Technologies for integrating biological data

Limsoon Wong

Date received (in revised form): 18th September 2002

Limsoon Wong
is Deputy Executive Director of Laboratories for the Institute for Infocomm Research in Singapore. He has done interesting research in several areas: he is known for his theorems on the 'Bounded Degree Property' of Nested Relational Calculus and SQL and invented the 'Kleisli Query System', which was the first broad-scale data integration system that solved many of the so-called 'impossible' bioinformatics integration problems. His most recent interest is in the analysis of emerging patterns of gene expression profiles.

Keywords: *data integration, data model, query language, application programming interface, EnsEMBL, Kleisli*

Abstract

The process of building a new database relevant to some field of study in biomedicine involves transforming, integrating and cleansing multiple data sources, as well as adding new material and annotations. This paper reviews some of the requirements of a general solution to this data integration problem. Several representative technologies and approaches to data integration in biomedicine are surveyed. Then some interesting features that separate the more general data integration technologies from the more specialised ones are highlighted.

REQUIREMENTS OF A GENERAL INTEGRATION SYSTEM FOR BIOLOGICAL DATA

The process of building a new database relevant to some field of study in biomedicine involves transforming, integrating and cleansing multiple data sources, as well as adding new material and annotations. The next section reviews some representative technologies for data integration in biomedicine. Then some of the features that distinguish the more general data integration technologies from the more specialised ones are highlighted. Lastly, we compare the surveyed technologies and comment on selecting such technologies. For now, let us first discuss the requirements of a good data integration system in biomedicine.

In a dynamic heterogeneous environment such as that of bioinformatics, many different databases and software systems are used. A large proportion of these databases were designed and built by biologists. When these databases were first created, the amount of data was small and it was important that the database entries were human readable. Database entries were therefore often created as flat files. As new types of data were captured, new databases were created using a variety of flat file formats. We ended up with a large number of different databases in different formats, typically using non-standard query softwares, and only accessible to bioinformatics experts.[1]

These databases and systems often do not have anything that can be thought of as an explicit database schema, which is a formalised queryable catalogue of all the tables in the database, the attributes of each of these tables, and the meaning of and indices on each of these attributes. Further compounding the problem is that research biologists demand flexible access and queries in *ad hoc* combinations. Simple retrieval of data is not sufficient for modern bioinformatics. The challenge is how to manipulate the retrieved data derived from various databases and re-structure the data in such a way to investigate specific biomedical problems.[2]

As observed by Baker and Brass,[1] many existing biology data retrieval systems are not fully up to the demand of painless and flexible data integration. These systems rely on low-level direct manipulation by the user, where he or she uses a keyword to extract summary records, then clicks on each resulting record to view its contents or to perform operations. This works well for simple actions. However, as the number of actions or records increases, such direct manipulations rapidly become a repetitive drudgery. Also when the questions become more complex and involve many databases, assembly of the data needed is likely to

Limsoon Wong,
Institute for Infocomm Research
21 Heng Mui Keng Terrace,
Singapore 119613

Tel: +65 6874 2099
Fax: +65 6872 5743
E-mail: limsoon@lit.or.sg/
limsoon@izr.a-star.edu.sg

exceed the skill and patience of the biologist. Merely providing a library package that interfaces to a large number of databases and analysis softwares is also not useful if it requires long-winded and tedious programming to make use of and add to the package.

A point solution is a highly specialised system

The systems provided by bioinformaticians in answer to the challenge above can roughly be divided into 'point' and 'general' solutions. A point solution is a highly specialised system: the data sources to be considered are small and fixed; the biomedical research questions to be addressed are small and fixed; and the point solution is a specific software that provides the expected answers and nothing else.[2] Hence, there is little database design and consideration for extensibility nor for flexibility. In contrast, a general solution is not designed with a specific set of biomedical research questions or with a specific set of data sources in mind. It must be designed with extensibility and flexibility in mind. A general solution can serve as the platform upon which to shorten the time needed for constructing various point solutions, just as a relational database management system can serve as the platform upon which to build specific accounting systems.

A general solution must be designed with extensibility and flexibility in mind

A system that aims to be a general integration mechanism in the bioinformatics environment described earlier must satisfy at least the following four conditions, which were identified previously by Wong.[3]

- It must not count on the availability of schemas. It must be able to compile any query submitted based solely on the structure of that query. If it needs a schema before it can compile a query, then it would be hard to use for our purpose because biomedical databases often do not have usable schemas.

- It must have a data model that the external database and software systems can easily translate to, without doing a lot of type declarations. If it does not have such a data model, then there would be a significant impedance in moving external data into the system, in moving internal data into external databases, and in manipulating the data when they are brought into the system.

- It must shield existing queries from evolution of the external sources as much as possible. For example, an extra field appearing in an external database table must not necessitate the recompilation or rewriting of existing queries over that data source. The external data sources used by a bioinformatician are typically owned by different organisations who have an autonomous right to evolve their databases. It is therefore important for a general data integration solution to be robust when the data sources evolve.

- It must have a data exchange format that is straightforward to use, so that it does not demand too much programming effort or contortion to capture the variety of structures of output from external databases and softwares. The data exchange format is the standard by which the system exchange data with the external data sources. If it is not straightforward to use, then great effort would be needed for connecting the system to the external data sources.

Besides the ability to query, assemble and transform data from remote heterogeneous sources, it is also important to be able to conveniently warehouse the data locally. The reasons to create local warehouses are given below, some of which were identified previously by Davidson *et al.*:[4]

Warehousing increases efficiency

- It increases efficiency. It is clear that we do not want to be choked by the slowest external data source nor by communication latency in the execution of our queries, especially if we own a fast computer. Warehousing gives us as much efficiency as we can afford to pay for.

Warehousing increases availability

- It increases availability. It is clear that we do not want to be unable to run our queries at a time we wish because a needed external source is unavailable. Warehousing guarantees that the data we need for our queries are always available whenever we need them.

Warehousing reduces risk of unintended 'denial of service' attacks

- It reduces the risk of unintended 'denial of service' attacks on the original sources. Some data sources, such as the Entrez web site at the National Center for Bioinformatic Information, impose a strict limit on the number of times or the amount of data that we can access within a single day. If we exceed that limit, then we risk being banned from the site. Unfortunately, some of our queries may require very intensive access to data held in such sites. Warehousing protects us from this risk by rendering it unnecessary for us to access the remote site.

Warehousing allows more careful data cleansing that cannot be done on the fly

- It allows more careful data cleansing that cannot be done on the fly. It is widely acknowledged that many of the biomedical sources contain a large number of errors.[5] For example, Schoenbach *et al*.[6] reported that up to 30 per cent of the database records that they accessed when constructing their warehouse on swine major histocompatibility complexes contained errors. Some of these errors can be detected and corrected on the fly, but some cannot. It therefore makes sense that if our queries are sensitive to certain errors that cannot be detected or corrected on the fly, then we should warehouse the data after careful cleansing.

