Genomics Algebra: A New, Integrating Data Model, Language, and Tool for Processing and Querying Genomic Information

Joachim Hammer and Markus Schneider

Department of Computer & Information Science & Engineering
University of Florida
Gainesville, FL 32611-6120, U.S.A.
{jhammer,mschneid}@cise.ufl.edu

Abstract

The dramatic increase of mostly semi-structured genomic data, their heterogeneity and high variety, and the increasing complexity of biological applications and methods mean that many and very important challenges in biology are now challenges in computing and here especially in databases. In contrast to the many query-driven approaches advocated in the literature, we propose a new integrating approach that is based on two fundamental pillars. The *Genomics Algebra* provides an *extensible* set of high-level *genomic data types* (*GDT*s) (e.g., genome, gene, chromosome, protein, nucleotide) together with a comprehensive collection of appropriate *genomic functions* (e.g., translate, transcribe, decode). The *Unifying Database* allows us to manage the semi-structured contents of publicly available genomic repositories and to transfer these data into GDT values. These values then serve as arguments of Genomics Algebra operations, which can be embedded into a DBMS query language.

1. Introduction

In the past decade, the rapid progress of genome projects has led to a revolution in the life sciences causing a large and exponentially increasing accumulation of information in molecular biology and an emergence of new and challenging applications. The flood of genomic data, their high variety and heterogeneity, their semi-structured nature as well as the increasing complexity of biological applications and methods mean that many and very important challenges in biology are now challenges in computing and here especially in databases. This statement is underpinned by the fact that millions of nucleic acid sequences with billions of bases have been deposited in the well-known persistent *genomic repositories* EMBL, GenBank, and DDBJ. Both SwissProt and PIR form the basis of annotated protein sequence repositories together with TrEMBL and GenPept, which contain computer-translated sequence entries from EMBL and GenBank. In addition, hundreds of specialized repositories have been derived from the above primary sequence repositories. Information from them can only be retrieved by computational means.

The indispensable and inherently integrative discipline of *bioinformatics* has established itself as the application of computing and mathematics to the management, analysis, and understanding of the rapidly expanding amount of biological information to solve biological questions. Consequently, research projects in this area must have and indeed have a highly interdisciplinary character. Biologists provide their expertise in the different genomic application areas and serve as domain experts for input and validation. Computer scientists contribute their knowledge about the management of huge data volumes and about sophisticated data structures and algorithms. Mathematicians provide specialized analysis methods based, e.g., on statistical concepts.

We have deliberately avoided the term ‘genomic database’ and replaced it by the term ‘genomic repository’ since many of the so-called genomic ‘databases’ are simply collections of flat files or accumulations of Web pages and do not have the beneficial features of *real* databases in the computer science sense. Attempts to combine these heterogeneous and largely semi-structured repositories have been predominantly based on federated or query-driven approaches leading to complex middleware tiers between the end user application and the genomic repositories.

This position paper propagates the increased and integrative employment of current database technology as

Proceedings of the 2003 CIDR Conference

well as appropriate innovations for the treatment of non-standard data to cope with the large amounts of genomic data. In a sense, we advocate a "back to the roots" strategy of database technology for bioinformatics. This means that general database functionality should remain inside the DBMS and not be shifted into the middleware.

The concepts presented in this paper aim at overcoming the following fundamental challenges: The deliberate independence, heterogeneity, and limited interoperability among multiple genomic repositories, the enforced low-level treatment of biological data imposed by the genomic repositories, the lack of expressiveness and limited functionality of current query languages and proprietary user interfaces, the different formats and the lack of structure of biological data representations, and the inability to incorporate owns own, self-generated data.

Our integrating approach, which to our knowledge is new in bioinformatics and differs substantially from the integration approaches that can be found in the literature (see Section 3), rests on two fundamental pillars:

1. *Genomics Algebra*. This *extensible* algebra is based on the conceptual design, implementation, and database integration of a new, formal data model, query language, and software tool for representing, storing, retrieving, querying, and manipulating genomic information. It provides a set of high-level *genomic data types* (*GDT*s) (e.g., genome, gene, chromosome, protein, nucleotide) together with a comprehensive collection of appropriate *genomic operations* or *functions* (e.g., translate, transcribe, decode). Thus, it can be considered a resource for biological computation.
2. *Unifying Database*. Based on latest database technology, the construction of a unifying and integrating database allows us to manage the semi-structured or, in the best case, structured contents of genomic repositories and to transfer these data into high-level, structured, and object-based GDT values. These values then serve as arguments of Genomics Algebra operations. In its most advanced extension, the Unifying Database will develop into a global database comprising the most important or, as a currently rather unrealistic vision, even all publicly available genomic repositories.

The main benefits resulting from this approach for the biologist can be summarized as follows: Instead of a currently low-level treatment of data in genomic repositories, the biologist can now express a problem and obtain query results in *biological terms* (using *high-level* concepts) with the aid of genomic data types and operations. In addition, the biologist is provided with a powerful, general, and extensible high-level *biological query language* and *user interface* adapted to his/her needs. In a long-term view, the biologist is not confronted any more with hundreds of different genomic repositories but is equipped with an integrated and consistent working environment and user interface based on a unifying or ultimately global database. This Unifying Database allows the biologist to combine, integrate, and process data from originally different genomic resources in an easy and elegant way. Finally, it enables the integration and processing of self-generated data and their combination with data stemming from public resources.

2. Requirements of genomic data management

Adequate employment of database technology requires a deep understanding of the problems of the application domain. These domain-specific requirements then make it possible to derive computer science and database relevant requirements for which solutions have to be found. Discussions and cooperation with biologists have revealed the following main problems regarding the information available in the genomic repositories:

B1. Proliferation of specialized databases coupled with continuous expansion of established repositories creates missed opportunities.

B2. Two or more databases may hold additive or conflicting information.

B3. There is little or no agreement on terminology and concepts among different groups and consequently among the repositories they are building.

B4. A familiar data resource will disappear or morph to a different site.

B5. Query results are unmanageable unless organized into a customized, project-specific database.

B6. Data records copied from a source become obsolete and possibly misleading unless updated.

B7. The portal to each data site is a unique interface forcing scientists to develop customized access and retrieval methods.

