INFORMATION INFRASTRUCTURE

FOR

THE HUMAN GENOME PROJECT

Robert J. Robbins[1]

US Department of Energy
robbins@er.doe.gov

Johns Hopkins University
rrobbins@gdb.org

---

[1] *Corresponding Author:*   Robert J. Robbins
1300 Southview Road
Baltimore, MD 21218

rrobbins@gdb.org

(phone: 410 467 9432)

TABLE OF CONTENTS

INFORMATION INFRASTRUCTURE FOR THE HUMAN GENOME PROJECT[1]

Robert J. Robbins

The original goals of the Human Genome Project (HGP) were: (1) construction of a high–resolution genetic map of the human genome; (2) production of a variety of physical maps of all human chromosomes and of the DNA of selected model organisms; (3) determination of the complete sequence of human DNA and of the DNA of selected model organisms; (4) development of capabilities for collecting, storing, distributing, and analyzing the data produced; and (5) creation of appropriate technologies necessary to achieve these objectives [15]. Goals 1–3 laid out the challenge for bench research, Goal 4 recognized the essential role of data management, and Goal 5 was a frank admission that the project was begun before the necessary technologies were in hand. In the spirit of that candor, it is appropriate to ask whether the HGP is meeting its goals and in particular whether the computational components will be adequate for handling the volume and complexity of data generated by the project.

In this essay we assert that the most pressing information–infrastructure requirement now facing the HGP is achieving better interoperation among electronic information resources. Other needs may be equally *important* (better methods to support large–scale sequencing and mapping, for example), but none are as *pressing*. The problem of interoperability grows exponentially with the data. Efforts to develop distributed information publishing are now underway in many locations. If the needs of the genome project are not soon defined and articulated, they will not be addressed by these external projects. *De facto* standards will emerge and if these prove inadequate for scientific data publishing, the research community will have little choice but to tolerate this inadequacy indefinitely.

THE CHALLENGE OF GENOME DATA MANAGEMENT

Figure 1 shows the growth in the world's sequence databases from the first release of GenBank to 1994. Although the data volume is increasing exponentially, with a doubling time less than two years, merely keeping up is no longer a problem. The sequence databases were falling far behind the data flow in the mid 1980s [5], but technical and sociological advances now allow the databases to

---

[1] The ideas in this paper are the opinions of the author and do not necessarily represent the views of the US Department of Energy or of any other Federal agency.

absorb easily a far greater amount of new information than previously conceivable. In 1986, 13 *months* elapsed between the publication of a sequence and its appearance in the databases. Now, the Genome Sequence Data Base processes a typical submission within 13 *hours*. Every 2–4 weeks, more sequence data enter the databases than did so in the first five years of their existence.
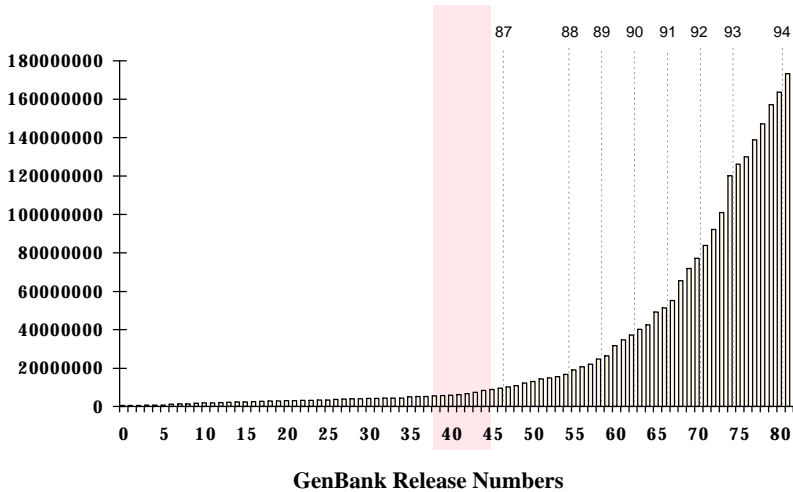

Figure 1. Growth in the world's collection of nucleotide sequence data, shown as the number of bases contained in every release of GenBank from 1 through 82. The numbers at the tops of the dotted lines show years (which do not necessarily coincide with a particular number of releases). The shaded bar in the middle represents the period in the mid 1980s when the data volume was, for a time, more than the databases could handle. (Data supplied by Michael Cinkosky and Dennis Benson.)

The map databases are also keeping up with the growth in the number of genetic markers in humans and selected model organisms. The past crisis of data acquisition has been resolved, leaving us to face a new and inherently more difficult crisis of data integration.

