# Challenges in Integrating Biological Data Sources

JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 2, Number 4, 1995
Mary Ann Liebert, Inc.
Pp. 557–572

# Challenges in Integrating Biological Data Sources

S.B. DAVIDSON,[1] C. OVERTON,[1,2] and P. BUNEMAN[1]

## ABSTRACT

**Scientific data of importance to biologists reside in a number of different data sources, such as GenBank, GSDB, SWISS-PROT, EMBL, and OMIM, among many others. Some of these data sources are conventional databases implemented using database management systems (DBMSs) and others are structured files maintained in a number of different formats (e.g., ASN.1 and ACE). In addition, software packages such as sequence analysis packages (e.g., BLAST and FASTA) produce data and can therefore be viewed as data sources. To counter the increasing dispersion and heterogeneity of data, different approaches to integrating these data sources are appearing throughout the bioinformatics community. This paper surveys the technical challenges to integration, classifies the approaches, and critiques the available tools and methodologies.**

**Key words:** molecular biology databases, database integration, database federation, database transformation.

## INTRODUCTION

IN THE 1985 NATIONAL ACADEMY OF SCIENCES REPORT, "Models for Biomedical Research: A New Perspective," Morowitz *et al.* (Committee on Models for Biomedical Research, 1985) argue that biological research has 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 10 years, the rate at which data have been generated has, if anything, exceeded the expectations of that report, but fortunately the data are mostly available online, overcoming at least one of the major hurdles in creating the Biomatrix.

However, despite substantial efforts at creating databases that unify genomic data [see, for example, Ritter *et al.* (1994)], we have to ask in what sense the ultimate goal of the creating the Biomatrix 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 schema evolution is a major challenge to database and knowledge base technology.

[1]Department of Computer and Information Science and [2]Department of Genetics, University of Pennsylvania, Philadelphia, PA 19104.

- The scope of the Biomatrix 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. And even if a complete Biomatrix could be developed, how would this relate to other, nonbiological, 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 a Biomatrix? The administrative problems alone appear unsurmountable.

Thus, while the Biomatrix calls for 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 aspects, not all of which have been addressed by current database technology. First, there is a collection of interrelated data in heterogeneous data sources that are connected over the internet. Some of these data sources are standard databases implemented using database management systems (DBMSs) 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 produce data and can therefore be viewed as data sources. 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 graphical user interfaces (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 may rise with technological advances in networking. Ultimately, even if the query interface at the most basic user-level is a fill-in-the-blanks forms interface, internally it should translate into a well-defined, high-level query language that is capable of supporting arbitrary queries, and which is complemented by an optimizer to provide answers to ad-hoc queries 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 modes of operation are informal update paths throughout a network of data sources.

Various approaches to data source integration are appearing throughout the bioinformatics community and it is unlikely that there will be a single satisfactory strategy. In this paper, 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 data source integration: (1) transformation of the various schemas to a common data model, (2) matching of semantically related schema components, (3) schema integration, (4) transformation of data to the global schema, either in response to queries or to create an instance of the global schema, 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 successfully addressed 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 integration and the "degree" of materialization, 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.

## STEPS IN DATA SOURCE INTEGRATION

The objective of integration is to make data distributed over a number of distinct, heterogeneous data sources accessible via a single interface. Having constructed the common 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 common interface; the other component will be discussed in later sections.

To begin, we define the terminology used throughout the paper. We have tried to adhere to standard definitions found in the computer science literature where they are unambiguous; however, as with many disciplines, usage often becomes muddled with time as terms are coopted by different communities. We will point out areas of ambiguous usage where relevant.

- A **database management system** (DBMS) is software that manages a large collection of persistent data. It supports a data model, or abstraction through which a user can view the data, as well as high-level languages that allow the user to define the structure of data, access data, and update data. It also typically provides transaction management to control concurrent access to the database, index structures to define efficient access paths to data, access control to limit access to data to authorized users, and resiliency to system failures (Ullman, 1989).
- The general term **data source** refers to any resource in which data of interest resides or is generated and which can be queried. Data sources include databases in various data models and implemented using various DBMSs (e.g., Sybase, a relational DBMS), custom systems (ACeDB and NCBI-ASN.1/Entrez), flat files (GenBank flat-file format), and applications such as BLAST or GRAIL that produce data in response to input. For the most part we are interested in integrating data sources, and will generally use the term "data source" in preference to the term "database" unless the data source in question is known to be a database.
- A **data model** is the collection of data types, constraints, and available operations on data. As an example, the relational data model has one type, the relation. A relation is a set of tuples, or mappings from attributes names to values in very simple domains (e.g., string, integer, date, boolean). The operations are theoretically defined by the relational algebra, although the commercial standard relational query language is SQL. Constraints that can be explicitly represented using the data definition language of most relational DBMS are keys and sometimes inclusion constraints, although a richer class can be specified by a logic-based constraint language.
- A conceptual **schema,** which we will refer to as simply a schema, is a representation of the real world entities and their relationships captured in a particular data source. A schema is expressed in a data definition language that is not procedural but is instead descriptive. The ACeDB community uses "data model" to mean "conceptual schema."
- **Federated database system** is a term first introduced in Heimbigner and McLeod (1985) and later crystallized in Sheth and Larson (1990). A federated database system is a collection of cooperating but autonomous component databases that are integrated to various degrees. The software that provides controlled and coordinated manipulation of the component databases is called a federated database management system. Since there are few, if any, true examples of federated database systems that are appropriate for or being used to integrate biological data sources, we avoid using this term throughout the remainder of this paper. Rather than discussing integration in the abstract, we present a simple example to make the steps concrete.