B8. The database schema and data types are unknown to the user making custom SQL queries impossible.

B9. Biologists do not understand SQL, don't want to understand database schemas, and would prefer constructing queries using familiar biological terms and operations.

B10. Data in most genomics repositories are noisy, e.g., it is estimated that 30-60% of sequences in GenBank are erroneous.

These ten information-related problems (B1-B10) identified from a biologist's perspective lead to the following computer science centric problems (C1-C15). The identifiers in parentheses serve as cross-references into the list above.

C1. *Multitude and heterogeneity of available genomic repositories* (B1, B2, B3). Finding all appropriate sites from the more than 300 genomic repositories available on the Internet for answering a question is difficult. Many repositories contain related genomic

data but differ with respect to contents, detail, completeness, data format, and functionality.

C2. *Missing standards for genomic data representation* (B1, B2, B3, B7). There is no commonly accepted way for representing genomic data as evident in the large number of different formats and representations in use today.

C3. *Multitude of user interfaces* (B7). The multitude of genomic repositories implies a multitude of user interfaces and ontologies a biologist is forced to learn and to comprehend.

C4. *Quality of user interfaces* (B5, B7, B8, B9). In order to utilize existing user interfaces effectively, the biologist needs detailed knowledge about computing and data management since they are often too system-oriented and not user-friendly enough.

C5. *Quality of query languages* (B5, B8, B9). SQL is tailored to answer questions about alphanumerical data but unsuited for biologists asking biological questions. Consequently, the biologist should have access to a *biological query language*.

C6. *Limited functionality of genomic repositories* (B2, B3, B8, B9). The interactions of the biologist with a genomic repository are limited to the functions available in the user interface of that repository. This implies a lack of flexibility and the ability to ask new types of queries.

C7. *Format of query results* (B5, B6). The result of a query against a genomic repository is often outputted to the computer screen or to a text file and cannot be used for further computation. It is then left to the biologist to analyze the results manually.

C8. *Incorrectness due to inconsistent and incompatible data* (B1, B2, B3, B6). The existence of different genomic repositories with respect to the same kind of biological data leads to the question whether and where similar or overlapping repositories agree and disagree with one another.

C9. *Uncertainty of data* (B2, B6, B10). A very important but extremely difficult question refers to the correctness of data stored in genomic repositories. Due to vague or even lacking biological knowledge and due to experimental errors, erroneous data in genomic repositories cannot be excluded. Frequently, it cannot be decided from two inconsistent pieces of data, which one is correct and which one is wrong. In this case, access to both alternatives should be given.

C10. *Combination of data from different genomic repositories* (B2, B8, B9). Currently, data sets from different, independent genomic repositories cannot be combined or merged in an easy and meaningful manner.

C11. *Extraction of hidden and creation of new knowledge* (B1, B2, B8, B9). The nature of stored genomic data, e.g., in flat files, semi-structured records, makes it difficult to extract hidden information and to create new knowledge. The extraction of relevant data from query results and their analysis has to be performed without much computational support.

C12. *Low-level treatment of data* (B1, B2, B5, B8, B9). Genomic data representations and query results are more or less collections of textual strings and numerical values and are not expressed in biological terms such as genes, proteins, and nucleotide sequences. Operations on these high-level entities do not exist.

C13. *Integration of self-generated data and extensibility* (B5, B6). A biologist generates new biological data from their own research or experimental work. It is not possible to store and retrieve this data, to perform computations with generally known or even own methods, and to match the data against the genomic databases. This requires an extensible database management system, query language, and user interface.

C14. *Integration of new specialty evaluation functions* (B5, B8, B9). The possibility to evaluate data originally stemming from genomic repositories as well as self-generated data with publicly available methods is insufficient. Thus, it must be possible to create, use, and integrate user-defined functions that are capable of operating on both kinds of data.

C15. *Loss of existing repositories* (B4). Due to the high competition at the bioinformatics market, many companies disappear and with them the genomic repositories that were maintained by them. The company's valuable knowledge should be preserved.

This detailed problem analysis shows the enormous complexity of the information-related challenges biologists and computer scientists are confronted with. It is our conviction, and we will motivate and explain this in the following sections, that the combination of Genomics Algebra and Unifying Database is a solution for *all* these problems, even though it raises a number of new, complicated, and hence challenging issues.

3. Biological database integration

Much research has been conducted to reduce the burden on the biologist when attempting to access related data from multiple genomic repositories. One can distinguish two commonly accepted approaches: (1) *query-driven integration* or *mediation* and (2) *data warehousing*. In both approaches, users access the underlying sources indirectly through an integrated, global schema (view), which has been constructed either from the local schemas of the sources or from general knowledge of the domain.

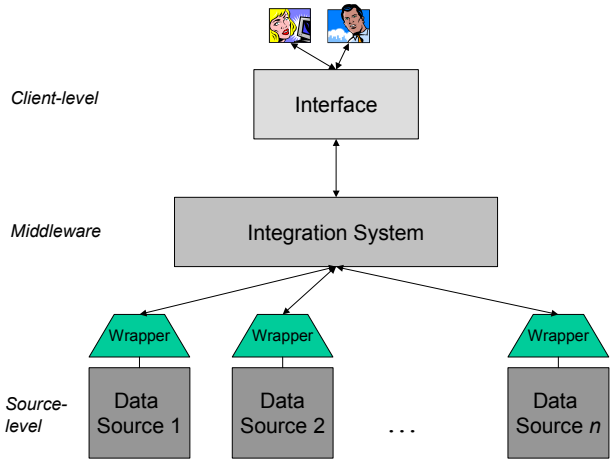

Figure 1: Generic integration architecture using the query-driven approach.