Creating warehouses leads to other requirements on a general data integration solution. Specifically the general data integration solution must provide for the construction of warehouses that have the following properties. (1) The warehouse should be efficient to query. (2) The warehouse should be easy to update. There are two aspects to this issue of ease of update. The first aspect is of making an individual change to the warehouse, such as modifying an existing record, deleting an existing record or adding a new record. This aspect is a fundamental characteristic of the data integration tools that are used for maintaining the warehouse. The second aspect is that of the number of such individual changes that need to be made to bring the warehouse up to date. The second aspect is more a consideration for the strategy for maintaining the warehouse and is dictated by the interval between updates to the warehouse and the amount of changes that the underlying data sources can accumulate during the interval. A data integration tool that offers greater ease on the first aspect obviously also allows a greater range of strategies on the second aspect. (3) Equally important in the biology arena is that the warehouse should model the data in a conceptually natural form. Although a relational database system is efficient for querying and is easy to update, its native data model of flat tables forces us to fragment our data unnaturally and unnecessarily in order to make them fit into the third normal form.[7] For example, a record in the popular SWISS-PROT database[8] would be fragmented into almost 30 tables in order to be stored in accordance to the third normal form. This unnatural fragmentation brings forth two problems. First, it increases the mental load of the programmer and the possibility of programming errors in answering a query, because (i) the implementer of a query at a later date may not be the same person who did the third normal form conversion and (ii) the implementer of a query may not be the biologist who asks the query. Secondly, it increases the cost of certain queries significantly. For example, if the query needs to reconstruct a large portion of a SWISS-PROT record, we would be required to perform 10–20 joins on the tables.

It is also important to realise that no single system is complete for all possible uses. A data integration system is rightly

focused on (1) reading data from multiple sources for integration, (2) simple database-style transformation of data to facilitate data being passed from one application to the next, and (3) writing data to warehouses. There are certain types of analysis and manipulations of data that a data integration system is not expected to perform but is merely expected to facilitate. These analyses and manipulations include bioinformatics-specific operations such as multiple alignment and visualisation-specific operations such as displaying data in a graphical user interface. These operations are best implemented either in a specialised scripting language designed for that purpose or in a full strength common programming language. In order to facilitate the programming of these operations, the general data integration system must provide a means for these scripting and programming languages to interface to it, via a language embedding or an application programming interface for these languages.

There are certain analysis and manipulations of data that a data integration system is not expected to perform but is merely expected to facilitate

Lastly, the semantics issue may also be important.[4] This issue concerns the equivalence and consistency between parts of records in different data sources, as well as the mappings between these parts. A data integration technology that understands which parts of two data sources have the same meanings and should be consistent with each other is desirable. However, it must be recognised that the same record in a database can sometimes be interpreted in different ways depending upon the purpose and requirement of the user. Consequently, this issue is sometimes considered as a part of building a specific application or integrated database, as opposed to as a part of the tools used for building that integrated database or application.

SOME DATA INTEGRATION SOLUTIONS

EnsMart has a good query builder interface

We survey here a few solutions to the data integration and warehousing problem in biomedicine. The surveyed solutions include EnsEMBL,[9] GenoMax, SRS,[10] DiscoveryLink,[11] OPM,[12] Kleisli[3] and XML.[13,14] These examples are chosen to span specialised point solutions to increasingly general solutions. For each of these systems, we provide an overview and a discussion of their strong and weak points.

EnsEMBL

EnsEMBL is an excellent example of a very successful integration of data and tools for the highly specific purpose of genome browsing

EnsEMBL is a software system jointly developed by the European Bioinformatics Institute and the Sanger Institute.[9] It provides easy access to eukaryotic genomic sequence data. It also performs automatic prediction of genes in these sequence data and assembles supporting annotations for these predictions. It is not so much an integration technology, but is an excellent example of a very successful integration of data and tools for the highly specific purpose of genome browsing. EnsEMBL organises raw sequence data from public databases into its internal database. It then assembles these sequences into their proper place in the genome. After that, it runs GenScan to predict the location of genes and applies various analysis programs to annotate these predicted genes. Finally, the results of the process described above are presented for public access. The main 'entry points' to these results on the EnsEMBL Genome Browser are by (i) searching by sequence similarity via the built-in BLAST component of the EnsEMBL Genome Browser; (ii) browsing from the chromosome level all the way down to the DNA sequence level; (iii) searching using special EnsEMBL identifiers; and (iv) free-text matching using annotation of databases linked to EnsEMBL, including OMIM, SWISS-PROT and InterPro. It can also dump its data into Excel spreadsheets for use by external data-mining softwares. Alternatively, the EnsMart data retrieval tool can also be used to access these results. EnsMart has a good query builder interface that allows a user to conveniently specify certain types of genomic regions and filters on these results. As a last resort, EnsEMBL

provides a Perl-based programmatic interface for the most flexible access to its stored results.

Its strengths lie in its highly tailored functionalities for genome browsing. Once the sequences are imported into the system, assembly and annotation are automatically performed, and the results are automatically prepared for browsing in a nice graphical user interface. Its weaknesses lie also in its highly tailored point solution nature. It is not possible to ask EnsEMBL to perform an *ad hoc* query in general, unless that particular type of query has been anticipated by the designer of the EnsEMBL and its associated access tools. For example, while it is possible to ask a query such as 'extract 500 bases flanking the translation initiation site of each confirmed gene in the database' using EnsMart, it does not seem possible to ask a query such as 'extract the first exon of each confirmed gene in the database' using EnsMart at this moment. For the latter query, the user can resort to accessing EnsEMBL and extracting the required information by Perl programming. EnsEMBL also does not have a flexible data model nor exchange format, other than the structure of its highly specialised internal database. Thus, it is not straightforward to add new kinds of data sources, and it is also not straightforward to output or export data from EnsEMBL other than in the fixed export formats.

The weaknesses mentioned above are viewed from the perspective of the requirements of a general data integration system. However, one has to remember that EnsEMBL is intended as a point solution for the specific purpose of genome browsing. Within the context of this specific purpose, EnsEMBL works much better than virtually any other alternative, as its design has anticipated the common queries a biologist may want to ask and makes it possible to ask them without requiring the help of a programmer. An added plus point is that EnsEMBL has no licence fee.