*Example 1*

Figure 1 shows two partial schemas representing data used in constructing YAC (yeast artificial chromosome) physical maps of chromosomes.[1] YACs contain large fragments of randomly chosen chromosomal DNA, typically on the order of $10^6$ bases in length, which, ideally, create a contiguous tiling path along the chromosome when ordered. Often there are gaps in the path resulting in smaller contiguous stretches, termed contigs, that cannot be joined. Contigs are built up by determining overlaps between pairs of YACs. The partial schemas model two possible experimental methods for determining overlap. (These schemas represent methods used primarily for verification rather than production contig assembly.)

The first schema represents the results of cross-hybridization experiments between a set of YACs and a probe made from an individual YAC. The probe, called an AluPCR product, is a set of labeled subsequences generated by priming between Alu repeats located along the selected YAC. Each AluPCR product is derived

[1] The graphic notation used here is inspired by Abiteboul and Hull (1987).

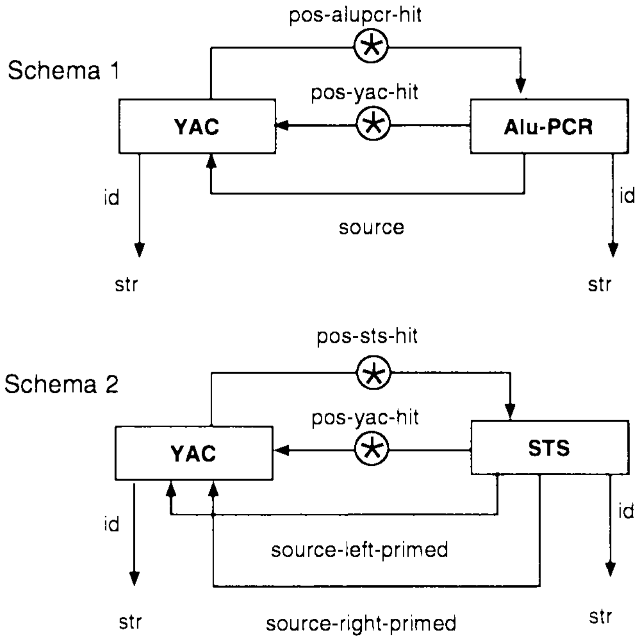

**FIG. 1.** Schemas for YAC/STS and YAC/Alu-PCR databases.

from one YAC, but many different YACs can cross-hybridize to the AluPCR and thus overlap with the source YAC. The schema corresponding to the experimental protocol has two classes: *YAC* and *Alu-PCR*. 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.

Similarly, the second schema represents an experimental protocol in which sequence tagged sites (STSs) generated from the very ends of a source YAC are used to detect overlapping YACs. In this case, the ends of the source YAC insert are relative to a known insert site in the YAC vector. The two sides of the insert site are named "left" and "right." Note that for each STS only one of "left" or "right" is valid (i.e., the other is null). As in Schema 1, this schema also has two classes: *YAC* and *STS*. The *YAC* class has attributes representing its unique ID and positive experimental hits with STSs, paralleling the ones in the *YAC* class for Schema 1. The *STS* class has four attributes representing its unique ID, the set of YACs for which it is positive, and the "left" and "right" ends of the YAC insert relative to the insert site.

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. 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* that 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 data source integration can be seen to consist of five conceptual tasks:

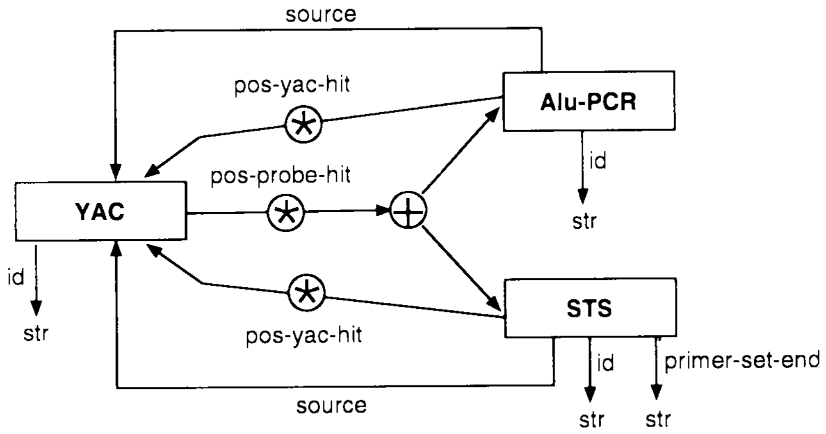

**FIG. 2.** An integrated schema of YAC/STS and YAC/Alu-PCR results.

- data model transformation,
- semantic schema matching,
- schema integration,
- data transformation, and
- semantic data matching.