The importance of integrating genome information resources has been recognized in reports from groups of leading biologists (e.g., the Genome Science and Technology Center directors; [3]) and of informatics experts (an invitational meeting held in Baltimore in April, 1993; reported in [11]):

A...major...goal of genome informatics should be the integration of genome and genome–related databases. [3]

Achieving coordination and interoperability among genome databases and other informatics systems must be of the highest priority. We must begin to think of the computational infrastructure of genome research...as a federated information infrastructure of interlocking pieces. [11]

For historical and operational reasons, HGP data are now, and will continue to be, housed in several independent data resources. Already, the lack of interoperability among these resources makes answering simple questions overly difficult, leading the Baltimore report [11] to observe:

> An embarrassment to the Human Genome Project is our inability to answer simple questions such as, "How many genes on the long arm of chromosome 21 have been sequenced?"

Removing this embarrassment will require several interoperability improvements:

- *Technical interoperability* must be achieved, so that minimum functional connectivity can be assumed among participating information resources. This would require network connectivity and database interoperability.

- *Semantic interoperability* must be developed, so that meaningful associations *could be made* between data objects in different databases. This would require enough agreement about the meaning of the data so that assertions about relationships would be at least possible.

- *Social interoperability* must occur, so that meaningful associations *are made* between data objects in different databases. This would require sufficient social pressure to motivate the creation, entry, and maintenance of this information, since each asserted link between data objects is an act of scientific creativity and must be made on the basis of expertise, not merely through routine computations on existing data.

These advances will likely occur in the order given. Without technical interoperability, the motivation for providing semantic interoperability is lacking. Without semantic interoperability, it is difficult to define, much less enter, links between objects.

Another embarrassment is the time that genomic databases have been promising, but not delivering, connectivity with other information resources. The problem has been a simple absence of the technical interoperability infrastructure necessary to enable and motivate the remaining work. However, recent advances now promise that solutions may soon be at hand. This essay will describe some relevant trends and advances and will describe a reference architecture to facilitate the remaining steps. For reasons of space, the essay will not treat either semantic or social interoperability. Semantic compatibility and other aspects of genome informatics have been discussed elsewhere [7],[8],[[9],[10],[11].

A Taxonomy of Multidatabase Approaches

Although the development of truly interoperable federated database systems is still considered a research problem in computer science (e.g., see the collection of papers in [4]), there have been many calls for a federated approach to the management of information in biology, both within and without the HGP.

The vocabulary used to describe interoperating distributed computer systems varies among authors. In this essay we follow the terminology and taxonomy of Sheth and Larson [14]: A *multidatabase system* supports simultaneous operations on multiple (perhaps different) component databases. A *federated database system* (FDBS) has autonomous components, whereas the components in *non–federated database systems* are under a unified management. A federated system with no strong central federation management is considered *loosely coupled*. One with strong central management and with FDBS administrators controlling access to the components is *tightly coupled*. Tightly coupled systems can have one or more centrally managed federated schemas.

Tightly coupled FDBSs offer several advantages, such as clearly integrated views for users and the ability to update participating databases. These systems are, however, fragile when changes occur in the participating databases and they have proven difficult to achieve in practice, even within single corporations under unified management [2]. Loosely coupled systems are more easily achieved, but they can put much of the data–integration burden on users (or third–party developers). Many authors consider the problem of coordinated updates across loosely coupled FDBSs to be essentially insoluble.

Biological Information Resources as Publishing

Databases within commercial enterprises are information resources that determine the behavior of the organization. Paychecks are issued, products manufactured, shipments made, and invoices sent, according to the contents of the enterprise's databases. Since acting on the basis of inconsistent data would lead to chaos, both within the enterprise and with its external interactions, commercial database management systems have emphasized update methods that maintain internal data consistency and data integrity. Not unexpectedly, this emphasis has carried over into research efforts to develop multidatabase systems.

Scientific community databases, however, have more in common with scientific publishing than with business database management systems. Projects such as the Genome Data Base or GenBank offer communication channels through which observations, *sometimes inconsistent observations*, may be shared among researchers. The role of databases in communication has been explicitly recognized by leading genome researchers [6]:

> Public access databases are an especially important feature of the Human Genome Project. They are easy to use and facilitate rapid communication of new findings (well in advance of hard–copy publications) and can be updated efficiently.

These biological information resources, as seen by users, are better conceived as *database publishing systems* (DBPSs), not as database management systems (DBMSs). Although DBMSs are used to build some of these information resources, when the data are made available to users, they are "published" in a sense, and it is read–only interoperability among *all* the resulting DBPSs that is

greatly needed by the broad scientific community. (Update interoperability involving smaller subsets, sometimes just pair–wise combinations, of the underlying DBMSs is also needed, but that will not be discussed here.)