In the biological domain, most integration systems are currently based on the query-driven approach[1]. SRS [4], BioNavigator [2], K2/Kleisli [3], TAMBIS [8] and DiscoveryLink [6] are representatives of this class. Although they differ greatly in the capabilities they offer, they can be considered *middleware systems*, in which the bulk of the query and result processing takes place in a different location from where the data is stored. For example, K2/Kleisli, DiscoveryLink, and TAMBIS use source-specific data drivers (wrappers) for extracting the data from underlying data sources (e.g., GenBank, dbEST, SWISS-PROT) including application programs (e.g., the BLAST family of similarity search programs[1]). The extracted data is then shipped to the integration system, where it is represented and processed using the data model and query language of the integration system (e.g., the relational model and SQL in TAMBIS, the object-oriented model and OQL in K2/Kleisli). Biologists access the integration system through a client interface, which hides many of the source-specific details and heterogeneities. The query-driven approach to accessing multiple sources is depicted in Figure 1.

The generic data warehousing architecture (not shown) looks similar to the one depicted in Figure 1, except for the addition of a repository (warehouse) in the middleware layer. This warehouse is used by the integration system to store (materialize) the integrated views over the underlying data sources. Instead of answering queries at the source, the data in the warehouse is used. This greatly improves performance but requires complex maintenance procedures to update the warehouse in light of changes to the sources. Among the integration systems for biological databases, the only representative of the data warehousing approach known to us is GUS (Genomics Unified Schema) [3]. GUS describes a relational data warehouse in support of organism and tissue-specific research projects at the University of Pennsylvania. GUS shares some of its goals and requirements with the system proposed in this paper.

Despite the continuous advancements in biological database systems research, we argue that current systems present biologists with only an incomplete solution to the growing data management problem they are facing. More importantly, we share the belief of the authors in [3] that in those situations where close control over query performance as well as accuracy and consistency of the results are important (problem C8 in Section 2), the query-driven approach is *not* an option. However, query performance and correctness are only two aspects of biological data management. As can be seen in our list of requirements in Section 2, a suitable representation of biological data and a powerful and extensible biological query language capable of dealing with the inherent uncertainty of the correctness of biological data are equally important (C2, C4, C6, C9, C12, C14). To our knowledge, *none* of the existing systems currently addresses these requirements.

Independent of the integration approach used, current systems lack adequate support for biologists, forcing them to adopt their research methods to fit those imposed by the data management tools instead of the other way around. Table 1 summarizes how the integration systems mentioned above address each of the computer science issues C1-C15. In the next sections, we outline our proposal for a *genomic data warehouse* and a powerful analysis component. We believe the combination of the two greatly enhances the way biologists analyse and process information including data stored in the existing genomic repositories.

4. Genomics Algebra

Based on the observations and conclusions made in Section 3, we pursue an alternative, integrative approach, which heavily focuses on current database and data warehouse technologies. The *Genomics Algebra* (*GenAlg*) is the first of two pillars of our approach. It incorporates a sophisticated, self-contained, and high-level type system for genomic data together with a comprehensive set of operations.

4.1 An ontology for molecular biology and bioinformatics

The first step and precondition for a successful construction of our Genomics Algebra is the design of an *ontology* for molecular biology and bioinformatics. By ontology, we are referring to "a *specification of a conceptualization*." That is, in general, an ontology is a description of the concepts and relationships that define an application domain.

Applied to bioinformatics, an ontology is a "controlled vocabulary for the description of the

[1] Historically, sharing architectures based on the query-driven approach have also been termed *federated databases.*

molecular functions, biological processes and cellular components of gene products." An obstacle to its unique definition is that the multitude of heterogeneous and autonomous genomic repositories has induced terminological differences (synonyms, aliases, formulae), syntactic differences (file structure, separators, spelling) and semantic differences (intra- and interdisciplinary homonyms). The consequence is that data integration is impeded by different meanings of identically named categories, overlapping meanings of different categories, and conflicting meanings of different categories. Naming conventions of data objects, object identifier codes, and record labels differ between databases and do not follow a unified scheme. Even the meaning of important high-level concepts (e.g., the notion of *gene* or *protein function*) that are fundamental to molecular biology is ambiguous.

Table 1: Analysis of data management capabilities of existing integration systems with respect to the requirements outlined in Sec. 2.

	SRS	***BioNavigator***	***K2/Kleisli***	***DiscoveryLink***	***TAMBIS***	***GUS***
C1	User shielded from source details	User shielded from source details	User shielded from source details	User shielded from source details	User shielded from source details	User shielded from source details
C2	HTML	HTML	Global schema using object-oriented model	Global schema using relational model	Global schema using description logic	GUS schema based on relational model; OO views
C3	Single-access point	Single-access point	Single-access point	Single-access point	Single-access point	Single-access point
C4	Simple to use visual interface	Simple to use visual interface	Not a user-level interface	Requires knowledge of SQL	Simple to use visual interface	Requires knowledge of SQL
C5	Limited query capability	Not query oriented	Comprehensive query capability	Comprehensive query capability	Comprehensive query capability	Comprehensive query capability
C6	No new operations	No new operations	New operations on integrated view data	New operations on integrated view data	New operations on integrated view data	New operations defined on warehouse data
C7	No re-organization of source data	No re-organization of source data	Reorganization of result possible	Reorganization of result possible	Reorganization of result possible	Reorganization of result possible
C8	No reconciliation of results	No reconciliation of results	No reconciliation of results	No reconciliation of results	Result reconciliation supported	Data in warehouse is reconciled and cleansed
C9	No provision for dealing with uncertainty in data	No provision for dealing with uncertainty in data	No provision for dealing with uncertainty in data	No provision for dealing with uncertainty in data	No provision for dealing with uncertainty in data	No provision for dealing with uncertainty in data
C10	Results not integrated; sources must be Web-enabled	Results not integrated; sources must be Web-enabled	Results integrated using global schema; source wrapper needed	Results integrated using global schema; source wrapper needed	Results integrated using global schema; source wrapper needed	Query results are integrated
C11	Not supported	Not supported	Not supported	Not supported	Not supported	Annotations supported
C12	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported
C13	Not supported	Not supported	Not supported	Not supported	Not supported	Supported
C14	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported
C15	No archival functionality	No archival functionality	No archival functionality	No archival functionality	No archival functionality	Archiving of data supported

If the user queries a database with such an ambiguous term, until now (s)he has full responsibility to verify the semantic congruence between what (s)he asked for and what the database returned. An ontology helps here to establish a standardised, formally and coherently defined nomenclature in molecular biology. Each technical term has to be associated with a unique semantics that should be accepted by the biological community. If this is not possible, because different meanings or interpretations are attached to the same term but in different biological contexts, then the only solution is to coin a new, appropriate, and unique term for each context. Uniqueness of a term is an essential requirement to be able to map concepts into the Genomics Algebra.