We expand on what we mean by each of these, and note that the steps are interrelated.

### *Data model transformation*

Bearing in mind that we are extracting *portions* of the underlying data sources for integration rather than necessarily dealing with *complete* data sources, the first step in integration is to transform each of the underlying schemas into a common data model. Obviously, the transformation should preserve the relevant information. The common data model must also be at least as expressive as any of the individual data models. By expressive, we mean that there is a natural correspondence of elements of the original data model to elements of the common data model. For example, it is possible to represent almost any data structure within the relational data model, however, the encoding is often elaborate and obtuse. Furthermore, the common data model should be sufficiently simple and expressive to allow data to be represented in multiple ways so that conflicts between alternative representations can be resolved. 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 data model must be able to support both representations.

Various data models have been proposed within the computer science community for use as the common data model, ranging from relational and extended entity-relationship data models to semantic, functional, and object-oriented data models. In Saltor *et al.* (1991) the various requirements for a data model for transforming heterogeneous databases are examined, and the authors conclude that a data model supporting complex data-structures (sets, records, and variants), object-identity, and specialization and generalization relations between object classes is desirable.

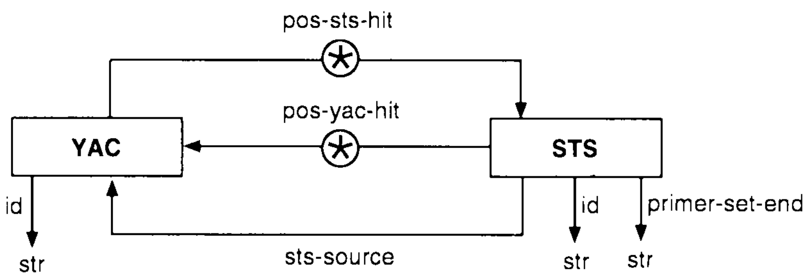

**FIG. 3.** A modified schema for a YAC/STS experiment results.

The transformation between data models needs to take into account information loss and equivalence of representation. For example, in an entity-relationship diagram (ER-diagram) many constraints on the data are implicitly represented, for example, arrows on links or cardinalities on edges. In transforming an ER-diagram to a relational schema these implicit constraints become explicit, i.e., must be captured as functional dependencies or assertions in some constraint language [see Markowitz and Shoshani (1992) for details]. Otherwise, updates that may be allowed within the relational instance would not be allowed within the ER instance, and a reverse mapping from the relational schema will not produce an equivalent ER schema to the original one.

### *Semantic schema matching*

Frequently, before a global schema is created, correspondences among concepts with semantic overlap are established across the component data sources. This is typically done by resolving naming conflicts between the schemas (whether homonym or synonym conflicts), or creating specializations and generalizations of concepts to show similarities among terms. 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 problems 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 data sources and establishing standardized thesauri of biological concepts (see the working group reports for the Meeting on Interconnection of Molecular Biology Databases, 1994). If done correctly, these steps in themselves may create a foundation for automated methods of semantic matching.

### *Schema integration*

At this point, a global schema can be created that represents the integration and into which the underlying (or component) data source schemas can be mapped. The mapping may be total with respect to a given component data source schema, meaning that all its schema components are reflected in the global schema, or partial, meaning that only a fragment of its schema components is reflected in the global schema. Furthermore, the global schema may be constructed in a bottom-up or top-down manner. By bottom-up we mean that the global schema is constructed explicitly from the component data source schemas. By top-down we mean that the global schema is decided first, and mappings from the underlying data source schemas into the global schema are determined subsequently. In either approach, additional information not present in the component data sources may be added to the global schema, for example, linking tables and annotations. We will consider this additional information as yet another component data source that is being mapped into the global schema. Either approach may also require further modifications of representation in the underlying schemas since concepts that overlap semantically are frequently represented in different ways in different schemas. Representations may differ for many reasons: for example, the underlying application requirements may be quite different. Even if they are the same, it is rare for different designers to come up with exactly the same design. 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 first necessary to perform a structural modification on the YAC/STS database 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. No information is lost in this transformation since at most one of "left" or "right" is valid for each STS. This yields an intermediate database with the schema shown in Figure 3.

Second, the way in which the integration itself will be used may be radically different from the ways in which the underlying data sources were designed, and it may therefore be necessary to add, modify, or delete concepts. As an example, it will generally be necessary to add explicit linking tables between concepts from different schemas that are related at some level; protein sequences in PIR are related to DNA sequences in GenBank, so a linking table between instances must be created in an integration of the two data sources.

Continuing with the YAC-Probe-Source example, the integration is fairly simple: We merely 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*

While structurally transforming schemas and mapping them into a global model, it is important to keep in mind that the changes specified at the *schema* level must be reflected in terms of the *data* in the underlying data sources. That is, the transformation and mapping should be unambiguous in its effect on the underlying data. However, it is frequently simpler to specify conceptual relationships between schema-level objects first and then specify the interpretation on the data-level objects—examples of this abound in the schema integration literature, but, unfortunately, sometimes the second step is skipped and the mapping remains badly defined. We therefore separate this step out from schema integration to underscore its importance.