GenoMax is a good illustration of an amalgamation of a few point solutions

GenoMax

GenoMax is an enterprise-level integration of bioinformatics tools and data sources developed by InforMax.[15] It is a good illustration of an amalgamation of a few point solutions, including a sequence analysis module and a gene expression module, developed on top of a data warehouse of fixed design.[2] The warehouse is an ORACLE database designed to hold sequence data, gene expression data, 3D protein structures and protein–protein interaction information. Load routines are built in for standard data sources such as GenBank and SWISS-PROT. The specialised point-solution modules provide capabilities such as performing BLAST[16] and GenScan[17] runs on sequences and computing differentially expressed genes from microarray experiments. A special scripting language of limited expressive power is also supported for building analytical pipelines.

Its strengths are twofold. Firstly, each of GenoMax's component point-solution modules is a very well-designed application for a specific purpose. For example, its gene expression module provides self-organising map clustering, principal component analysis and so forth on microarray data via simple-to-use graphical user interfaces. Secondly, these components are integrated in a tight way via the specially designed data warehouse. Its weakness is its tight point-solution-like application integration. While GenoMax has a broader scope than EnsEMBL, it does cover fewer data types and products than products such as SRS, DiscoveryLink and Kleisli. For example, these latter systems can easily incorporate chemical assay data that are beyond the current data warehouse design of GenoMax. In addition, GenoMax's scripting language is not designed for large-scale database-style manipulations and hence this type of *ad hoc* query is not always straightforward or optimised in GenoMax. There are also difficulties in adding new kinds of data sources and

GenoMax's scripting language is not designed for large-scale database-style manipulations

analysis tools. For example, it is probably impossible to express in the GenoMax scripting language the 'rosetta stone' method for extracting protein interactions.[18]

SRS

SRS is sometimes considered to serve 'more of a user interface integration role than as a true data integration tool'

SRS[10] is marketed by LION Bioscience and is arguably the most widely used database query and navigation system for the Life Science community. It provides easy-to-use graphical user interface access to a broad range of scientific databases, including biological sequences, metabolic pathways and literature abstracts. SRS provides some functionalities to search across public, in-house and in-licensed databases. In order to add a new data source into SRS, this data source is generally required to be available as a flat file and a description of the schema or structure of the data source must be available as an Icarus script, which is the special built-in wrapper programming language of SRS. The notable exception to this flat file requirement on the data source is when the data source is a relational database. SRS then indexes this data source on various fields parsed and described by the Icarus script. A biologist then accesses the data by supplying some keywords and constraints on them in the SRS Query Language. Then all records matching those keywords and constraints are returned. The SRS Query Language is primarily a navigational language. This query language has limited data joining capabilities based on indexed fields and has limited data restructuring capabilities. The results are returned as a simple aggregation of records that matched the search constraints. In short, in terms of querying power, SRS is essentially an information retrieval system. It brings back records matching specified keywords and constraints. These records can contain embedded links that a user can follow individually to obtain deeper information. However, it does not offer much help in organising or transforming the retrieved results in a way that might be needed for setting up an analytical pipeline. There is also a browser-based interface for formulating SRS queries and viewing results. In fact, this interface of SRS is often used by biologists as a unified front end to independently access multiple data sources, rather than learning the idiosyncrasies of the original search interfaces of these data sources. For this reason, SRS is sometimes considered[2] to serve 'more of a user interface integration role rather than as a true data integration tool.'

In summary, SRS has two main strengths. First, it is very straightforward to add new data sources into the system, because of the use of the Icarus scripting language and the simplicity of flat file indexing. In fact, several hundred data sources have been incorporated into SRS to date. Secondly, it has a nice user interface that greatly simplifies query formulation, making the system usable by a biologist without the assistance of a programmer. In addition, SRS has an extension known as PRISMA that is designed for automating the process of maintaining a SRS warehouse. PRISMA integrates the tasks of monitoring remote data sources for new data sets, and downloading and indexing such data sets. On the other hand, SRS also has some weaknesses. Firstly, it is basically a retrieval system that simply returns entries in a simple aggregation. If the biologist wishes to perform further operations or transformations on the results, they must do that by hand or write a separate postprocessing program using some external scripting languages such as C or Perl, which is cumbersome. Secondly, its principally flat-file-based indexing mechanism rules out the use of certain remote data sources – in particular, those that are not relational databases – and does not provide for straightforward integration with dynamic analysis tools. However, this latter shortcoming is mitigated by the SCOUT suite of applications marketed by LION Bioscience that are specifically designed to interact with SRS. It should also be mentioned that SRS is free for academic users.

DiscoveryLink

DiscoveryLink[11] is an IBM product and, in principle, it goes one step beyond SRS as a general data integration system for biomedical data. The first thing that stands out – when DiscoveryLink is compared with SRS, EnsEMBL and GenoMax – is the presence of an explicit data model. This data model dictates the way a DiscoveryLink user views the underlying data, the way he or she views results, as well as the way he or she queries the data. The data model is the relational data model.[19] The relational data model is the *de facto* data model of most commercial database management systems, including IBM's DB2 database management system upon which DiscoveryLink is based. As a result, DiscoveryLink comes with a high-level query language, SQL, that is a standard feature of all such database management systems. This gives DiscoveryLink several advantages over SRS. Firstly, not only can a user easily express SQL queries that go across multiple data sources – which an SRS user is able to do – but they can also perform fruther manipulations on the results – which an SRS user is unable to do. Secondly, not only are the SQL queries more powerful and expressive than those of SRS, the SQL queries are also automatically optimised by DB2. The use of query optimisation allows a user to concentrate on getting the query right without worrying about getting it fast.

The first thing that stands out is the presence of an explicit data model

The use of query optimisation allows a user to concentrate on getting the query right

However, DiscoveryLink still has a some way to go in practice. The reason is twofold. The first reason is that DiscoverLink is tied to the relational data model. This implies every piece of data that it handles must be a table of atomic objects such as strings and numbers. Unfortunately, most of the data sources in biology are not that simple and are deeply nested. Therefore, there is severe impedance mismatch between these sources and DiscoveryLink. Consequently, it is not straightforward to add new data sources or analysis tools into the system. For example, to put the SWISS-PROT database into a relational database in the third normal form would require us to break every SWISS-PROT record into nearly 30 pieces in a normalisation process! Such a normalisation process requires a certain amount of skill. Similarly, to query the normalised data in DiscoveryLink requires some mental and performance overhead, as we need to figure out which part of SWISS-PROT has gone to which of the 30 pieces and we need to join some of the pieces back again. The second reason is the DiscoveryLink supports only wrappers written in C++, which is not the most suitable programming language for writing wrappers. In short, it is difficult to extend DiscoveryLink with new sources. In addition, DiscoveryLink does not store nested objects in a natural way and is limited in its capability for handling long documents. It also has limitations as a tool for creating and managing data warehouses for biology.