Achieving read–only interoperability among loosely coupled DBPSs is much easier than doing so with read–write DBMSs. With DBPSs, the notions of "loosely coupled" and "tightly coupled" are better considered as naming the ends of a continuum of relationships, rather than designating two mutually exclusive states. Figure 2 illustrates some possible points along the continuum.
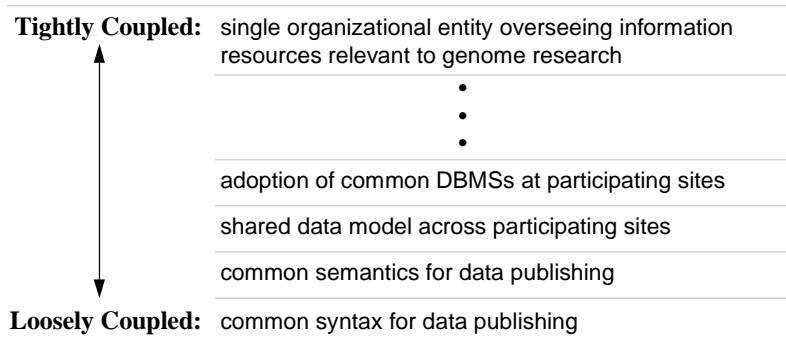
Tightly Coupled:	single organizational entity overseeing information resources relevant to genome research
	•
	•
	•
	adoption of common DBMSs at participating sites
	shared data model across participating sites
	common semantics for data publishing
Loosely Coupled:	common syntax for data publishing

Figure 2. The distinction between tightly coupled and loosely coupled systems, seen as designating the ends of a continuum of relationships among database publishing systems. The tightest level of coupling yields a completely integrated, single management structure. The loosest level of coupling involves merely a collection of wholly independent organizations that publish their data in a common syntax.

Stand–alone database management systems provide robust local functionality, but low interoperability across heterogeneous sites. Loosely coupled generic, read–only systems, such as gopher and World–Wide Web (WWW), provide wide interoperability, but with lower local functionality. Because the incremental cost of mounting gopher and WWW servers is small for those already building large local databases, many biological information resources are now using gopher and WWW to supplement, not replace, existing services.

The value of participation in widely available generic systems, especially to users, can be astoundingly high, since the overall value of an interoperable network of cross–referencing information systems increases non–linearly with the number of participants. *Thus, for the HGP in particular and for biology in general, attaining increasing generic database interoperability among all relevant information resources must be a continuing goal*.

ACHIEVING INTEROPERABILITY

Many hold that achieving full read–and–write interoperability across multiple databases requires an integrated data model, or schema, spanning the participating information resources. A recent refinement is the integration of only portions of the local schema, which may be specially modified to facilitate integration. These modified subschemas are known as export schemas. [4],[14]

Export schemas buffer against changes in the underlying databases, but only if the export schemas themselves are stable. Ultimate fragility due to inevitable changes in the underlying systems has led Chorafas and Steinmann [2] to dismiss global schema integration as impractical and to characterize such attempts as an "approach which has been tried and failed since 1958".

Evolution of Complex, Integrated Systems

Building large, complex software systems is best done through the assembly of stable, interoperating components. Attempts to build truly large systems as integrated monoliths rarely succeed, since the inter–related complexity of the resulting behemoth soon exceeds the ability of programmers and managers to track and maintain. Ideally, systems should be based, at least in part, on a foundation of components developed elsewhere, since without some cumulative development continued functional advances cannot occur

In 1982, Frederick Brooks observed [1] that writing a simple, stand–alone program is relatively easy, compared with the additional effort required to extend that program so that it acquires product–like qualities of robustness and portability, or so that it can function as a component in a complicated system. Crossing the complexity boundaries to achieve these improvements, Brooks estimated, increases the level of effort at least ten–fold, but both are required to produce the programming systems product, "the truly useful object, the intended product of most system programming efforts." In short, building good components is hard work, but essential, if large integrated systems are the goal.

If Brooks' insights are any guide, *the development of interoperating biological databases will require an engineering solution that maximizes the utility and cost–effectiveness of the entire system, not the elegance of individual components.*

Historical Trends in Database Management

Early on, managing data was seen as just another computational problem, to be solved by local programmers. Custom solutions were developed to handle all aspects of the system's behavior, with the exception of a few basic services, such as file management, provided by the operating system. Over time, the realization that nearly all data–management problems require certain common services led to the development of commercially available DBMSs.

