cope with schema and data evolution. At this point in time, molecular biology databases are modest in size and bulk reloads of whole database are feasible. This will not be true in the future where incremental uploads will be unavoidable. At that point, issues of schema evolution, data level maintenance and linking table maintenance become paramount. While these issues are by no means restricted to IGD, we are not sure how IGD plans to address these problems.

Tight/View/Cooperative Integration: GDB, GSDB and TIGR have proposed a cooperative federation using a homogeneous DBMS environment, Sybase. While we do not know the details of this collaboration, we assume that an effort will be made to coordinate schema design; that linking tables will be maintained relating data instances across the cooperation; and that updates while primary updates will be performed locally at each site, dependencies at other sites will be immediately updated.

The advantages of this controlled environment are obvious: shared schemas, a homogeneous, industrial strength DBMS capable of distributed queries, common name spaces and identifiers, cooperatively developed linking tables, and both local and distributed updates. However, while this solution is perfectly reasonable for a small-scale cooperative effort, it does nothing to address the larger problem of data source integration. Further, it is not clear how far this top-down approach can scale. Imagine trying to integrate the twenty data sources available in IGD by a cooperative approach.

Loose/Partial Views/Autonomous Integration: The CPL/Kleisli system [6] supports ad hoc queries over heterogeneous, distributed databases. This system could easily be inserted at many points in the space of integration possibilities. To date, however, it has been used to integrate autonomous, read-only resources through trans-database user-views (mediators). In this mode, CPL/Kleisli offers the following advantages: a uniform interface to heterogeneous systems; inexpensive construction and relatively inexpensive maintenance of complex queries across multiple database; uniform treatment of heterogeneous resources and databank analysis algorithms (e.g., BLAST); optimization of distributed queries including parallelism and lazy evaluation; a rich type system necessary for integrating heterogeneous resources; and modularization of data drivers for access to distributed resources.

However, there is a significant practical drawback to this mediator style of integration. Recent experiments with the CPL system have shown that the existing network system is too fragile and too slow to permit adequate response times for many reasonable distributed queries. This of course depends strongly on the particular resource being accessed; bulk queries against the Entrez server are intolerable while those against relational database systems are fast, robust and can be parallelized for significant performance improvements. Furthermore, while updates to individual underlying systems can be executed within the CPL-Morphase system [10], updates at a global level is currently not supported.

Links/Views/Autonomous Integration: Several examples of link driven federations now exist. Among them are SRS [12], the Genera system developed by Letovsky, the ExPASy WWW server by Bairoch and Genome Net. The interface basically allows a set of data sources to be browsed by casual end-user; users execute single data source retrievals and then hop to another data source via Web links. While these have proven to be an initial boon to biologists they do not scale well, as they do not provide "machine access" to the underlying data sources. It is also not clear whether the "point and navigate" style of access really constitutes an interface, since the functionality is extremely limited. Such an interface is also extremely inefficient and does not support bulk queries.

5 New Technical Challenges

There are several new technical challenges that need to be addressed. Many lie with the central concern of querying the federation; others lie in the yet unexplored concern of updating the federation.

New Technical Challenges with Querying. The fundamental challenge with querying is to develop an appropriate data model for the federation. Within the current environment of scientific databases, the data types are complex – for example, ASN.1 has arbitrarily embedded records, sets, lists and variants. Other formats include arrays. Although it is possible to map these into a much simpler type system, say relational, a simpler representation loses the "intuition" and often the semantics of the original representation For example, mapping a list of strings into a relation will lose positional information. One could represent it as a set of of pairs representing the string and its position within the list, however the representation is not as intuitive and translation of list operations unnatural. Current commercial products and many research prototypes use an extremely simple type systems (e.g. are mainly for relational databases) and are not appropriate.

Another technical challenge is to "add value" to existing query interfaces. Although some biological data sources have sophisticated access techniques and implement projections as well as selections of data, others do not, such as ASN.1 and ACeDB. For example, while Entrez queries allow the selection of a complex value from an ASN.1 source, they do not allow any pruning or field selection from that value. Although such pruning could be done to an ASN.1 value after it has been retrieved by the federation server, it is much less expensive in terms of communication costs to prune at the level of the ASN.1 driver, i.e. the interface that translates from ASN.1 values into values of whatever model is being used for the federation. Placing this software at the ASN.1 server would significantly reduce the amount of data transmitted over the network.