It is not straightforward to add new data sources into DiscoveryLink

In spite of these weaknesses, in theory, DiscoveryLink has greater generality than point solutions such as EnsEMBL, specialised application integration such as GenoMax, and user interface integration solutions such as SRS. Unfortunately, this greater generality is achieved at the price of requiring that SQL be used for expressing queries. While writing queries in SQL is generally simpler than writing in Perl, it is probably still beyond the skill of an average biologist. This is a disadvantage in comparison with EnsEMBL, GenoMax and SRS, which have good user interfaces for a biologist to build the simpler queries.

OPM

OPM[12] was developed at Lawrence-Berkeley National Labs and is a general data integration system. OPM was marketed by Gene Logic, but its sales were discontinued some time ago. It goes one step beyond DiscoveryLink in the sense that it has a more powerful data model, which is an enriched form of the entity-relationship data model.[20] This data model can deal with the deeply nested structure of biomedical data in a natural

way. Thus it removes the impedance mismatch. This data model is also supported by an SQL-like query language that allows data to be seen in terms of entities and relationships. Queries across multiple data sources, as well as transformation of results, can be easily and naturally expressed in this query language. Queries are also optimised. Furthermore, OPM comes with a number of data management tools that are useful for designing an integrated data warehouse on top of OPM.

OPM comes with a number of data management tools that are useful for designing an integrated data warehouse

However, OPM has several weaknesses. First, it requires the use of a global integrated schema. It needs significant skill and effort to design a global integrated schema well. If a new data source is to be added, the effort needed to re-design the global integrated schema potentially goes up quadratically with respect to the number of data sources already integrated. If an underlying source evolves, the global integrated schema tends to be affected and significant re-design effort is potentially needed. Therefore, it may be costly to extend OPM with new sources. Secondly, OPM stores entities and relationships internally using a relational database management system. It achieves this by automatically converting the entities and relationships into a set of relational tables in the third normal form. This conversion process leads to an entity being broken up into many pieces when stored. This process is transparent to the OPM user, so they can continue to think and query in terms of entities and relationships. Nevertheless, the underlying fragmentation often causes performance problems, as many queries that required no join – when viewed at the conceptual level of entities and relations – are mapped to queries that required many joins on the physical pieces that entities are broken into. Thirdly, OPM does not have a simple format to exchange data with external systems. At one stage, it interfaces to external sources using CORBA. The effort required for developing CORBA-compliance wrappers is generally significant.[21]

New wrappers are generally easy to develop and insert into the Kleisli system

Furthermore, CORBA is not designed for data intensive applications.

Although OPM's query language is at a higher level and is simpler to use than the SQL of DiscoveryLink, it shares the same disadvantage as DiscoveryLink from the perspective of an average biologist. The programming of queries other than the simplest kind is probably still beyond his or her expertise.

Kleisli

Kleisli[3] is marketed by geneticXchange Inc. of Menlo Park. It is one of the earliest systems that have been successfully applied to some of the earliest data integration problems in the human genome project, including the US Department of Energy's so-called 'impossible' queries in early 1994. The approach taken by the Kleisli system is illustrated by Figure 1. It is positioned as a mediator system encompassing a nested relational data model, a high-level query language, and a powerful query optimiser. It runs on top of a large number of light-weight wrappers for accessing various data sources. There are also a number of application programming interfaces (APIs) that allow Kleisli to be accessed in an open database connectivity (ODBC)- or Java database connectivity (JDBC)-like fashion in various programming languages for a various applications. The Kleisli system is highly extensible. It can be used to support several different high-level query languages by replacing its high-level query language module. Currently, Kleisli supports a 'comprehension syntax'-based language called CPL[3] and a 'nested relationalised' version of SQL called sSQL. The Kleisli system can also be used to support many different types of external data sources by adding new wrappers, which forward Kleisli's requests to these sources and translate their replies into Kleisli's exchange format. These wrappers are light weight and new wrappers are generally easy to develop and insert into the Kleisli system. The optimiser of the Kleisli system can also be customised by different rules and

Kleisli is positioned as a mediator system

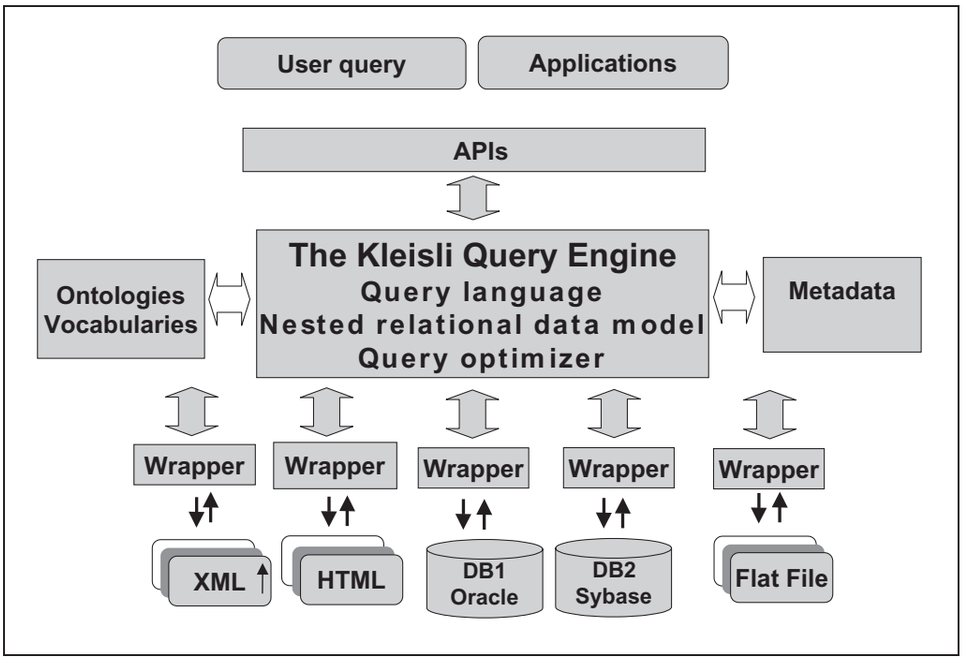

Figure 1: Kleisli, positioned as a mediator

Kleisli has the ability to store, update and manage complex nested data

Kleisli has the ability to turn many kinds of database systems into an updatable store conforming to its nested relational data model

strategies.[3] Besides the ability to query, assemble and transform data from remote heterogeneous sources, it is also important to be able to conveniently warehouse the data locally. Kleisli does not have its own native database management system. Instead, it has the ability to turn many kinds of database systems into an updatable store conforming to its nested relational data model. In particular, Kleisli can use flat relational database management systems such as Sybase, Oracle and MySQL to be its updatable store. It can even use all of these systems simultaneously. It is also worth noting that Kleisli stores nested relations into flat relational database management systems using an encoding scheme that does not require these nested relations to be fragmented over several tables.