Consequently, one of our current research efforts and challenges is to develop a comprehensive ontology, which defines the terminology, data objects and operations including their semantics that underlie genome sequencing. Since there has been much effort in defining ontologies for various bioinformatics projects [7], for example, Eccocyc, Pasta, Gene Ontology Consortium, we are about to study and compare these and other existing contributions in this field when defining our ontology. Therefore, besides an important contribution in itself, a comprehensive ontology forms the starting point for the development of our Genomics Algebra. In total, this goal especially contributes to a solution of the problems C1, C2, C3, C5, C8, C9, C11, and C12. Besides developing such a genomic ontology, a challenge is to devise an appropriate formalism for its unique specification.

4.2 The algebra

In a sense, the *Genomics Algebra* as the second step is the derived, formal, and executable instantiation of the

resulting genomic ontology. Entity types and functions in the ontology are represented directly using the appropriate data types and operations supported by our Genomics Algebra. This algebra[2] has to satisfy two main tasks. First, it has to serve as interface between biologists, who use this interface, and computer scientists, who implement it. An essential feature of the algebra is that it incorporates high-level biological terminology and concepts. Hence, it is not based on the low-level concepts provided by database technology. Second, as a result, this high-level, domain-specific algebra will greatly facilitate the interactions of biologists with genomic information stored in our Unifying Database (see Section 5) and incorporating at least the knowledge of the genome repositories. This is much like the invention of the 3-tier architecture and how the resulting data independence simplified database operations in relational databases. To our knowledge, no such algebra currently exists in the field of bioinformatics. The main impact of this goal is in solving the problems C2 to C4 and C6 to C14. This requires close coordination between domain experts from biology, who have to identify and select useful data types, relevant operations, and their semantics, and computer scientists, whose task it is to formalize and implement the algebra.

In order to explain the notion of algebra, we start with the concept of a *many-sorted signature*, which consists of two sets of symbols called *sorts* (or *types*) and *operators*. Operators are annotated with strings of sorts. For instance, the symbols *string*, *integer*, and *char* may be sorts and $concat_{string\ string\ string}$ and $getchar_{string\ integer\ char}$ two operators. The annotation with sorts defines the functionality of the operator, which in a more conventional way is usually written as *concat* : *string* × *string* → *string* and *getchar* : *string* × *integer* → *char*. To assign semantics to a signature, one must assign a (*carrier*) *set* to each sort and a *function* to each operator. Each function has domains and a codomain according to the string of sorts of the operator. Such a collection of sets and functions forms a *many-sorted algebra*. Hence, a signature describes the syntactic aspect of an algebra by associating with each sort the *name of a set* of the algebra and with each operator the *name of a function*. A signature especially defines a set of *terms* such as *getchar*(*concat*("Genomics", "Algebra"), 10). The *sort of a term* is the result sort of its outermost operator, which is *char* in our example.

Our *Genomics Algebra* is a domain-specific, many-sorted algebra incorporating a type system for biological data. Its sorts, operators, carrier sets, and functions will be derived from the genomic ontology developed in the first step. The sorts are called *genomic data types* (*GDTs*) and the operators *genomic operations*. To illustrate the concept, we assume the following, very simplified signature, which is part of our algebra:

sorts
gene, *primaryTranscript*, *mRNA*, *protein*

ops

transcribe:	*gene*	→ *primaryTranscript*
splice:	*primaryTranscript*	→ *mRNA*
translate:	*mRNA*	→ *protein*

This "mini algebra" contains four sorts or genomic data types for genes, primary transcript, messenger RNA, and protein as well as three operators *transcribe*, which for a given gene returns its primary transcript, *splice*, which for a given primary transcript identifies its messenger RNA, and *translate*, which for a given messenger RNA determines the corresponding protein. We can assume that these sorts and operators have been derived from our genomic ontology. Hence, the high-level nomenclature of our genomic ontology is directly reflected in our algebra. The algebra now allows us to (at least) syntactically combine different operations by (function) composition. For instance, given a gene *g*, we can syntactically construct the term *translate*(*splice*(*transcribe*(*g*))), which yields the protein determined by *g*. For the semantic problems of this term, see below.

Obviously, our mini algebra is incomplete. It is our conviction that finding a "complete" set of GDTs and genomic operations (what does "completeness" mean in this context?) is impossible, since new biological applications can induce new data types or new operations for already existing data types. Therefore, we pursue an *extensible* approach, i.e., if required, the Genomics Algebra can be extended by new sorts and operations. In particular, we can combine new sorts with sorts already present in the algebra, which leads to new operations. In other words, we can combine information stemming originally from different genomic repositories. Our hope is to be able to identify new, powerful, and fundamental genomic operations that nobody has considered so far.

From a software point of view, the Genomics Algebra is an extensible, self-contained software package and tool providing a collection of genomic data types and operations for biological computation. It is principally independent of a database system and can be used as a software library by a stand-alone application program. Thus, we also denote it as *kernel algebra*.

This kernel algebra develops its full expressiveness and usability if it is designed and integrated as a collection of *abstract data types* (ADTs) into the type system and query language of a database system (Section 6) [11]. ADTs encapsulate their implementation and thus hide it from the user or another software component like the DBMS. From a modelling perspective, the DBMS data model and the application-specific algebra or type system are *separated*. This enables the software developer

[2] The following algebraic model expresses our *object-based* understanding of the genomics domain. The realization of this model, e.g., using an object-oriented approach, is irrelevant in this context.

to focus on the application-specific aspects embedded in the algebra. Consequently, this procedure supports modularity and conceptual clarity and also permits the reusability of an algebra for different DBMS data models. It requires extensibility mechanisms at the type system level in particular and at all levels of the architecture of a DBMS in general, starting from user interface extensions down to new, external representation and index structures. From an implementation point of view, ADTs support modularity, information hiding, and the exchange of implementations. Simple and inefficient implementation parts can then be replaced by more sophisticated ones without changing the interface, that is, the signature of algebra operations.