DBMSs provide their services transparently, so that developers need only specify *what* must be done, while allowing the underlying DBMS to determine *how* it will be accomplished. Thus, when specific applications are produced using a particular DBMS, the overall system can be seen as operating in two parts:

- a top layer consisting of the application program itself, and
- a bottom layer, or layers, consisting of relatively transparent services provided by the DBMS and called by the application program as needed.

The general trend has been to increase the activities delivered as transparent services. Early DBMSs added a data model (schema) to the application, abstracted the underlying data structures into records, and moved the processing of input and output into generic tools. The relational model further abstracted the data structures into tables and pushed access methods into the generic tool layers. Object–oriented databases are now moving even more into the generic tools layer, while abstracting the data structures into objects that encompass both data and methods—code that manipulates the data.

This pattern of increasing reliance upon generic services continues to spread, with some systems, exemplified by WWW, merging generic information–retrieval and network tools into a conceptually unified, yet physically distributed information space.

Layered Architectures in the Networking Model

Networking has followed a similar evolution, with generic functionality being pushed increasingly into layers below the executing application. Early networking solutions were *ad hoc*, local, and proprietary, so that application programs had to be custom designed for a specific network infrastructure. Now, however, *generic* networking protocols allow application programs to exchange *specific* messages transparently.

A layered stack of software protocols allows application programs running on physically separated computers to interact as if they were directly connected. Each application program communicates to a layer just underneath it, according to standardized protocols. Bottom layers on the sending system prepare the message for transmission on some physical medium. Those same layers on the receiving system retrieve the message from the physical medium, then decode and reconstruct it, so that the message presented to the application program on system B is exactly what was sent by the application program on system A (Figure 3). Because it would be impossible to guarantee perfect transmission of every packet on the physical medium, appropriate metadata are transmitted so that the receiving system can determine that packets have been lost or damaged and request retransmission.

In this layered architecture, responsibility for determining *how* various tasks are to be accomplished resides within individual layers. Higher layers need only be aware of *what* services are provided by the lower layers. Although communication

protocols *between* layers are well defined and stable, the *internal* details of how a layer is implemented may be changed at any time.

The layering of responsibility for *how* things are to be done while preserving a well–defined stack of *what* services are needed has given modern networking its great strength and flexibility. So long as a layer continues to meet the specifications for what is to be performed, improvements may be made in how it carries out its tasks, without necessitating changes in other layers. Entire layer modules may be readily snapped out, and others substituted. For example, an underlying ethernet layer may be replaced with FDDI, with no changes whatsoever required in programs running at the application layer.
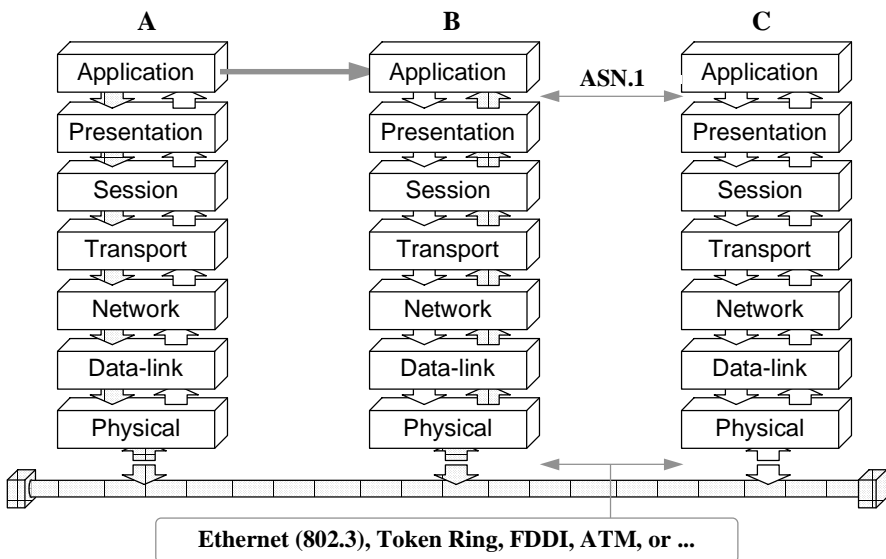

Figure 3. The seven layers of the ISO–OSI reference model. Virtual connectivity between applications on different computers is accomplished via direct communication between adjacent layers within each system, according to well–defined protocols. For example, abstract syntax notation (ASN.1) defines the communication protocols between the application and presentation layers. (ASN.1 is a powerful scheme for representing data of arbitrary complexity. This has led some [cf. Ostell, this volume] to devise very clever, non–networking uses for the protocol.)