Kleisli possesses the following strengths. It does not require data schemas to be available. It has a nested relational data model and a data exchange format that external databases and software systems can easily translate into. It shields existing queries, via a type inference mechanism, from certain kinds of structural changes in the external data sources. Kleisli also has the ability to store, update and manage complex nested data. It has a good query optimiser. Finally, Kleisli is also equipped with two application programming interfaces so that it can be accessed in a JDBC-like manner from Perl and Java.[22] However, Kleisli shares a common weakness with DiscoveryLink and OPM. Even though CPL and sSQL are both high-level query languages and protect the user from many low-level details – such as communication protocols, memory management, thread scheduling and so on – the programming of queries using CPL or sSQL other than the simplest kind is probably still beyond the expertise of an average biologist.

XML

XML (Extensible Markup Language) is a standard for formatting documents. As such, it is not a data integration system by itself. However, there is a growing suite of tools based on XML that, taken as a whole, can be used as a data integration system. We therefore believe it is pertinent to include a discussion on XML and its associated tools in the context of this paper.

A feature that separates the more general data integration technologies from the more specialised technologies is the explicit presence of a data model

XML allows for a hierarchical nesting of tags and the set of tags can be defined flexibly. Thus XML can be viewed as a powerful data model and a useful data exchange format, providing directly for two of the important ingredients of a general data integration solution for biomedicine. As a result, an increasing number of tools and sources in biomedicine such as PIR, Entrez and so on are becoming XML-compatible.[14] The intense interest in the development of query languages for semi-structured data[13] in the database community has also resulted in a number of powerful XML query languages such as XQL[23] and XQuery,[24] which provided the means for querying across multiple data sources and for transforming the results into a more suitable form for subsequent analysis steps. Research and development works are also in progress on XML query optimisation[25] and on XML data stores.[26] A robust and stable XML-based general data integrating and warehousing system does not yet exist for biomedicine. However, once high-performance XML data stores become available, we can also expect the database research community to begin more research and development on data warehousing using these stores. Consequently, we believe that given sufficient time, XML and the growing suite of XML-based tools can mature into an alternative data integration system in biomedicine that is comparable to Kleisli in generality and sophistication.

HIGHLIGHT OF SELECTED FEATURES

This section highlights some features that distinguish the more general data integration technologies from the more specialised data integration solutions surveyed earlier.

Data model and data exchange format

A key feature that separates the more general data integration technologies – DiscoveryLink, OPM, Kleisli – from the more specialised technologies – EnsEMBL, GenoMax, SRS – is the explicit presence of a data model. From the point of view of the traditional database world,[27] a data model provides the means for specifying particular data structures, for constraining the data associated with these structures, and for manipulating the data *within* a database system. In order to handle data outside the database system, this traditional concept of data model is extended to include a data exchange format, which is a means for bringing data outside the database system into it and also for bringing data inside the database system outside. We use Kleisli's data model to illustrate this concept.

The data model underlying the Kleisli system is a complex object type system that goes beyond the 'sets of records' or 'flat relations' type system of relational databases.[19] It allows arbitrarily nested records, sets and a few other data types.[3] Having such a 'nested relational' data model is useful and matches the structure of biomedical data sources well. For example, if we are restricted to the flat relational data model, the GenPept report in Example 1 must necessarily be split into many separate tables in order to be stored in a relational database without loss. The resulting multi-table representation of the GenPept report is conceptually unnatural and operationally inefficient.

Example 1 *The GenPept report is the format chosen by the US National Center for Biotechnology Information to present amino acid sequence information. The feature table is the part of the GenPept report that documents the positions and annotations of regions of special biological interest. The following type represents the feature table of a GenPept report from Entrez.*[28] *Here we use { and } brackets for sets, (and) brackets for records, [and] brackets for lists, and #l: to label the field l of a record. In fact, the same bracketing scheme is used as the data exchange format of Kleisli.*[22]

```
(#uid:num, #title:string, #accession:
string, #feature:{(
  #name: string, #start:num, #end:num,
  #anno:[(
```

```
    #anno_name: string,
    #descr:string)])})
```

The feature table of GenPept report 131470, a tyrosine phosphatase 1C sequence, is shown partially below.

```
(#uid:131470, #accession:"131470",
#title:"...(PTP-1C)...", #feature:{(
  #name:"source", #start:0, #end:594,
  #anno:[
    (#anno_name:"organism", #descr:"Mus
    musculus"),
    (#anno_name:"db_xref",
    #descr:"taxon:10090")]),
    ...})
```

The particular feature displayed above goes from amino acid 0 to amino acid 594, which is actually the entire sequence, and has two annotations. The first annotation indicates that this amino acid sequence is derived from mouse DNA sequence. The second is a cross-reference to the US National Center for Biotechnology Information taxonomy database. □

This type of output would definitely present a problem if we had to give it to a system based on the flat relational model

It is generally easy to develop a wrapper for a new data source, or modify an existing one, and insert it into Kleisli. The main reason is that there is no impedance mismatch between the data model supported by Kleisli and the data model that is necessary to capture the data source. The wrapper is often a very light-weight parser that simply parses records in the data source and prints them out in Kleisli's very simple data exchange format.

There is no impedance mismatch between the data model supported by Kleisli and the data model that is necessary to capture the data source

Example 2 *Suppose we want to implement a function* `webomim-get-detail` *that uses an OMIM identifier to access the OMIM database and returns a set of objects matching the identifier. Suppose the ouput is of type*

```
{(#uid: num, #title: string,
#gene_map_locus: {string},
  #alternative_titles: {string},
  #allelic_variants: {string})}
```

Note that is this a nested relation: it is a set of records, and each record has three fields that are also of set types, viz. `#gene_map_locus`, `alternative_titles` *and* `allelic_variants`. *This type of output would definitely present a problem if we had to give it to a system based on the flat relational model, as we would need to arrange for the information in these three fields to be sent into separate tables. Fortunately, such a nested structure can be mapped directly into Kleisli's exchange format. So the wrapper implementer would only need to parse each matching OMIM records and to write it out in a format like this:*

In the case of SRS, this implicit data model in Icarus also greatly facilitates the rapid scripting of wrappers