For example, in a data model supporting classes of objects and optional attributes of classes, changing an attribute of an existing class from being optional to being required has several interpretations at the data level. 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. Note that these two interpretations are not equivalent and produce completely different instances. Which one is correct depends on what we intend to model in the modified schema and what our assumptions are. Continuing with the example, although the first transformation seems to be the most plausible (inserting a default value for the attribute wherever it is omitted), it could be that we intend to model only objects that have the attribute present, hence deleting the erroneous data is the correct transformation. It is, therefore, important that the meaning of this schema modification be represented explicitly as a data transformation. [See Kosky *et al.* (1995) for more details on data 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 abstracts at the 1995 Meeting on the Interconnection of Molecular Biology Databases (Meeting on Interconnection of Molecular Biology Databases, 1995). 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.

## DIMENSIONS OF INTEGRATION STRATEGIES

A number of examples of integrated data sources have emerged within the biological community over the past few years. These solutions can be roughly classified as points in a plane defined by two axes: (1) loosely to tightly integrated and (2) materialized to virtual (or view). See Figure 4. These axes, and how they characterize solutions, will be described next.

### *Degree of integration*

The first axis deals primarily with how many of the tasks of DB integration have been performed, and in how much detail. A *tightly* integrated solution is one in which the schemas have been transformed into schemas in a common data model, and a global schema has been created in which as much semantic schema

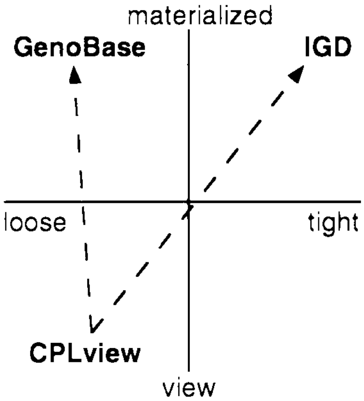

FIG. 4. Solution space for data source integration.

matching has been performed as possible. Data transformation and semantic data matching have also been performed, either by specifying the transformations and data matchings in some high-level language or procedure, or performed and stored in a special "integration data source" (this point will be revisited momentarily). In a *loosely* integrated solution, there is a common data model into which the underlying schemas have been translated, but the global schema is nothing more than the union of the underlying schemas. Although the underlying data sources have in some sense been integrated by providing a common interface to the underlying heterogeneous data sources, issues of semantic schema matching are left to the users since none has been performed by the administrator or creator of the system.

The advantages of tight integration from the user's point of view are numerous, basically addressing all end-user and application developer desiderata: a single data representation (schema) for all relevant data sources enormously reduces the task of navigating the semantic tar pits of multiple data sources; and the system presents a single, uniform query language for the software developer and end-user upon which simple views and graphic 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 frequently in the context of biological data sources), the integration must be modified. Another disadvantage is that it is inflexible. As mentioned earlier, the representation of concepts within a data source depends on their intended usage. Thus it may be necessary to provide multiple global schemas or multiple views on a single global schema.

### *Materialized versus view*

The second axis deals with whether there exists a physical (materialized) copy of the integrated solution at a single site, or whether the integration describes how to translate queries against a view into queries against the underlying distributed data sources.

The advantages of a materialized solution is that system performance will, in general, be much better than is possible in a heterogeneous environment. First, query optimization can be performed locally (assuming the system is a good one); second, interdata 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 system in which the materialized solution is stored is reliable). It is also less costly to implement any interdata source constraints that have been determined in the integration (Rusinkiewicz *et al.*, 1991; Wiederhold and Qian, 1987). The disadvantages of a materialized solution is not just the initial cost, but even more the cost of maintenance. Every time an update is made in one of the underlying data sources it must be reflected in the materialized 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 (Zhuge *et al.*, 1995); this will be discussed in more detail later.

The two axes are not entirely independent in our characterization. As we have described it, a tight integration encompasses semantic data matching and integration at the data level. Almost without exception, these steps demand the creation of new relationships that explicitly define links between related data items in the individual data sources. Ideally, such links would be computed on-the-fly as part of a query; however, it is most often the case that such links are expensive to determine and usually necessitate manual intervention. Thus once these links have been determined they are typically stored. Consequently, a tight integration almost always requires at least a partially materialized solution. It is also worth mentioning that tightly integrated, fully materialized solutions have been a common approach to integration, and are popularly called "data warehouses" (Zhuge *et al.,* 1995).

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 materialized"; 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 the solution space defined by the axes discussed above, the more useful it will be since it will admit a greater range of solutions. It is typically the case that a methodology that admits a tightly integrated solution can be used to implement a loosely integrated solution by ignoring the techniques or steps involved in semantic schema and data matching. Similarly, a methodology that admits a view solution can typically be used to implement an materialized solution by executing the view specification—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 (Dayal and Hwang, 1984) is a methodology for implementing view integrations, and suggests a limited family of schema transformations for integrating schemas. The underlying data 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 Dayal and Hwang (1984) of the first step of integration, translation from a given data model into the functional data model, it is immediately applicable only for functional databases that support Daplex.