Crucial to the success of any system that deals with large amounts of data, and probably the most difficult challenge for querying, is the ability to perform optimizations. For example, the relational model took a number of years to hit the commercial market and gain widespread acceptance because performance was an initial problem. Optimization techniques within this model that were subsequently developed include [29]: algebraic rewrite rules, such as those that push "restrictive" operators (i.e. selections and projections) close to base relations, and the translation of selections followed by a cartesian product into a join; various data placement strategies, such as clustering, and the development of indexing, such as B-trees and hash structures; optimization of selections to use indices where appropriate; various optimizations of joins, such as blocked nested-loop join [16] and hashed-loop joins [23]. While many of these optimizations fall outside of the semantics of the model itself, the existence of algebraic rewrite rule simplifies the process enormously since commutativity of various "positive" operators (cartesian product and union) is assured.

Optimizations are even more important in a distributed environment due to the latency and unreliability of communication as well as the varied capabilities of the data servers, and therefore admit more possibilities. Examples of the types of optimizations possible in such an environment follow:

1. The ability to use rewrite rules in whatever high-level query language is adopted for the federation can dramatically improve performance if operations are pushed down to the source databases whenever possible. For example, if the federation language were SQL, and R and S were two relations at the same (relational) data source, it would be significantly more efficient to execute the join of R and S, $R \times S$, completely at the data source than to first retrieve R, then retrieve S and then compute the join at the federation server.

   Within the CPL query system developed by our group [6] the following query retrieves all known DNA sequences on chromosome [22].

   ```
   define Loci22 == {[locus_symbol= x, genbank_ref= y] |
   ```

```
[locus_symbol=\x,locus_id=\a, ...]
    <- GDB_Tab("locus"),
[genbank_ref=\y,object_id=a,object_class_key=1, ...]
    <- GDB_Tab("object_genbank_eref"),
[loc_cyto_chrom_num="22", locus_cyto_location_id=a,
   ...]  <- GDB_Tab("locus_cyto_location")};
```

As written, the query first accesses the locus table in GDB via the call GDB_Tab("locus"). For each tuple in the table, it then pulls out entries in object_genbank_ref whose object_id field matches the locus_id field of the locus tuple (indicated by GDB_Tab("object_genbank_ref") and the reuse of variable a). It then pulls out entries in locus_cyto_location whose locus_cyto_location_id field matches the locus_id field of the locus tuple. Using rewrite rules for CPL, this inefficient query which accesses GDB multiple times is automatically converted into a single (more complex) access against GDB:

```
define GDB ==
   Open-Sybase([server="GDB",user="cbil",
                    password="bogus" ]);

define Loci22 ==
    GDB([query=
       "select locus_symbol, genbank_ref
       from locus, object_genbank_ref, locus_cyto_location
       where locus.locus_id = locus_cyto_location_id
       and locus.locus_id = object_genbank_eref.object_id
       and object_class_key = 1
       and loc_cyto_chrom_num = '22' "]);
```

2. The ability to implement various join strategies at the federation server, for joins that are distributed (i.e. cannot be pushed down to a single data server). There has been quite a bit of research on semi-join strategies within the (non-heterogeneous) distributed database community [8]. The problem becomes even more interesting when the capabilities of the data servers themselves are also taken into consideration. Taking the example of computing $R \times S$ where R and S are at different data sources, if R is very small (a few tuples), S is very big and has indexing capabilities, it would cost effective to send each join value of R to the server containing S since communication costs would be reduced. However, if the server containing S didn't have indexing capabilities the whole relation S would have to be retrieved.

3. The ability to exploit parallelism at the data servers. Taking the $R \times S$ example again, rather than sending join values of R to the server containing S sequentially, we can exploit the fact that many data servers can handle several requests simultaneously and send the join values in parallel.

4. The ability to exploit redundancy within the federation. There is frequently more than one data source containing the same information. For example, IGD contains complete

replications of about 20 important databases. For a query that accesses data within one of these databases, it might be useful to be able to fire off queries *in parallel* against IGD and the original data source, and take whichever one responds first. Or it might be possible to poll each of the servers to determine their load and network conditions, and send the query to whichever server seems to be the most available. Additionally, the capabilities of the servers could be taken into account, and the server with the greatest capability for answering the query could be chosen. For example, an ASN.1 server might provide faster response to a query than a Sybase server; however, if projection was important in the query one might prefer the Sybase server.