Another feature that separates the more general data integration technologies from the more specialised ones is the presence of a flexible high-level query language

```
{(#uid: 189965,
  #title: "CCAAT/ENHANCER-BINDING
  PROTEIN, BETA; CEBPB",
  #gene_map_locus: {"20q13.1"},
  #alternative_titles: {"C/EBP-BETA",
     "INTERLEUKIN 6-DEPENDENT DNA-
    BINDING PROTEIN; IL6DBP",
     "LIVER ACTIVATOR PROTEIN; LAP",
     "LIVER-ENRICHED TRANSCRIPTIONAL
    ACTIVATOR PROTEIN",
     "TRANSCRIPTION FACTOR 5; TCF5"},
  #allelic_variants: {})}
```

Here, instead of needing to create separate tables to keep the sets nested inside each record, the wrapper would simply print the appropriate set brackets { and } to enclose these sets. Kleisli would automatically deal with them as they were handed over by the wrapper. This kind of parsing and printing is extremely easy to implement. □

OPM shares with Kleisli a nested relational data model, except that the former lacks a data exchange format. Hence the mapping of the examples to OPM's data model is conceptually just as straightforward, but the practical implementation in OPM demands considerably more effort. It is worth pointing out that while SRS does not have an explicit data model, it does have an implicit one supported by its Icarus language for scripting parsers. In the case of SRS, this implicit data model in Icarus also greatly facilitates the rapid scripting of wrappers.

Query capability

Another feature that separates the more general data integration technologies from the more specialised ones is the presence of a flexible high-level query language for manipulating data conforming to the data model. We use sSQL, the primary query

language of Kleisli, to illustrate this feature. sSQL is based on the *de facto* commercial database query language SQL, except for extensions made to cater for the nested relational model and for the federated heterogeneous data sources.

Example 3 *The feature table of a GenBank report has the type below. The field* `#position` *of a feature entry is a list indicating the start and stop positions of that feature. If the feature entry is a CDS, this list corresponds to the list of exons of the CDS. The field* `#anno` *is a list of annotations associated with the feature entry.*

```
{(#uid: num, #title: string, #accession:
string, #seq: string, #feature: {(
  #name: string,
  #position: [(#start: num, #end: num,
  #negative: bool, ...)],
  #anno: [(#anno_name: string, #descr:
  string)], ...)}, ...)}
```

Given a set DB *of feature tables of GenBank chromosome sequences, we can extract the 500 bases upstream of the translation initiation sites of all disease genes – in the sense that these genes have a cross-reference to OMIM – on the positive strand in* DB *as below. Here* `l2s` *is a function that converts a list into a set:*

```
select
  uid: x.uid,
  protein: r.descr,
  flank: string-span(x.seq, p.start - 500,
  p.start)
from
  DB x, x.feature f,
  {f.position.list-head} p,
  f.anno.l2s a, f.anno.l2s r
where not (p.negative)
and a.descr like "MIM:%" and
a.anno_name = "db_xref"
and r.anno_name = "protein_id"
```

Similarly, we can extract the first exons of these same genes as follows:

```
select
  uid: x.uid,
  protein: r.descr,
  exon1: string-span(x.seq, p.start,
  p.end)
from
  DB x, x.feature f,
  {f.position.list-head} p,
  f.anno.l2s a, f.anno.l2s r
where not (p.negative)
and a.descr like "MIM:%" and
a.anno_name = "db_xref"
and r.anno_name = "protein_id"
```

These two example queries illustrate how a high-level query language makes it possible to extract very specific output in a relatively straightforward manner. □

We illustrate how to combine multiple sources using high-level query languages. An *in silico* discovery kit (ISDK) prescribes experimental steps carried out in computers very much like the experimental protocol carried out in wet laboratories for specific scientific investigation. From the perspective of Kleisli, an ISDK is just a script written in sSQL and performs a defined information integration task. It takes an input data set and parameters from the user, executes and integrates the necessary computational steps of database queries and applications of analysis programs or algorithms, and outputs a set of results for specific scientific inquiry.

Example 4 *The simple ISDK in Figure 2 demonstrates how to use an available ontology data source to get around the problem of inconsistent naming in genes and proteins, and to integrate information across multiple data sources. It is implemented in the sSQL script below. With the user input of a gene name* `G`*, the ISDK performs the following tasks: first, it retrieves a list of aliases for* `G` *from the Gene Nomenclature database provided by the Human Genome Organization (HUGO). Then it retrieves information for diseases associated with this particular protein in the Online Mendelian Inheritance of Man Database (OMIM), and finally it retrieves all relevant references from MEDLINE. Here,* `s21` *is a function that converts a set into a list;* `list-sum` *is a function to sum a list of numbers;* `ml-get-count-general` *is a function that accesses the MEDLINE database in Bethesda and computes the number of MEDLINE reports matching a given keyword;* `ml-get-abstract-by-uid` *is a function that accesses MEDLINE for a report given a unique identifier;* `webomim-`

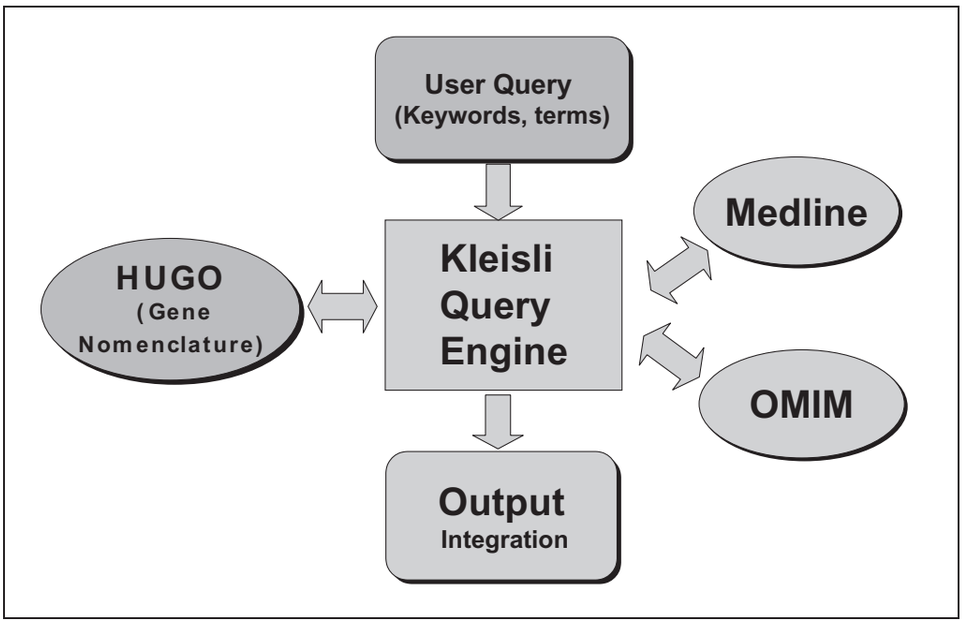

Figure 2: An *in silico* discovery kit that uses an available ontology data source to get around the problem of inconsistent naming in genes and proteins, and integrates information across multiple data sources