4.3 Research challenges

We have already addressed two main research challenges, namely the design of the genomic ontology and the derivation of the signature of the Genomics Algebra from it. This leads us to the third main challenge, which is to give a formal definition of the genomic data types and operations, i.e., to specify their semantics, in terms that can be transferred to computer science and especially to database technology. A serious obstacle to the construction of the Genomics Algebra is the biologists' vague or even lacking knowledge about genomic processes. Biological results are inherently uncertain and never guaranteed (in contrast to the results of the application domains mentioned above) but always attached with some degree of uncertainty. For instance, it is known that the *splice* operation takes a primary Transcript and produces a messenger RNA, i.e., the effect of splicing (the "what"?) is clear since the cell demonstrates this observable biological function all the time. But it is unknown *how* the cell performs ("computes") this function. Transferred to our Genomics Algebra, this means that the signature of the *splice* operation is known with domain and codomain as shown in Section 4.2. We can even define the semantics of the operation in a denotational way. However, we cannot determine its operational semantics in the form of an algorithm and thus not implement it directly. A way out of this "dilemma" can be to map the procedure that biologists use in their everyday work to elude the problem or to compute an approximated solution for the problem. This inherent feature of uncertainty due to lacking knowledge must be appropriately reflected in the Genomics Algebra in order not to pretend correct results, which actually are vague or error-prone. The challenging issue is how this can be done in the best way.

The fourth main challenge is to implement the Genomics Algebra. This includes the design of sophisticated data structures for the genomic data types and efficient algorithms for the genomic operations. We discuss two important aspects here. A first aspect is that algorithms for different operations processing the same kind of data usually prefer different internal data representations in order to be as efficient as possible. In contrast to traditional work on algorithms, the focus is here not on finding the most efficient algorithm for each single problem (operation) together with a corresponding sophisticated data structure, but rather on considering the Genomics Algebra as a whole and on reconciling the various requirements posed by different algorithms within a single data structure for each genomic data type. Otherwise, the consequence would be enormous conversion costs between different data structures in main memory for the same data type. A second aspect is that the implementation is intended for use in a database system. Consequently, representations for genomic data types should not employ pointer data structures in main memory but be embedded into compact storage areas which can be efficiently transferred between main memory and disk. This avoids unnecessary and high costs for packing main memory data and unpacking external data.

5. Unifying Database

The Unifying Database is the second pillar of our integrating approach. By Unifying Database, we are referring to a data warehouse, which integrates data from multiple genomic repositories. We have chosen the data warehousing approach to take advantage of the many benefits it provides, including superior query processing performance in multi-source environments, the ability to maintain and annotate extracted source data after it has been cleansed, reconciled and corrected, and the option to preserve historical data from those repositories that do not archive their contents. Equally important, the Unifying Database is also the persistent storage manager for the Genomics Algebra.

5.1 Component overview

The component most visible to the user is the integrated schema. We distinguish between the portions of the schema that house the restructured and integrated external data (i.e., the entities that store the genomic data brought in from the sources) and which is publicly available to every user, and those that contain the user data (i.e. the entities that store user-created data including annotations), which may be private. The schema containing the external data is read-only to facilitate maintenance of the warehouse; user-owned entities are updateable by their owners. Separating between user and public space provides privacy but does not exclude sharing of data between users, which can be controlled via the standard database access control mechanism. Since all information is integrated in one database using the same formats and representation, cross-referencing, linking, and querying can be done using the declarative database language provided by the underlying database management system (DBMS), which has been extended by powerful

operations specific to the characteristics of the genomic data (see Section 6.3). However, users do not interact directly with the database language; instead, they use the commands provided by the Genomics Algebra, which may be embedded in a graphical user interface.

Conceptually, the Unifying Database may be implemented using any DBMS as long as it is extensible. By extensible we are referring to the capability to extend the type system and query language of the database with user-defined data types. For example, all of the object-relational and most object-based database management systems are extensible. We have more to say on the integration of the Genomics Algebra with the DBMS hosting the Unifying Database in Section 6. We believe our integration of the Genomics Algebra with the Unifying Database represents a dramatic improvement over current technologies (e.g., a query-driven integration system connected to BLAST sources) and will cause a fundamental change in the way biologists will conduct sequence analysis.

Conceptually, the component responsible for loading the Unifying Database and making sure its contents are up-to-date is referred to as *ETL (Extract-Transform-Load)*. In our system architecture, ETL comprises four separate activities:

1. Monitoring the data sources and detecting changes to their contents. This is done by the *source monitors*.
2. Extracting relevant new or changed data from the sources and restructuring the data into the corresponding types provided by the Genomics Algebra. This is done by the *sources wrappers*.
3. Merging related data items and removing inconsistencies before the data is loaded into the Unifying Database. This is done by the *warehouse integrator*.
4. Loading the cleaned and integrated data into the unifying database. This is done by the *loader*.

A conceptual overview of the Unifying Database is depicted in Figure 3 in Section 6. As we can see from the figure, the ETL component interfaces with a DBMS-specific adapter instead of the DBMS directly. This adapter, which implements the interface between database engine and Genomics Algebra, is the only component that has knowledge about the types and operations of the Genomics Algebra as well as how they are implemented and stored in the DBMS. The adapter is discussed in more detail in the next section.

Although much has been written and documented about building data warehouses for different applications including the GUS warehouse for biological data at the University of Pennsylvania [3], we briefly highlight the challenges that we face during the development of the Unifying Database.

5.2 Research challenges

We have identified the following challenges, which are directly related to implementing the components described above:

1. How do we best design the integrated schema so that it can accommodate data from a variety of genomic repositories?
2. How do we automate the detection of changes in the data sources?
3. How do we integrate related data from multiple sources in the Unifying Database?
4. How do we automate the maintenance of the Unifying Database?