The experience of networking illustrates an important *principle of distributed, scalable design: distributed interoperating systems benefit from layers of collectively designed, but independently developed components, interacting through defined, stable, open protocols.*

Interoperating Genome Information Resources

Obtaining interoperability among genome and genome–related databases involves two related, but distinguishable goals:

- *Increase the homogeneity of participating genome systems* (i.e., tighten the coupling among the data systems of the genome community). This would allow genome data to be more easily obtained from multiple sites in a common format appropriate for integrated analyses. This will require achieving greater semantic and social interoperability among the systems.

- *Develop general interoperability while tolerating loosely coupled heterogeneous systems* (i.e., participate in a loosely coupled federation of general biological information resources). This would allow the further integration of genome data with other relevant data, such as metabolic information, structural biology data, comparative findings, etc. These needs could be met through a more loosely coupled, read–only approach.

These are not mutually exclusive and, in fact, are more likely to be mutually reinforcing. Efforts to achieve intra–community homogeneity can be made at the same time that steps are taken to permit interoperation with heterogeneous systems. Both paths should be followed simultaneously, since within a small, cooperating community (e.g., a few collaborating sites), homogeneity may be attainable, whereas in larger communities heterogeneity is inescapable.

Although achieving a moderately tightly coupled architecture might be a useful goal for genome databases, simultaneous participation in a loosely coupled system is also necessary for several reasons:

- It will take time to increase the coupling across *all* genome databases and there is a need to improve interoperability before improved coupling becomes widespread.

- Although some genome databases will likely converge upon a few common data models and database systems, the probability that the entire community will converge upon a *single* standard is essentially zero. Therefore, loosely coupled interactions among different groups of genome databases are essential.

- The HGP community will receive real benefits from participating as a component in other, larger federated information infrastructure systems, and these larger systems will surely be loosely coupled at best.

- The use of loosely coupled distribution systems is actually likely to facilitate the development of more tightly coupled approaches. The ready availability of data from multiple sources will give wide exposure to arbitrary differences in data models, resulting in significant community pressure for convergence upon greater semantic consistency.

If networking experience is a guide, any suggestion that the HGP requires *only* highly coupled information resources must be soundly rejected. Network connectivity is useless unless an entire stream of data can be moved reliably from one host to another. But networking technology cannot guarantee the delivery of any given packet. Therefore, reliable end–to–end transmission of data streams is implemented *on top* of an unreliable packet–transfer system. Protocols in higher layers detect the inevitable occurrence of lost or damaged packets and request that they be resent, so that ultimately a reliable duplicate of the data stream from the sender can be reassembled at the receiver. If efforts had been made to implement complete reliability at the packet transfer layer, effective networking would still be a far–off dream. This experience illustrates an important *principle of appropriate foundations: sometimes robust solutions are best built upon seemingly weak foundations*.

RECENT ADVANCES

New methods for creating loosely coupled federations of read–only electronic publications are being rapidly adopted across the Internet. The implications of these advances for genome informatics are best appreciated after first briefly considering historical patterns in the development of bio–informatics systems.

Evolution of Biological Information Systems

With early computerized biological information resources, users had to install the entire system, software and data, on a local computer before the local value of the resource could even be tested. As installing these systems was expensive, both in effort and in resources, their appeal was limited.

The next step was the development of dedicated client–server systems, in which the data resided on a centrally located server and only the client software had to be installed locally. This approach had several advantages, especially in reducing the local disk–space requirements and in providing access to up–to–date data without requiring that the data be distributed to all users. It suffered from not providing any interoperability among different information resources.

The most recent step has been the emergence of loosely coupled generic client–server systems, such as WWW. Here, a single generic client is capable of accessing data on any server that "publishes" its data according to the generic server protocol. Incorporated in the protocols are the ability of one data server to "reference" information present on another server. This allows the ready creation of a basic form of read–only interoperability.

Much of the power of this approach derives from the way generic components and standard cross–referencing schemes allow the user to perceive systems to be interoperating when in fact the operators of the systems may not even know that each other exists. This illustrates an important *principle of anonymous*

interoperability: scalability is greatly enhanced if interoperation can be effected between anonymous partners.

Power of Generic Client–Server Computing

Although database research has emphasized tightly coupled approaches, recent experience has shown that appropriate protocols, coupled with middleware tools, can make some loosely coupled systems incredibly effective, albeit in a read–only manner. The two most successful recent internet applications have been gopher and WWW, and both of these are loosely coupled federations serving read–only multi–media hypertext and other file–based resources. User have voted with their feet, so to speak—over the past 18 months, usage of these products has increased nearly twenty–thousand–fold, while the internet itself has grown only four–fold.

Distributed Object–Oriented Programming

In addition to being generic client–server systems, gopher and WWW also exemplify a rudimentary form of *distributed object–oriented programming*. Data objects, along with the *names* of methods that may be used with the objects, are provided by the servers, while generic clients contain the actual hardware–dependent *binaries* necessary for executing the methods. This distribution of data and methods between server and client offers a powerful and extensible system for searching, browsing, and retrieving data of a variety of types.