`get-id` *is a function that accesses the OMIM database in Bethesda to obtain unique identifiers of OMIM reports matching a keyword;* `webomim-get-detail` *is a function that accesses OMIM for a report given a unique identifier; and* `hugo-get-by-symbol` *is a function that accesses the HUGO database and returns HUGO reports matching a given gene name.*

```
create function get-info-by-genename (G)
as
Select
  hugo: w, omim: y, pmidl-abstract: z,
  num-medline-entries: list-sum(lselect
  ml-get-count-general (n) from
  x.Aliases.s21 n)
from
  hugo-get-by-symbol (G) w,
  webomim-get-id (searchtime:0,
  maxhits:0, searchfields:{},
  searchterms:G) x,
  webomim-get-detail (x.uid) y,
  ml-get-abstract-by-uid(w.PMID1) z
where
  x.title like ("%" ^ G ^ "%");
```

Such queries fulfil many of the requirements for efficient in silico *discovery processes: (1) their modular nature gives scientists the flexibility to select and combine specific queries for specific research projects; (2) they can be executed automatically by Kleisli in batch mode and can handle large data volume; (3) their scripts are reusable to perform repetitive tasks and can be shared among scientific collaborators; (4) they form a base set of templates that can be readily modified and refined to meet different specifications and to make new queries; and (5) new databases and new computational tools can be readily incorporated to existing scripts.* □

The flexibility and power shown in these sSQL examples can also be experienced in OPM, and to a lesser extent in DiscoveryLink. With good planning, a specialised data integration system can also achieve great flexibility and power within a narrower context. For example, the EnsMart tool of EnsEMBL is a very well-designed interface that helps a non-programmer build complex queries in a simple way. In fact, an equivalent query to the first sSQL query in Example 3 can also be specified using EnsMart with a few clicks of the mouse. Nevertheless, there are some unanticipated cases that cannot be

expressed, such as the second sSQL query in Example 3.

Application programming interfaces

The high-level query languages of the more general data integration systems surveyed are all SQL-like and are thus designed to express traditional (nested relational) database-style queries. Not every query in bioinformatics falls into this class. For these non-database-style queries, some other programming languages can sometimes be a more convenient or more efficient means of implementation. Therefore, it is useful to develop some application programming interfaces to these more general data integration systems for various popular programming languages.

It is useful to develop some application programming interfaces to these more general data integration systems

In the case of Kleisli, there is the Pizzkell suite[22] of interfaces to the Kleisli Exchange Format for various popular programming languages. Each of these interfaces in the Pizzkell suite is a library package for parsing data in Kleisli's exchange format into an internal object of the corresponding programming language. It also serves as a means for embedding the Kleisli system into that programming language, so that the full power of Kleisli is available within that programming language in a manner similar to that achieved by JDBC and ODBC for relational databases. The Pizzkell suite currently includes CPL2Perl and CPL2Java, for Perl and Java.

The presence of such application programming interfaces may be even more crucial for the more specialised integration solution

The presence of such application programming interfaces may be even more crucial for the more specialised integration solution. While a point solution such as EnsEMBL is typically designed with a specific aim in mind, it is not unusual to subsequently discover that a user wants to use the integrated data in an unanticipated way. In such a situation, it would be convenient if an application programming interface is available on the integrated data. For example, in the case of EnsEMBL, as EnsEMBL is implemented in Perl using BioPerl as the backbone, the same library of routines that have been accumulated in the course of implementing EnsEMBL would be the perfect application programming interface to EnsEMBL.

CONCLUDING REMARKS

Let us first summarise our opinion on how well each of the surveyed systems satisfies the requirements of a general data integration system for biomedicine. EnsEMBL and GenoMax are point solutions and thus naturally do not satisfy the requirements of a general data integration system well. SRS and DiscoveryLink were claimed by their inventors as general data integration systems for biomedicine. However, in reality, SRS is a form of user interface integration and hence it does not satisfy the requirements well. On the other hand, while DiscoveryLink has most of the components required, these components come in the wrong flavour – the adoption of the flat relational model causes it to be impotent in the biomedical data integration arena. OPM is a well-designed system for the purpose of biomedical data integration, except for (1) a problem in performance due to data fragmentation as it unwisely maps all data to the third normal form, (2) the lack of a simple data exchange format, and (3) the need of a global schema. XML and Kleisli have all the qualities required for good general data integration. However, compared with Kleisli, XML still needs more time to mature, especially in terms of query optimisation and data warehousing capabilities.

Let us next look at these surveyed systems from the perspective of an average biologist. While general data integration systems such as DiscoveryLink, OPM and Kleisli simplify the programming of *ad hoc* queries, it must also be acknowledged that the programming skills required are still significant. In contrast, data integration systems that are nearer to the point-solution end of the spectrum – such as EnsEMBL, GenoMax and SRS – have considerably better user interfaces that help a biologist to build the simpler type

A biologist may find it frustrating that the graphical user interfaces of EnsEMBL, GenoMax and SRS cannot let him or her express a particular *ad hoc* query

of queries. Of course, a biologist may find it frustrating that the graphical user interfaces of EnsEMBL, GenoMax and SRS cannot let him or her express a particular *ad hoc* query such as the one that asks for the sequence of the first exon of all genes in a database. However, it is very likely that the same biologist may also find it equally frustrating that they do not know how to express that query in DiscoveryLink, OPM and Kleisli, even though they know that the query is expressible in these systems. In other words, the more general data integration systems can directly increase the productivity of a bioinformatics programmer, but they probably cannot directly increase the productivity of an average biologist.

The central distinguishing feature of TAMBIS is the presence of an ontology and a reasoning system over the ontology

We see a dichotomy between expressiveness and simplicity

Drawing from the remarks above, we see a dichotomy between expressiveness and simplicity. Therefore, which type of data integration system is preferred necessarily depends on the trade-off between these two factors. Many problems in biomedical research on drug targets and candidates require access to many data sources that are voluminous, heterogeneous, complex and geographically dispersed. If these data sources are successfully integrated into a new database, researchers can then uncover relationships that enable them to make better decisions on understanding and selecting targets and leads. Therefore, a successful integration of data is crucial to improving productivity in this research. However, it is important to stress that a successful data integration must be in support of a specific research problem, and different research problems are likely to need different ways of integrating and analysing data. Even though a point solution such as EnsEMBL does not fare well as a general data integration system, it works much better than any general data integration system in the specific context of genome browsing. However, if one's data integration needs are of a more *ad hoc* nature, a general data integration system can often ease the implementation significantly as such a system provides greater adaptability. It is also worth remarking that the more specialised solutions may themselves be implemented on top of a more general data integration solution. One such example is TAMBIS,[29] which is built on top of Kleisli.