The federated architecture suggested in Heimbigner and McLeod (1985) and later in Litwin *et al.* (1990) is a methodology that always yields a loosely federated system with a view implementation. The focus in Heimbigner and McLeod (1985) is on how to gain and grant access permissions for data in a distributed system, while that in Litwin *et al.* (1990) is how to extend an SQL-like language to facilitate expressions over multiple relational databases with common names. There also exist a variety of methodologies that address only the schema transformation and integration aspects of integration (Motro, 1987; Navathe *et al.,* 1986; Buneman *et al.,* 1992; Shoval and Zohn, 1991; Batini and Lenzerini, 1984). Techniques that address primarily implementing schema transformations can be found in (Motro, 1987; Banerjee *et al.,* 1987; Miller *et al.,* 1994; Shoval and Zohn, 1991; Abiteboul and Hull, 1988).

We should also remark that the cost of integration is 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 is 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. For example, it appears that GDB, GSDB, and TIGR have recently proposed a cooperative federation using a homogeneous DBMS environment, Sybase.

## SAMPLE SOLUTIONS AND COST

### *Tightly integrated, materialized solution*

The Integrated Genomic Database (IGD) developed by Ritter *et al.* (1994) represents the most demanding approach to integration: it is a tight, fully materialized integration (a data warehouse), where each of the 20 or so underlying data sources (including GDB, OMIM, EMBL, SWISS-PROT, RLDB, and DNA Probe Bank among others) is fully autonomous with respect to the integration. The global schema for IGD, expressed in ACeDB, is an example of a top-down integration. It was designed to model the domain of molecular biology rather than integrating the schemas of the underlying data sources. Semantic schema matching and data transformation were performed using conversion rules, as well as by using domain knowledge and the semantics of the underlying data source schemas. IGD is stored in multiple ACeDB sites.

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

IGD can be queried without remote data source 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 size and performance. Anecdotal evidence suggests that ACeDB data sources approaching one gigabyte are unreliable and require ways of subverting the generality of the system name space. 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 data sources are modest in size and bulk reloads of whole data sources are feasible. This will not be true in the future, and 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 [more information on IGD is available in Markowitz and Ritter (1995)].

### *Loosely integrated, materialized solution*

GenoBase (Overbeek *et al.* 1995) is a materialized, loose integration of the EMBL, SWISS-PROT, PROSITE, and Enzyme data sources. All data have been converted into a common data model, Prolog clauses. However, the conversion does not appear to have performed any semantic schema matching; the global schema is little more than the union of the underlying data source schemas translated into Prolog clauses. When necessary, GenoBase connects to outside servers to perform certains tasks, such as performing BLAST runs, looking up MEDLINE and PIR references in SWISS-PROT peptide objects, displaying Selkov EMP metabolic pathway drawings, or finding requested GenBank entries that are not in their encorporated version of EMBL. GenoBase is written in Quintus Prolog with C subroutines to handle file access and certain pattern-matching procedures.

The World Wide Web (WWW) interface allows users to view information for a given GenoBase object, move from object to object within GenoBase by traversing predefined connections set up between objects, or access one of the static tables automatically constructed from the current contents of GenoBase that lists various types of objects in some sort of order for easier user lookup and selection. GenoBase also has its own built-in Prolog-like query language. It permits queries of sequence data that are either extremely difficult and time-consuming, or simply impossible using GCG and other standard sequence analysis tools.

Although there is little discussion of how links have been created between the data sources that comprise GenoBase, one form of links present is built-in ties in the SWISS-PROT data source between a peptide and the EMBL locus that encodes it. For several selected genomes, they have also pinpointed the coding feature of the EMBL local entry that actually produced the peptide rather than having a link to the entire locus (which may include introns, miscellaneous "junk" DNA, or entire other genes). This has been done using automated comparison of the DNA sequences of all the coding features for each of the locus entries.

GenoBase appears to be extremely useful, but many of the comments about IGD also apply to GenoBase. Although Prolog-based approaches have been used by many researchers to make inferences about data, they typically do not scale to large amounts of data. GenoBase is currently roughly 3574 megabytes; it is hard to believe that it will scale to incorporate many more data sources.

### *Loosely integrated, view solution*

The CPL/Kleisli system (Buneman *et al.*, 1995) supports ad hoc queries over heterogeneous data sources. This system could easily be used at many points in the solution space, illustrated by the dotted arrows in Figure 4. To date, however, it has been used to loosely integrate autonomous, read-only resources through trans-data source views (mediators). In this mode, CPL/Kleisli offers the following advantages: a well-defined, high level language that can be used to query a wide variety of data sources, including but not limited to relational, ACeDB, and ASN.1, the ability to incorporate calls to software analysis packages such as BLAST and FASTA within queries, data drivers that access specific types of data sources, and a modular and extensible optimizer. Currently encorporated within the optimizer are rules that extend the optimizations commonly found in relational systems to the richer type systems found in biological data sources (such as ASN.1), as well as rules that exploit the parallelism inherent in a distributed system and various capabilities of the underlying systems.