New Technical Challenges with Data-Level Updates: In centralized or (homogeneous) distributed databases, updates are traditionally handled by enclosing them within transactions. Transactions give a number of "guarantees", named the ACID properties: 1) Atomicity: a transaction is either executed in entirety, or not at all; 2) Consistency: transactions preserve database consistency, i.e. transform an initially consistent database state into another consistent database state; 3) Isolation: transactions are isolated from one another, i.e. the updates of an uncommitted transaction cannot be observed by another transaction; 4) Durability: the effects of a transaction persist despite system crashes. However, due to the dramatic variation in update capabilities of the underlying data servers, notably the ability to enter a "prepared-to-commit" state, it is impossible to enforce this strong a notion of correctness for updates within a view federation [5]. For example, Sybase servers have a (relatively) strict implementation of transactions, while ACeDB servers require single-user, sequential modifications to the database file.

However, it is likely that one would like to state a number of integrity constraints that must hold for the federation. For example, an integrity constraint could be used to state that replicated or derived data is not contradictory, or at least that updates to replicated data are current within some window of time. Other integrity constraints might be used to state that information is not duplicated or that referential integrity holds (e.g. all genbank_ref within GDB occur within some GenBank entry). We need to formulate the requirements of updating within federations of molecular biology databases, and develop/adapt approaches to specifying and enforcing such constraints. Advances in this area are being made in the database community [13], and need to be extended to our environment.

Updating within an instantiated integrated database also poses problems: As the underlying data sources are updated, the "view" or instantiation must also be updated. Rather than recomputing the view each time an update occurs, one should minimize the amount of work necessary to make the view current and only update the part that actually depends on the update. Although much work has been done on the view maintenance problem in the database community, updating an integrated repository is significantly more complicated since the underlying data sources presumably do not understand view management and know the relevant view definitions to propagate updates as they occur. Thus the integrated source may have to perform distributed computations to compute the correct update; however, due to the lack of transaction management at a global level, it may not be possible to get temporally consistent data and the integrated source may compute an incorrect view (see [31] for details). Again, we need to formulate requirements and develop techniques to correctly update instantiated integrated databases.

New Technical Challenges with Structural Updates: It is well known that the schemas of databases within the domain of molecular biology evolve extremely rapidly in response to changing experimental techniques and requirements, perhaps as often as 2 or 3 times per year. This poses a number of problems with maintaining view definitions and existing applications, since it is difficult to determine how the schema transformations, or structural updates, affect these (frequently complex) definitions. It is therefore useful to be able to capture schema transformations in a high-level declarative language so that they can be easily modified. Furthermore, capturing integrity constraints in a common paradigm potentially gives us a mechanism for checking the correctness, or information preservation, of such transformations [21, 15, 17].

6 Conclusions

Some of the goals of the Biomatrix have been accomplished. There exist some linking tables relating objects in different databases, a first step in semantic integration. There exist some physical integrations of important overlapping biological databases. There exist tools for querying and transforming data in heterogeneous data sources that are currently being used within some centers. There is ongoing work on resolving data conflicts.

However, there remain many technical and organizational challenges which were alluded to in this report. Are there strategies for helping with semantic schema matching? Are there general strategies for resolving data conflicts? What data-types should be included in the common data model? What are the natural operations over those types? What are useful optimizations for queries over in a language based on this type? What are useful source and network specific optimizations for distributed queries over heterogeneous data sources? How should updates be modeled in a distributed Biomatrix, and how should they be implemented?

References

[1] Abiteboul, S., and Hull, R. IFO: A formal semantic database model. *ACM Transactions on Database Systems 12*, 4 (December 1987), 525–565.

[2] Abiteboul, S., and Hull, R. Restructuring hierarchical database objects. *Theoretical Computer Science 62* (1988), 3–38.

[3] Banerjee, J., Kim, W., Kim, H., and Korth, H. Semantics and implementation of schema evolution in object-oriented databases. *SIGMOD Record 16*, 3 (1987), 311–322.

[4] Batini, C., and Lenzerini, M. A methodology for data schema integration in the entity-relationship model. *IEEE Transactions on Software Engineering SE-10*, 6 (November 1984), 650–663.

[5] Breitbart, Y., Garcia-Molina, H., and Silberschatz, A. Overview of multidatabase transaction management. *VLDB Journal 1*, 2 (October 1992).

[6] Buneman, P., Davidson, S., Hart, K., Overton, C., and Wong, L. A data transformation system for biological data sources. In *Proceedings of 21st VLDB* (September 1995). Also Technical report MS-CIS-95-10, Dept. of Computer and Information Science, University of Pennsylvania. March 1995.