Gopher and WWW

Gopher is a loosely coupled federation of standard file servers, distributed around the world, accessed with a copy of generic client software. Any gopher server can be interrogated from any client. The basic interface is the simple menu, with every menu choice either (1) retrieving another menu, (2) retrieving text, data, graphics, software, or other files, (3) initiating a query directed to a specific database, or (4) initiating a search for more menu items. The power of gopher lies in the invisibility of its infrastructure to users, who feel they are just making choices from options presented by a single system when in fact they can be jumping from computer to computer, around the world.

At its most basic level, each gopher transaction is basically a "please send me a thing named X" request directed to a particular server, followed by the sending of X. The server has no knowledge, nor any need for knowledge, about how it happened that this particular client asked for that particular object. The elegance is in the simplicity—at base level, every transaction is just a request–response exchange. Additional functionality is achieved by layering other logical functions on that fundamental transaction.

Although gopher permits cross referencing between *servers*, in the sense that menus on any server can reference files or other menus on other servers, it does not support cross referencing at the level of actual objects being returned. The WWW

approach, however, moved the ability to cross reference into the data objects themselves, thus creating a distributed hypertext information space.

WWW architecture is based on: (1) a standard (hypertext markup language, or HTML) for producing formatted text that may contain embedded cross references to other such files, (2) a naming scheme (Uniform Resource Locators, or URLs) that allows for the unambiguous resolution of such embedded references, and (3) a protocol (hypertext transfer protocol, or HTTP) for the efficient retrieval of documents in a hypertext environment. The WWW philosophy includes a commitment to providing access to information via older protocols (e.g., ftp, WAIS, gopher) as well as being sufficiently extensible to accommodate new protocols as they become available. Like gopher, WWW–conforming systems can spawn external "viewer" programs to present new data types to the user.

The National Center for Supercomputing Applications (NCSA) developed and released Mosaic, a graphical browser into WWW information resources. Mosaic added the ability to display graphical images directly in the browser, so that HTML source pages could contain embedded references to graphics files, which would be displayed as images in the basic browser display. Extensions now allow the embedding and presentation of sound and full–motion video, creating a multi–media hypertext gateway into the Internet information space. The success of Mosaic has stimulated other developers, so now there are many WWW browsers available, each competing with each other to add new functionality.

A truly remarkable aspect of the WWW phenomenon has been its rapid acceptance. Client usage has increased because of ease of use, but probably more importantly because of the rapid proliferation of WWW information servers. The value to the user of an integrated set of information resources increases greatly with the number of participants. The number of WWW sites has now reached the point where a run–away positive feedback system has been generated.

Nearly every major biological database now publishes information via WWW, and more resources are coming on–line daily. Anyone with a computer attached to the internet and a copy of WWW server software (available free) can become a publisher of electronic information simply by preparing a few files in HTML format, making them available through their server, and then simply sending out an announcement of the new resource and giving its name in URL format. From that moment on, all of the millions of users with client software have instant access to the resource. This illustrates an important *principle of value explosion: once the number of components in an interoperating network of information resources passes a critical size, the overall value of the network grows explosively*.

Networks as Distributed Information Spaces

The generic client–server system for retrieving information from the Internet, exemplified by WWW, has stimulated a new vision of just what the Internet represents. Schatz and Hardin [13] note:

> Originally intended as a distributed network of computers, [the Internet] is increasingly viewed instead as a distributed space of information. Rather than

transferring files between computers, a user navigates an information space of distributed items of information. The users concentrate on the logical structure of the interconnection of information and data items rather than on the underlying physical stricture of computer and communication systems.

This new concept of the internet raises many interesting challenges, too numerous to consider here. Some relevant discussions may be found in [17].

Middleware Extends Functionality

Initially, gopher and WWW systems were available only for retrieval of text and file–based information or for multi–media hypertext. However, clever extensions, particularly in the development of middleware and gateways to other systems, are allowing these tools to access more structured data and to provide an apparently integrated joint interface to more than one server.

Powerful middleware can be developed simply by sandwiching custom code between a generic server and a generic client. Users accessing the server side see only an integrated data resource that returns information according to standard protocols. The server, acting as middleman, takes information provided by the user, manipulates it using whatever custom routines are needed, dispatches the results to one or more local or remote servers, receives, processes, and integrates the results from the various servers, and finally presents them to the user.