However, there is a significant practical drawback to using the CPL methodology to create a view solution. Recent experiments with the current 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 resources 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. Note, however, that the CPL methodology could also be used to create a materialized solution.

### *Solutions that are not integrations*

A currently popular solution to the problem of accessing multiple heterogeneous data sources is to bypass any form of integration and merely provide links between related data sources. We do not consider these solutions to be true "integrations" since there is no global schema, and no uniform, high-level interface for navigating between the underlying data sources. Examples of this include SRS (Etzold and Argos, 1993), WebDBget (Akiyama *et al.*, 1995), the ExPASy WWW server by Bairoch, and Genome Net. The interfaces basically allow a set of data sources to be browsed by casual end-users; users execute single data source retrievals and then hop to other data sources via WWW links. Details of some of these systems follow.

SRS is an indexing and retrieval tool for flat file data sources that currently includes over 43 data sources, including SWISS-PROT, PIR, EMBL, PROSITE, EPD, MEDLINE, and LIMB. It allows simultaneous access to different data sources, and can create linked data using an indexing mechanism to cross-reference entries.

SRS has a command line interface with retrieval commands that includes logical operations on sets of data (basic selections) and links between sets of different data sources. Links in SRS are used to find references in other data sources. For example, "$A > B$" gives those entries in $B$ that are referenced by entries in $A$, and "$A < B$" gives entries in $A$ that reference entries in $B$. The index used to cross-reference entries is described as follows: (1) If there are links in both directions between two entries from different data sources, the same pairwise link index is used for both directions. (2) In the case of indirectly connected entries from two different data sources, an "optimal" or cheapest succession of direct links is used to create a link index entry. (3) If a set contains entries from different data sources, then the subsets of all data sources found in the set are linked independently.

WebDBget connects PRF, LITDB, PDBSTR, PMD, LIGAND, and AAIndex using a linking data source called LinkDB. Users of this system simply specify keywords in a search box, and are given the list of entries that contain matching keywords. A data source entry is obtained from this set by selecting an entry name in the list. Once an entry is obtained, users can further retrieve related entries in different data sources by clicking on the marked items. In particular, if the identifier of an entry is clicked on, a list of all the related entries is given. Related entries include the links provided by the original data sources, as well as reverse links and indirect links derived from multiple links.

Linking information in LinkDB (Goto *et al.*, 1995) is updated daily by extracting explicit links from data sources such as GenBank and SWISS-Prot. GenBank has links to MEDLINE, and SWISS-Prot has links to EMBL, PDB, PROSITE, etc. Links to EC numbers described in the DEFINITION lines of many data sources are also extracted, and used for establishing links to the LIGAND enzyme reaction data source. These explicit links are also reversed to create back-references, and indirect links are also constructed by following paths of links. Such indirect links enable users to retrieve, for example, SWISS-PROT and PDB entries from GenBank in one step.

While these primarily "browsing" systems have proven to be an initial boon to biologists, there are several limitations. First, the construction of links by following paths of links is not guaranteed to preserve biological meaning. Second, even the existing direct links may be erroneous. Several examples of this are cited in Goto *et al.* (1995): data version mismatches, parsing confusion (for example, EC number vs. other dotted notation), or external errors (for example, PDB to MEDLINE links are generated using external correspondence information). Third, browsing mechanisms do not scale well since they do not provide "machine access" to the underlying data sources. Fourth, this type of interface is extremely inefficient, does not support bulk queries, and cannot be optimized.

## NEW TECHNICAL CHALLENGES

There are several new technical challenges that remain to be addressed in constructing a satisfactory solution to the problem of integrating heterogeneous data sources. Many lie with the central concern of querying the integration; others lie in the yet unexplored concern of updating the data sources.

### *New technical challenges with querying*

The fundamental challenge with querying is to develop a language appropriate to the types expressed in the common data model. Within the current environment of scientific data sources, 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 pairs representing the string and its position within the list, however, the representation is not intuitive and the translation of list operations is unnatural. Current commercial products and many research prototypes use 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, such as ASN.1 and ACeDB, do not. 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 integration software, 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 data model is being used for the global schema. Placing this software at the ASN.1 server would significantly reduce the amount of data transmitted over the network. As another example, although the spreadsheet-like "Table Maker" interface of ACeDB supports field projection and class joins, this capability is a hack added to the language and is not fully compositional with other operators of the ACeDB language.

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 data 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 data model that were subsequently developed include (Ullman, 1989) 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 (Kim, 1980) and hashed-loop joins (Nakayama *et al.,* 1988). While many of these optimizations fall outside of the semantics of the data 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 with the global query language can dramatically improve performance if operations are pushed down to the data source whenever possible. For example, if the 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 $S$, $R \bowtie S$, completely at the data source than to first retrieve $R$, then retrieve $S$ and then compute $R \bowtie S$ at the server implementing the integration software.