[7] Buneman, P., Davidson, S., and Kosky, A. Theoretical aspects of schema merging. In *LNCS 580: Advances in Database Technology — EDBT '92* (1992), Springer-Verlag, pp. 152–167.

[8] Ceri, S., and Pelagatti, G. *Distributed Databases: Principles and Systems.* McGraw Hill, 1984.

[9] Committee on Models for Biomedical Research. *Models for Biomedical Research: A New Perspective.* National Academy Press, Washington, D.C., 1985.

[10] Davidson, S., Hara, C., and Kosky, A. Morphing sparsely populated data, July 1995. Available at url `http://www.cis.upenn.edu/~kosky/mimbd95.html`.

[11] Dayal, U., and Hwang, H. View definition and generalisation for database integration in Multibase: A system for heterogeneous distributed databases. *IEEE Transactions on Software Engineering SE–10*, 6 (November 1984), 628–644.

[12] Etzold, T., and Argos, P. Srs: an indexing and retrieval tool for flat file data libraries. *Computer Applications of Biosciences 9* (1993), 49–57.

[13] Grefen, P., and Widom, J. Integrity checking in federated databases. Tech. rep., Department of Computer Science, Stanford University, 1994.

[14] Heimbigner, D., and McLeod, D. A federated architecture for information management. *ACM Transactions on Office Information Systems 3*, 3 (July 1985).

[15] Hull, R. Relative information capacity of simple relational database schemata. *SIAM Journal of Computing 15*, 3 (August 1986), 865–886.

[16] Kim, W. A new way to compute the product and join of relations. In *Proceedings of ACM SIGMOD International Conference on Management of Data* (1980), pp. 179–187.

[17] Kosky, A., Davidson, S., and Buneman, P. Semantics of database transformations. Tech. rep., Department of Computer and Information Science, University of Pennsylvanic, Philadelphia, PA 19104-6389, July 1995.

[18] Litwin, W., Mark, L., and Roussopoulos, N. Interoperability of multiple autonomous databases. *ACM Computing Surveys 22*, 3 (September 1990), 267–293.

[19] MARKOWITZ, V., AND SHOSHANI, A. Representing extended entity-relationship structures in relational databases: A modular approach. *ACM Transactions on Database Systems 17*, 3 (September 1992).

[20] MEETING ON INTERCONNECTION OF MOLECULAR BIOLOGY DATABASES. Abstract of participants, July 1995.

[21] MILLER, R. J., IOANNIDIS, Y. E., AND RAMAKRISHNAN, R. Schema equivalence in heterogeneous systems: Bridging theory and practice. *Information Systems 19* (1994).

[22] MOTRO, A. Superviews: Virtual integration of multiple databases. *IEEE Transactions on Software Engineering SE-13*, 7 (July 1987), 785–798.

[23] NAKAYAMA, M., KITSUREGAWA, M., AND TAKAGI, M. Hash-partitioned join method using dynamic destaging strategy. In *Proceedings of Conference on Very Large Databases* (1988), pp. 468–478.

[24] NAVATHE, S., ELMASRI, R., AND LARSON, J. Integrating user views in database design. *IEEE Computer 19*, 1 (January 1986), 50–62.

[25] RITTER, O., KOCAB, P., SENGER, M., WOLF, D., AND SUHAI, S. Prototype implementation of the integrated genomic database. *Computers and Biomedical Research 27* (1994), 97–115.

[26] RUSINKIEWICZ, M., SHETH, A., AND KARABATIS, G. Specifying interdatabase dependencies in a multidatabase environment. *IEEE Computer* (December 1991).

[27] SALTOR, F., CASTELLANOS, M., AND GARCIA-SOLACO, M. Suitability of data models as canonical models for federated databases. *SIGMOD Record 20*, 4 (December 1991), 44–48.

[28] SHOVAL, P., AND ZOHN, S. Binary-relationship integration methodology. *Data and Knowledge Engineering 6* (1991), 225–249.

[29] ULLMAN, J. D. *Principles of Database and Knowledgebase Systems I.* Computer Science Press, Rockville, MD 20850, 1989.

[30] WIEDERHOLD, G., AND QIAN, X. Modeling asynchrony in distributed databases. *Proc. 1987 International Conference on Data Engineering* (1987), 246–250.

[31] ZHUGE, Y., GARCIA-MOLINA, H., HAMMER, J., AND J.WIDOM. View maintenance in a warehousing environment. In *Proceedings of ACM SIGMOD Conference* (June 1995), pp. 316–327.