Examples of middleware can be found that provide general network services, such as the veronica search engine (which helps users locate information resources in GopherSpace), or that meet specific needs for a target community, such as the Johns Hopkins University's GenQuest server (which provides sequence analysis services to the molecular biology research community via a WWW server).

The GenQuest Server

GenQuest (available as a choice on the Johns Hopkins Computational Biology home page; URL = http://www.gdb.org/hopkins.html) uses a WWW forms interface to offer a variety of analytical algorithms (e.g., Smith–Waterman, FASTA, Blast) for analyzing nucleotide or protein sequences. The user selects the kind of sequence to be analyzed and the algorithm to be used, sets parameters for the algorithm, pastes the sequence into a receiving window, and clicks a button to initiate the analysis. The software at Hopkins reformats the query and sends it to an on–line analysis server at Oak Ridge, Tennessee. The output from Oak Ridge is reformatted into HTML, with hot links added dynamically to all referenced objects available via WWW. The results are then returned to the user's client software as a standard HTML page, with hot links established to all external data objects referenced in the report. The user may then navigate over the hot links to obtain related information.

Dan Jacobson (danj@gdb.org) was able to assemble GenQuest very easily, because a powerful on–line compute server, capable of returning analyses quickly enough to service a real–time interface, was available at Oak Ridge and because all

of the databases referenced in reports from the Oak Ridge server publish their data using standard WWW protocols and servers. The availability of these resources via standard on–line protocols allowed Jacobson to create an apparently unified information resource simply by providing value–adding integration through pure third–party middleware. *Explicit support and encouragement for value–adding activities by third–party developers must be a guiding principle for genome informatics.*

Middleware Allows Unilateral "Collaborations"

Operators of public databases often find themselves besieged with would–be collaborators. No matter how public–spirited the proprietors of the database, they will have to turn some (occasionally many) potential collaborators away, because true collaborations require effort on the part of both parties and no public database has unlimited resources with which to pursue collaborations.

If, however, databases publish via widely used generic client–server systems, third–party developers can effect apparent "collaborations" simply by developing appropriate middleware to interact with the databases. For example, developing a system like GenQuest requires no active collaboration of any information resource providing data to which hot links are generated.

Schatz and Hardin [13] describe the power of such "unilateral collaboration" in the context of Mosaic and WWW extensibility:

> This is an example of the idea of "Open Information Systems," systems that allow for the easy integration of existing information sources and that can be *extended and expanded by users in ways that were often unanticipated by the original developers*. [emphasis added]

Empowering third–party developers to expand the functionality of federated information resources without requiring the active collaboration of the original developers promotes incredible functional growth at very low cost. This illustrates an important *principle of value additivity: in a well–designed information infrastructure, most value will ultimately be added by third–party developers.*

DATA PUBLISHING IN A LOOSELY COUPLED FEDERATION

Although many scientific information resources now use WWW technology to share their data with others, additional extensions are needed before such a system can become truly effective for publishing structured data, not merely textual information.

Why WWW is Not Enough

WWW is presently inadequate for retrieving and integrating some kinds of richly structured scientific data. A few issues are:

- Set–based retrievals are needed, which WWW does not directly support.

- Automated data retrieval must be supported. This requires an automatable means for extracting the intended semantics of published data objects (as can be done with data dictionaries of structured databases). WWW homepages have no established semantics and WWW provides no standard way to publishing the metadata necessary to declare semantics.

- A PROJECT operator, in the relational sense, is essential. Some data objects may have thousands of fields but a user may only need, say, three of them. The idea of retrieving them all, then editing locally is not efficient, since the database may contain tens or even hundreds of thousands of relevant objects.

- The ability to do automated, set–based, distributed joins (equivalent to a relational JOIN across distributed databases) across data in multiple servers is a crucial requirement for scientific data publishing. This will require a significantly different client and a significantly different server than is presently available with WWW.

- Identifiers that have much in common with relational primary and foreign keys are needed. URLs and embedded URLs as presently implemented do not have the necessary semantic constraints. WWW offers no support for referential integrity.

Some of the inadequacies in dealing with structured data stem from the developmental history of gopher and WWW. Both projects have intellectual ties with information retrieval (IR), not database development, and many differences exist between the needs of database users and the services delivered by IR systems:

- IR query systems support ambiguous queries and resolve them using probabilistic retrieval systems, whereas databases hold structured data and provide exact answers to well–formed, structured queries.

- Hypertext browsers are intended for human usability, with the assumption that they will present multiple navigation options to a human user. Database users frequently need a computational application programming interface with which to interact, so that they can direct an application program to extract and analyze data sets, then return the analytical results.

- Hypertext supports flexible linkages between objects, but more structured linkages, with defined semantics (such as a foreign key to primary key reference), are required for structured data.