Design of the integrated schema

There are two seemingly contradictory goals when designing the schema that defines the unifying database. On one hand, the schema should reflect the informational needs of the biologists, and should therefore be defined in terms of a global, biologist-centric view of the data (top-down design). On the other hand, the schema should also be capable of representing the union of the entities stored in the underlying sources (bottom-up design). We use a bottom-up approach by designing an integrated schema for the unifying database that contains the most important entities from all of the underlying repositories; which entities are considered important will be determined in discussions with the collaborating biologists. However, using a bottom-up approach does not imply a direct mapping between source objects and target schema. Given the wealth of data objects in the genomic repositories, a one-to-one mapping would result in a warehouse schema that is as unmanageable and inefficient as the source schemas it is trying to replace (e.g., GUS contains over 180 tables to store data from five repositories). Instead, we aim for a schema that combines and restructures the original data to obtain the best possible query performance while providing its users with an easy-to-use view of the data. If desired, each user can further customize the schema to his individual needs.

Schema design will likely be an iterative process, aiming to first create a schema that contains all of the nucleotide data, which will later be extended by new tables storing protein data, and so forth. This iterative process is possible since there is little overlap among the repositories containing different types of genomic data; furthermore, this type of schema evolution will mainly result in new entities being added instead of existing ones being removed or updated.

Change detection

The type of change detection algorithm used by the source monitor depends largely on the information source capability and the data representation. Figure 2 classifies the types of change detection for common sources and

data representations, where the abscissa denotes four different source types (explained below), and different data representations occur along the ordinate. A third dimension (*degree of cooperation* of the underlying source) is omitted for simplicity since source capability and degree of cooperation are related.

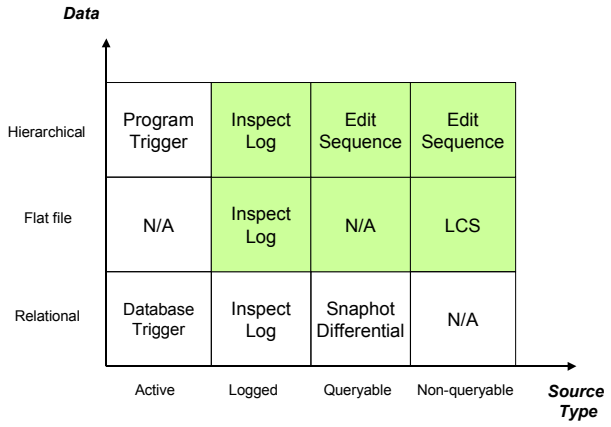

Figure 2. Classification of data sources using data representation and capability of the source management system as the defining characteristics. The grid identifies several proposed approaches to change detection. Shaded squares denote areas of particular interest to our project.

In Figure 2, *relational* refers to the familiar row/column representation used by the relational data model, *flat file* refers to any kind of unstructured information (e.g., text document), and *hierarchical* refers to a data representation that exhibits nesting of elements such as the tree and graph structures or data typically represented in an object model. Since most of the genomic data sources use either a flat file format (e.g., GenBank) or a hierarchical format (e.g., AceDB), we focus our investigation mainly on the upper two rows in the graph.

A*ctive data sources* provide active capabilities such that notifications of interesting changes can be programmed to occur automatically. Active capabilities are primarily found in relational systems (e.g., *database triggers*). However, some of the non-relational genomic data sources (e.g., SWISS-PROT) are now beginning to offer push capabilities, which will notify requesting users when relevant sequence entries have been made.

Where *logged sources* maintain a log that can be queried or inspected, changes can be extracted for hierarchical, flat file, or relational data. *Queryable sources* allow the database monitor to query information at the source, so periodic polling can be used to detect changes of interest. Two approaches are possible: detecting edit sequences for successive snapshots containing hierarchical data, or computing snapshot differentials for relational data.

Finally, *non-queryable sources* do not provide triggers, logs, or queries. Instead, periodic data dumps (snapshots) are provided off-line, and changes are detected by comparing successive snapshots. In the case of flat files, one can use the "longest common subsequence" approach, which is used in the UNIX diff command. For hierarchical data, various diff algorithms for ordered trees exist. In the case of the ACe databases, the "acediff" utility will compute minimal changes between different snapshots. For data sources that export snapshots in XML, IBMS's XMLTreeDiff can be used.

Despite the existence of prior existing work, change detection remains challenging, especially for the shaded regions in Figure 2. For example, in queryable sources, performance and semantic issues are associated with the polling frequency (PF). If the PF is too high, performance can degrade. Conversely, important changes may not be detected in a timely manner. A related problem independent of the change detection algorithm involves development of appropriate representations for deltas during transmission and in the warehouse. At the very least, each delta must be uniquely identifiable and contain (a) information about the data item to which it belongs and (b) the *a priori* and *a posteriori* data and the time stamp for when the update became effective.

Data integration

Before the extracted data from the sources can be loaded into the Unifying Database, one must establish the relationships among the data items in the source(s) with the existing data in the Unifying Database to ensure proper loading. In addition, in case data from more than one source is loaded, related data items from different sources must first be identified so that duplicates can be removed and inconsistencies among related values can be resolved. This last step is referred to as reconciliation.

The fundamental problem is as follows: How do we automatically detect relationships among similar entities, which are represented differently in terms of structure or terminology? This problem is commonly referred to as the *semantic heterogeneity* problem. Being able to find an efficient solution will allow us to answer the following important questions that arise during data integration:

- Which source object is related to which entity in the Unifying Database and how?
- Which data values can be merged, (for example because they contain complimentary or duplicate information)?
- Which items are inconsistent?

There has been a wealth of research on finding automated solutions to the semantic heterogeneity problem. For a general overview and theoretical perspective on managing semantic heterogeneities see [5].

Unifying database maintenance

Since the Unifying Database contains information from existing genomic repositories, its contents must be refreshed whenever the underlying sources are updated.

Ideally, the Unifying Database should always be perfectly synchronized with respect to the external sources. However, given the frequency of updates to most repositories, this is not realistic. On the other hand, there is evidence that a less synchronized warehouse is still useful. For example, SWISS-PROT, which is a curated version of the protein component of GenBank is updated on a quarterly basis; yet it is extensively used due to the high quality of its data. A similar experience has been documented by the authors in [3], whose GUS warehouse contents lag those of the data sources by a few months. As an important variation to existing refresh schemes, namely to automatically maintain certain pre-defined consistency levels between sources and warehouse, we plan to offer a manual refresh option. This allows the biologist to defer or advance updates depending on the situation.