Within the CPL query system developed by our group (Buneman *et al.*, 1995) 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 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 semijoin strategies within the (nonheterogeneous) distributed database community (Ceri and Pelagatti, 1984). The problem becomes even more interesting when the capabilities of the data servers themselves are also taken into consideration. Taking the example of computing $R \bowtie S$ 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 then be 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 $S$ did not have indexing capabilities the whole relation $S$ would have to be retrieved.
3. The ability to exploit parallelism at the data servers. Taking the $R \bowtie S$ 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 underlying data sources. There is frequently more than one data source containing the same information. For example, IGD contains complete replications of about 20 important data sources. For a query that accesses data within one of these data sources, 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 the data transmission is expensive then one might prefer the Sybase server to the ASN.1 server since the latter lacks the capability of performing projections.

### *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 integration (Breitbart *et al.*, 1992). For example, Sybase servers have a (relatively) strict implementation of transactions, while ACeDB servers require single-user, sequential modifications to the data file.

However, it is likely that one would like to state a number of integrity constraints that must hold for the integrated solution. For example, an integrity constraint could be used to state that replicated or derived data are 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 an integrated solution, and develop/adapt approaches to specifying and enforcing such constraints. Advances in this area are being made in the database community (Grefen and Widom, 1994), and need to be extended to the bioinformatics community.

Updating within a tightly integrated, materialized solution also poses problems: As the underlying data sources are updated, the materialization must also be updated. Rather than recomputing the materialization each time an update occurs, one should minimize the amount of work necessary to make the materialization current and only update the part that actually depends on the update. Although much work has been done on this "view maintenance" problem within the database community, updating a tightly integrated, materialized repository is significantly more complicated since the underlying data sources are autonomous. It will be the responsibility of the integrated, materialized data source 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 integration may compute an incorrect view [see Zhuge *et al.* (1995) for details]. Again, we need to formulate requirements and develop techniques to correctly update materialized, tightly integrated data sources.

### *New technical challenges with structural updates*

It is well known that the schemas of data sources within the domain of molecular biology evolve rapidly in response to changing experimental techniques and requirements, perhaps as often as two or three 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 (Miller *et al.*, 1994; Hull, 1986; Kosky *et al.*, 1995).

## CONCLUSIONS

Much progress has recently been made within the biology informatics community in providing integrated access to the large amounts of data that are increasingly available online. There exist linking tables relating objects in different data sources, a first step in semantic integration. There exist some physical integrations of important overlapping data sources. There exist tools for querying and transforming data in heterogeneous data sources that are geared toward applications in biology. There is ongoing work on resolving data conflicts. Given these various tools and resources, diverse approaches to solving data source integration problems are possible. It is unlikely that any one approach will be adopted as the right one for all applications. Rather, each approach has certain advantages and costs that must be understood and accepted in the context of the desired applications.

However, there also remain many unsolved technical and organizational challenges to which we have alluded. For example, 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 in a language based on this data model? What are useful source and network specific optimizations for distributed queries over heterogeneous data sources? How should updates be modeled and implemented? Answers to these questions will dramatically impact the usefulness and long-term viability of systems developed around the various approaches to integration outlined here.

While the Biomatrix as originally envisioned may never come to pass, the trends in technology and resources development are certainly encouraging. A likely scenario is that there will be many minibiomatrices, each customized for the needs of a particular user community. These will still require the same fundamental technologies for creating, maintaining, and querying an integrated resource, but may be vastly less expensive to build and maintain simply because, with a smaller number of component resources, the interconnectivity is reduced. We expect that in the near-term, the demand for small-scale integration will drive the development of the advanced technology and auxiliary resources (e.g., linking tables, and semantic resolution of concepts and data) that will make integration cost effective and efficient. Nonetheless, integration on the scale of Biomatrix remains problematic.

## ACKNOWLEDGMENTS

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.

## REFERENCES

Abiteboul, S., and Hull, R. 1987. IFO: A formal semantic database model. *ACM Trans. Database Syst.* 12(4), 525–565.

Abiteboul, S., and Hull, R. 1988. Restructuring hierarchical database objects. *Theoret. Comput. Sci.* 62, 3–38.

Akiyama, Y., Goto, S., Uchiyama, I., and Kanehisa, M. 1995. WebDBGET: An integrated database retrieval system which provides hyper-links among related database entries. In abstracts of participants of the 1995 Meeting on Interconnecting Molecular Biology Databases. Available at url http://www.genome.ad.jp/dbget/dbget.html.

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

Batini, C., and Lenzerini, M. 1984. A methodology for data schema integration in the entity-relationship model. *IEEE Trans. Software Eng.* SE-10(6), 650–663.

Breitbart, Y., Garcia-Molina, H., and Silberschatz, A. 1992. Overview of multidatabase transaction management. *VLDB J.* 1(2).

Buneman, P., Davidson, S., and Kosky, A. 1992. Theoretical aspects of schema merging. *LNCS 580: Adv. Database Technol.—EDBT '92* 152–167.