A successful data integration must be in support of a specific research problem

The systems surveyed so far generally do not consider the semantics aspect of the underlying data sources. Let us end this paper with a brief mention of TAMBIS. TAMBIS[29] is a data integration solution that specifically addresses the semantics aspect. The central distinguishing feature of TAMBIS is the presence of an ontology and a reasoning system over this ontology. The TAMBIS ontology contains nearly 2,000 concepts that describe both molecular biology and bioinformatics tasks. TAMBIS provides a user interface for browsing the ontology and for constructing queries. A query is formulated by starting from one concept, browsing the connected concepts and applicable bioinformatics operations in the ontology, selecting one such connected concept or applicable bioinformatics operation, and browsing and selecting for further connected concepts and applicable bioinformatics operations. The ontology and the associated reasoning component thereby guide the formulation of the query, ensuring that only a query that is logically meaningful can be formulated. The query is then translated by TAMBIS and passed to an underlying Kleisli system for execution. From the point of view of TAMBIS, Kleisli significantly simplifies the task of implementing TAMBIS, as the TAMBIS implementers can concentrate on the ontology and reasoning components and leave the details of handling the underlying data sources to Kleisli. From the point of view of Kleisli, TAMBIS makes it possible for a biologist to ask more complicated *ad hoc* queries on the data sources integrated by Kleisli without the assistance of a programmer. The ontology of TAMBIS is currently being enriched by its inventors in the University of Manchester to allow an

even larger range of complicated queries to be expressed.

References

1. Baker, P. G. and Brass, A. (1998), 'Recent development in biological sequence databases', *Curr. Opin. Biotech.*, Vol. 9, pp. 54–58.
2. 'Practical data integration in biopharmaceutical R&D: Strategies and technologies' (2002), White Paper, 3rd Millennium Inc., Cambridge, MA.
3. Wong, L. (2000), 'Kleisli, a functional query system', *J. Funct. Prog.*, Vol. 10, pp. 19–56.
4. Davidson, S. B. *et al.* (1995), 'Challenges in integrating biological data sources', *J. Comput. Biol.*, Vol. 2, pp. 557–572.
5. Brenner, S. E. (1999), 'Errors in genome annotation', *Trends Genet.*, Vol. 15, pp. 132–133.
6. Schoenbach, C. *et al.* (2000), 'Data warehousing in molecular biology', *Brief. Bioinformatics*, Vol. 1, pp. 190–198.
7. Ullman, J. D. (1989), 'Principles of Database and Knowledgebase Systems I', Computer Science Press, Rockville, MD.
8. Bairoch, A. and Apweiler, R. (2000), 'The SWISS-PROT protein sequence database and its supplement TrEMBL in 2000', *Nucleic Acids Res.*, Vol. 28, pp. 45–48.
9. Hubbard, T. *et al.* (2002), 'The EnsEMBL genome database project', *Nucleic Acids Res.*, Vol. 30, pp. 38–41.
10. Etzold, T. and Argos, P. (1996), 'SRS: Information retrieval system for molecular biology data banks', *Methods Enzymol.*, Vol. 266, pp. 114–128.
11. Haas, L. M. *et al.* (2001), 'DiscoveryLink: a system for integrated access to life sciences data sources', *IBM Syst. J.*, Vol. 40, pp. 489–511.
12. Chen, I.-M. A. and Markowitz, V. M. (1995), 'An overview of the object-protocol model (OPM) and OPM data management tools', *Informat. Syst.*, Vol. 20, pp. 393–418.
13. Abiteboul, S. *et al.* (1999), 'Data on the Web: From Relations to Semistructured Data and XML', Morgan Kaufmann, San Francisco, CA.
14. Achard, F. *et al.* (2001), 'XML, bioinformatics and data integration', *Bioinformatics*, Vol. 17, pp. 115–125.
15. URL: http://www.informaxinc.com/solutions/genomax
16. Altschul, S. F. *et al.* (1997), 'Gapped BLAST and PSI-BLAST: a new generation of protein database search programs', *Nucleic Acids Res*, Vol. 25, pp. 3389–3402.
17. Burge, C. and Karlin, S. (1997), 'Prediction of complete gene structures in human genomic DNA', *J. Mol. Biol.*, Vol. 268, pp. 78–94.
18. Marcotte, E. M. *et al.* (1999), 'Detecting protein function and protein–protein interactions from genome sequences', *Science*, Vol. 285, pp. 751–753.
19. Codd, E. F. (1970), 'A relational model for large shared data bank', *Comm. ACM*, Vol. 13, pp. 377–387.
20. Chen, P. P. S. (1976), 'The entity-relationship model: toward a unified view of data', *ACM Trans. Database Syst.*, Vol. 1, pp. 9–36.
21. Selletin, J. and Mitschang, B. (1998), 'Data-intensive intra- & internet applications – experiences using Java and CORBA in the World Wide Web', in 'Proc. 14th IEEE Int. Confl. Data Engineering', IEEE Computer Society, Los Alamitos, CA, pp. 302–311.
22. Wong, L. (2000), 'Kleisli, its exchange format, supporting tools, and an application in protein interaction extraction', in 'Proc. IEEE Intl. Symp. Bio-Informatics and Biomedical Engineering', IEEE Computer Society, Los Alamitos, CA, pp. 21–28.
23. Robie, J. *et al.* (1998), 'XML Query Language (XQL)', in 'Position Papers of QL'98 – The Query Languages Workshop' (URL: http://www.w3.org/TandS/QL/QL98/pp/xql.html).
24. DeRose, S. J. (1998), 'XQuery: a unified syntax for linking and querying general XML documents', in 'Position Papers of QL'98 – The Query Languages Workshop' (URL: http://www.w3.org/TandS/QL/QL98/pp/xquery.html).
25. Fernandez, M. F. *et al.* (2001), 'Efficient evaluation of XML middle-ware queries', *SIGMOD Record*, Vol. 30, pp. 103–114.
26. Florescu, D. and Kossmann, D. (1999), 'Storing and querying XML data using RDBMS', *Data Eng. Bull.*, Vol. 22, pp. 27–34.
27. Abiteboul, S. *et al.* (1995), 'Foundations of Databases', Addison-Wesley, Reading, MA.
28. Schuler, G. D. *et al.* (1996), 'Entrez: molecular biology database and retrieval system', *Methods Enzymol.*, Vol. 266, pp. 141–162.
29. Goble, C. A. *et al.* (2001), 'Transparent access to multiple bioinformatics information sources', *IBM Syst. J.*, Vol. 40, pp. 532–552.