Independent of the update frequency, refreshing the contents of the Unifying Database in an automatic and efficient manner remains a challenge. Since a warehouse can be regarded as an integrated "view" over the underlying sources, updating a warehouse has been documented in the database literature as the view maintenance problem. In general, one can always update the warehouse by reloading the entire contents, i.e., by re-executing the integration query(s) that produced the warehouse view. However, this is very expensive, so the problem is to find a new load procedure (or view query) that takes as input the updates that have occurred at the sources and possibly the original source instances or the existing warehouse contents and updates the warehouse to produce the new state. When the load procedure can be formulated without requiring the original source instances, the warehouse is said to be self-maintainable.

View maintenance has been studied extensively in the context of relational databases. However, fewer results are known in the context of object-oriented databases or semistructured databases. To our knowledge, no work has been done on recomputing annotations or corrections that need to be applied to existing data in the warehouse.

6. Interaction between genomics algebra and unifying database

Both pillars of our approach develop their full power if they are integrated into a common system architecture. In the following, we will discuss the system architecture, the requirements with respect to DBMSs, and appropriate integration mechanisms.

6.1 System architecture

A conceptual overview of the high-level architecture that integrates the *Genomics Algebra* with the *Unifying Database* is shown in Figure 3. The Unifying Database is managed by the DBMS and contains the genomic data, which comes either from the external sources or is user generated. The link between the Genomics Algebra and the Unifying Database is established through the *DBMS-specific adapter*. Extracting and integrating data from the external sources is the job of the *extract-transform-load* (ETL) tool shown on the right-hand side of Figure 3. User-friendly access to the functionality of the Genomics Algebra is provided by the *GUI* component depicted in the top center. In the following, we describe the remaining components of the architecture and some further aspects in more detail.

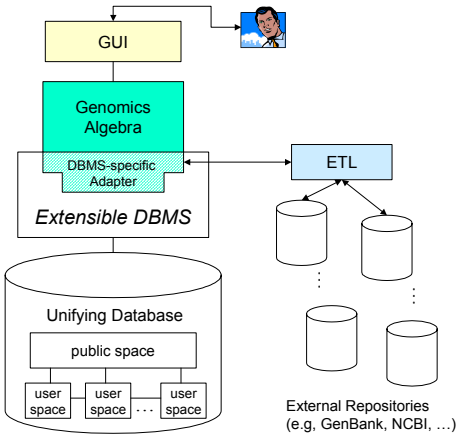

Figure 3: Integration of the Genomics Algebra with the Unifying Database through a DBMS-specific adapter.

6.2 Adapters and user-defined data types

Databases must be inherently *extensible* to be able to efficiently handle various rich, application-domain-specific complex data types. The *adapter* provides a DBMS-specific coupling mechanism between the ADTs together with their operations in the Genomics Algebra and the DBMS managing the Unifying Database. The ADTs are plugged into the adapter by using the *user-defined data type* (*UDT*) mechanism of the DBMS. UDTs provide the ability to efficiently define and use new data types in a database context without having to re-architect the DBMS. The adapter is registered with the database management system at which point the UDTs become add-ons to the type system of the underlying database.

Two kinds of UDTs can be distinguished, namely *object types*, whose structure is fully known to the DBMS, and *opaque types* whose structure is not. Object types can only be constructed by means of types the DBMS provides (e.g., native SQL data types, other object types, large objects, reference types). They turn out to be too limited. For our purpose, opaque types are much more appropriate. They allow us to create new fundamental data types in the database whose internal and mostly complex structure is unknown to the DBMS. The database provides storage for the type instances. *User-defined operators* (see also Section 6.3) that access the internal structure are linked as *external methods* or *external*

functions. They as well as the types are implemented, e.g., in C, C^{++}, or Java. The benefit of opaque types arises in cases where there is an external data model and behavior available like in the case of our Genomics Algebra.

All major database vendors support UDTs and external functions and provide mechanisms to package them up for easy installation (e.g., cartridges, extenders, datablades). A very important feature of the Genomics Algebra is that it is completely independent of the software that is used to provide persistence. That is, it can be integrated with *any* DBMS (relational, object-relational, object-oriented), as long as the DBMS is extensible. The reason is that the genomic data types are included into the database schema as *attribute data types* (like the standard data types real, integer, boolean, etc.). Tuples in a relational setting or objects in an object-oriented environment then only serve as containers for storing genomic values.

6.3 Integration of user-defined operations into SQL[3]

Typically, databases provide a set of pre-defined operators to operate on built-in data types. Operators can be related to arithmetic (e.g., +, -, *, /), comparison (e.g., =, <, >), Boolean logic (e.g., *not*, *and*, *or*), etc. From Section 6.2 we know that the UDT mechanism also allows us to specify and include user-defined operators as external functions. For example, it is possible to define a *resembles* operator for comparing nucleotide sequences.

User-defined operators can be invoked anywhere built-in operators can be used, i.e., wherever expressions may occur. In particular, this means that they can be included in SQL statements. They can be used in the argument list of a `SELECT` clause, the condition of a `WHERE` clause, the `GROUP BY` clause, and the `ORDER BY` clause. This ability allows us to integrate all the powerful operations and predicates of the Genomics Algebra into the DBMS query language, which, by the way, need not necessarily be SQL, and to extend the semantics of the query language in a domain-specific way. Let us assume the very simplified example of a predicate *contains* which takes as input a decoded DNA fragment and a particular sequence and which returns *true* if the fragment contains the specified sequence. Then we can write an SQL query as

```
SELECT id
FROM DNAFragments
WHERE contains(fragment, "ATTGCCATA")
```

6.4 User and system interface

For the biologist the quality of the user interface plays an important role, because it represents the communication mechanism between him/her on the one side and the Genomics Algebra and the Unifying Database on the other side. Currently, the biologist has to cope with the multitude, heterogeneity, fixed functionality, and simplicity of the user interfaces provided by genomic repositories. Hence, uniformity, flexibility, extensibility, and the possibility of graphical visualization are important requirements of a well-designed user interface. Based on an analysis and comparison of the currently available and most relevant user interfaces, our goal is to construct such a graphical user interface (GUI) for our Genomics Algebra. The aforementioned GUIs suffer from (at least) two main problems. First, they do not provide database and query facilities, which is an essential drawback, and second, their formats are only either HTML, ASN.1 (a binary format for bioinformatics data), or graphical output.