Buneman, P., Davidson, S., Hart, K., Overton, C., and Wong, L. 1995. A data transformation system for biological data sources. *Proc. 21st VLDB.* Also Technical Report MS-CIS-95-10, Department of Computer and Information Science, University of Pennsylvania, March.

Ceri, S., and Pelagatti, G. 1984. *Distributed Databases: Principles and Systems.* McGraw-Hill, New York.

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

Dayal, U., and Hwang, H. 1984. View definition and generalisation for database integration in Multibase: A system for heterogeneous distributed databases. *IEEE Trans. Software Eng.* SE-10(6), 628–644.

Etzold, T., and Argos, P. 1993. Srs: An indexing and retrieval tool for flat file data libraries. *Comput. Appl. Biosci.* 9, 49–57.

Goto, S., Akiyama, Y., and Kanehisa, M. 1995. LinkDB: A Database of Cross Links between Molecular Biology Databases. In abstracts of participants of the 1995 Meeting on Interconnecting Molecular Biology Databases. Available at url http://www.genome.ad.jp/dbget/dbget.html.

Grefen, P., and Widom, J. 1994. Integrity checking in federated databases. Technical Report, Department of Computer Science, Stanford University.

Hammer, N., and McLeod, D. 1981. Database description with SDM: A semantic database model. *ACM Trans. Database Syst.* 6(3), 351–386.

Heimbigner, D., and McLeod, D. 1985. A federated architecture for information management. *ACM Trans. Office Inform. Systm.* 3(3), 46–69.

Hull, R. 1986. Relative information capacity of simple relational database schemata. *SIAM J. Comput.* 15(3), 865–886.

Kim, W. 1980. A new way to compute the product and join of relations. *Proc. ACM SIGMOD Int. Conf. Manage. Data* 179–187.

Kosky, A., Davidson, S., and Buneman, P. 1995. Semantics of database transformations. Technical Report MS-CIS-95-25, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA.

Litwin, W., Mark, L., and Roussopoulos, N. 1990. Interoperability of multiple autonomous databases. *ACM Comput. Surv.* 22(3), 267–293.

Markowitz, V., and Ritter, O. 1995. Characterizing heterogeneous molecular biology database systems. *J. Comput. Biol.* 2(4).

Markowitz, V., and Shoshani, A. 1992. Representing extended entity-relationship structures in relational databases: A modular approach. *ACM Trans. Database Syst.* 17(3), 423–464.

Meeting on Interconnection of Molecular Biology Databases. 1994. Abstract of participants. Available at url http://www.ai.sri.com/~pkarp/mimbd.html.

Meeting on Interconnection of Molecular Biology Databases. 1995. Abstract of participants. Available at url http://www.ai.sri.com/~pkarp/mimbd/95/abstracts.html.

Miller, R. J., Ioannidis, Y. E., and Ramakrishnan, R. 1994. Schema equivalence in heterogeneous systems: Bridging theory and practice. *Inform. Syst.* 19.

Motro, A. 1987. Superviews: Virtual integration of multiple databases. *IEEE Trans. Software Eng.* SE-13(7), 785–798.

Nakayama, M., Kitsuregawa, M., and Takagi, M. 1988. Hash-partitioned join method using dynamic destaging strategy. *Proc. Conf. Very Large Databases* 468–478.

Navathe, S., Elmasri, R., and Larson, J. 1986. Integrating user views in database design. *IEEE Comput.* 19(1), 50–62.

Overbeek, R., and Taylor, R.C. 1995. GenoBase Database Gateway. Available at url http://specter.dcrt.nih.gov:8004.

Ritter, O., Kocab, P., Senger, M., Wolf, D., and Suhai, S. 1994. Prototype implementation of the integrated genomic database. *Comput. Biomed. Res.* 27, 97–115.

Rusinkiewicz, M., Sheth, A., and Karabatis, G. 1991. Specifying interdatabase dependencies in a multidatabase environment. *IEEE Comput.* 24(12), 46–53.

Saltor, F., Castellanos, M., and Garcia-Solaco, M. 1991. Suitability of data models as canonical models for federated databases. *SIGMOD Record* 20(4), 44–48.

Sheth, A., and Larson, J. 1990. Federated database systems for managing distributed heterogeneous and autonomous databases. *ACM Comput. Surv.* 22(3), 183–236.

Shoval, P., and Zohn, S. 1991. Binary-relationship integration methodology. *Data Knowledge Eng.* 6, 225–249.

Ullman, J. D. 1989. *Principles of Database and Knowledgebase Systems I.* Computer Science Press, Rockville, MD.

Wiederhold, G., and Qian, X. 1987. Modeling asynchrony in distributed databases. *Proc. 1987 Int. Conf. Data Eng.* 246–250.

Zhuge, Y., Garcia-Molina, H., Hammer, J., and J. Widom (1995). View maintenance in a warehousing environment. *Proc. ACM SIGMOD Conf.* 316–327.

Address reprint requests to:
*S.B. Davidson*
*Department of Computer and Information Science*
*University of Pennsylvania*
*Philadelphia, PA 19104*

*susan@cis.upenn.edu*