Our GUI is to comprise the following main elements: (1) a *biological query language* combined with a *graphical output description language*, (2) a *visual language* for the graphical specification of queries, and (3) the development of an *XML application* as a standardized input/output facility for genomic data.

The extended SQL query language enriched by the operations of the Genomics Algebra (see Section 6.3) is not necessarily the appropriate end user query language for the biologist. Biologists frequently dislike SQL due to its complexity. For them SQL is solely a database query language but not apt as a *biological query language*. Thus, the issue is here to design such a biological query language based on the biologists' needs. A query formulated in this query language will then be mapped to the extended SQL of the Unifying Database. A general, connected question is how the query result should be displayed. To enable high flexibility of the graphical output, the idea is to devise a *graphical output description language* whose commands can be combined with expressions of the biological query language.

The textual formulation of a query is frequently troublesome and only possible for the computationally experienced biologist. A *visual language* can help to provide support for the graphical specification of a query. The graphical specification is then evaluated and translated into a textual SQL representation which itself is executed by the Unifying Database. The design of such a visual language and the translation process are here the challenging issues.

As far as system interfaces are concerned, a number of *XML applications* exist for genomic data (e.g., GEML [9], RiboML [10], phyloML [12]). Unfortunately, these are inappropriate for a representation of the high-level objects of the Genomics Algebra. Hence, we plan to design our own XML application, which we name *GenAlgXML*.

[3] The integration approach of our Genomics Algebra with a relational DBMS and SQL can also be accomplished using an object-oriented DBMS and query language, such as OQL, for example.

6.5 Genomic index structures and genomic data optimization

We briefly mention two important research topics which are currently not the focus of our considerations but which will become relevant in the future, since they enhance the performance of the Unifying Database and the Genomics Algebra. These topics relate to the construction of *genomic index structures* and to the design of *optimization techniques* for genomic data.

As we add the ability to store genomic data, a need arises for indexing these data by using domain-specific, i.e., genomic, indexing techniques. These should support, e.g., similarity or substructure search on nucleotide sequences, or 3D structural comparison of tertiary protein sequences. The DBMS must then offer a mechanism to integrate these user-defined index structures.

The development of optimisation techniques for non-standard data (e.g., spatial, spatio-temporal, fuzzy data) must currently be regarded as immature due to the complexity of these data. Nevertheless, optimisation rules for genomic data, information about the selectivity of genomic predicates, and cost estimation of access plans containing genomic operators would enormously increase the performance of query execution.

7. Vision

We believe our project will cause a fundamental change in the way biologists analyze genomic data. No longer will biologists be forced to interact with hundreds of independent data repositories each with their own interface. Instead, biologists will work with a unified database through a single user interface specifically designed for biologists. Our high-level Genomics Algebra allows biologists to pose questions using biological terms, not SQL statements. Managing user data will also become much simpler for biologists, since his/her data can also be stored in the Unifying Database and no longer will s/he have to prepare a custom database for each data collection. Biologists should, and indeed want to invest their time being biologists, not computer scientists.

From a computer science perspective, the main implications consist in obtaining extended knowledge about the design and implementation of new, sophisticated data structures and efficient algorithms in the non-standard application field of biology and bioinformatics. The Genomics Algebra comprising all these data structures and algorithms will be made publicly available so that other groups in the community can study, improve, and extend it.

From a database perspective, our project leverages and extends the benefits and possibilities of current database technology. In particular, we demonstrate the elegance and expressive power of modeling and integrating non-standard and extremely complex data by the concept of abstract data types into databases and query languages. In addition, our approach is independent of a specific underlying DBMS data model. That is, the Genomics Algebra can be embedded in a relational, object-relational, or object-oriented DBMS as long as it is equipped with the appropriate extensibility mechanisms. The separation of DBMS and application-specific type system demonstrates the generality and flexibility of our approach. For example, in the future it is possible to develop algebras for other application domains.

References

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, pp. 403-410, 1990.

[2] Bionavigator Inc., "BioNavigator," http://www.bionavigator.com.

[3] S. Davidson, J. Crabtree, B. Brunk, J. Schug, V. Tannen, C. Overton, and C. Stoeckert, "K2/Kleisli and GUS: Experiments in integrated access to genomic data sources," *IBM Systems Journal*, vol. 40, pp. 512-531, 2001.

[4] T. Etzold, A. Ulyanov, and P. Argos, "SRS: information retrieval system for molecular biology data banks," *Methods Enzymol*, vol. 266, pp. 114-128, 1996.

[5] R. Hull, "Managing Semantic Heterogeneity in Databases: A Theoretical Perspective," *Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Tucson, Arizona, 1997.

[6] IBM Corp., "DiscoveryLink," http://www.ibm.com/discoverylink.

[7] R. McEntire, P. Karp, N. Abernethy, D. Benton, G. Helt, M. DeJongh, R. Kent, A. Kosky, S. Lewis, D. Hodnett, E. Neumann, F. Olken, D. Pathak, P. Tarczy-Hornoch, L. Toldo, and T. Topaloglou, "An Evaluation of Ontology Exchange Languages for Bioinformatics," *2000 Conference on Intelligent Systems for Molecular Biology*, 2000.

[8] N. W. Paton, R. Stevens, P. G. Baker, C. A. Goble, S. Bechhofer, and A. Brass, "Query processing in the TAMBIS bioinformatics source integration system," *11th International Conference on Scientific and Statistical Databases*, 1999.

[9] Rosetta Biosoftware, "The Gene Expression Markup Language (GEML)," http://www.rosettabio.com/products/conductor/geml/default.htm.

[10] Stanford Medical Informatics, "RiboML," http://www.smi.stanford.edu/projects/helix/riboml/.

[11] M. Stonebraker, "Inclusion of New Types in Relational Data Base Systems," *2nd International Conference On Data Engineering*, 1986.

[12] Washington and Lee University, "phyloML," http://cs.wlu.edu/~roycet/phyloML/.