The list could be extended. But, the goal here is to offer neither the definitive characterization of the problem nor the definitive solution. Instead, we wish to establish that, *in their present form*, the widely available IR tools for easily fetching text and hypertext do not meet the needs of those who desire integrated access into structured databases.

Protocol Extensions Needed

The limits are not only with WWW, but also with the networking protocols on which it is based. At present, the fundamentals of internetworking assume that the ultimate goal is to connect processes running on different hosts. IP addresses provide two–part, *network:host* identifiers. A process can be associated with a particular port on a given host, extending the identifier to *network:host:process*. URLs add one more level—*network:host:process:object*, with the hard–wired assumption that these are all related one–to–many, left–to–right.

What is needed instead is something that identifies databases independently of their host and objects independently of their location. And, more importantly, a system is needed that would allow one name to be associated with several different instances of the same database or object. For example, the Genome Data Base (GDB) is a scientific database that has a primary location in Baltimore, Maryland. However, there are also more than a dozen read–only, public copies of the database scattered around the world. A naming convention is needed that would let users request objects from GDB without having to specify which GDB location to use. However, allowing the user the options of specifying either a particular host or particular conditions (e.g., the nearest copy, the most current copy, the currently least–loaded server, the copy with the highest average bandwidth between it and the user, etc.) would be useful.

In short, rethinking of network architecture is needed, guided by expertise from the worlds of networking, information retrieval, and database development. Without all three, whatever results will likely be missing some key functionality. A good discussion of extended network–protocol functionality needed for the future can be found in [17].

Reference Architecture for a Federated Object–Server Model

In a keynote address at the Third International Conference on Bioinformatics and Genome Research, Robbins [12] introduced a reference architecture for a *Federated Object Server Model* (FOSM) as a "robust straw man." (A *reference architecture* summarizes a system's basic functional elements and the interfaces between them. It identifies needed protocols and suggests groupings of functionality, but it does not imply a physical implementation.) FOSM is a straw man in the sense that it is freely admitted not to be *the* (or even necessarily *a*) solution. But FOSM is also robust, in that it provides a focus around which requirements for interoperating structured databases may be considered. An outline of the FOSM concept, emphasizing some aspects of the data model, is presented here. A more detailed description is being prepared and preliminary drafts are available from the author.

FOSM Overview

Like WWW, the FOSM approach derives data structures and protocols from a vision of how a networked information space might operate. In FOSM, servers provide access to richly structured data objects that can contain semantically well–

defined cross references to other data objects, allowing the rough equivalent of distributed joins in a relational database. The FOSM concept entails a strong commitment to resource discovery and resource filtering. Resource filtering, the deliberate restriction of queries to "trusted" sources, is essential if retrieved data are to be passed directly to other software for analysis. Support for third–party, value–adding developers is central.

The FOSM approach is generally applicable to any set of information resources involving structured data. Examples would certainly include scientific data resources and also many types of commercial information, either to be published externally for customers or as an internal resource within an enterprise.

FOSM Assumptions and Requirements

A complete discussion of FOSM assumptions and requirements would require a book–length presentation. Some examples are given here.

Basic Assumptions

The FOSM system will follow a generic client–server design, emphasizing autonomy of local sites and enabling structured queries into structured data.

- FOSM sites will *publish* their data in a read–only format via a standard object–server system (although they will *maintain* their databases in whatever manner they choose).

- Generic client software will obtain data from the read–only federation.

- With a single query, users will be able to obtain *sets* of related data objects from multiple independent data resources.

General Requirements

The FOSM system:

- should be relatively impervious to changes in data volume or in the number of participating sites—i.e., scalability is essential.

- must facilitate value–adding activities by third–party developers.

- must be data driven and self configuring. This means that a naive client should be able to contact a server for the first time and, as a result of transactions with the server, produce a usable user interface and initiate a query dialogue.

- should provide a local (i.e., client side) API, as well as the networked API into the server.

- should permit "subscription" to user–constructed queries. That is, users should be able to capture the steps necessary to execute a query, then request the system to execute that same query on regular timed intervals, returning data to the user via some specified route (email, ftp, etc.).

- must retrieve data in both human readable and computable format.

- must provide support for multiple concepts of object identity.

- must provide support for resource discovery in a manner at least loosely equivalent to that offered by the data dictionary in a stand–alone database.

- must support the equivalent of foreign key to primary connectivity between objects in different databases.

- must be able to provide query operators more or less equivalent with the SELECT, PROJECT, and JOIN operators of relational databases.

- must provide some minimal support for domain and referential integrity across entries in multiple data resources.

- must support both outer and true equi–joins across distributed object servers. Semantically well–defined cross–referencing (equivalent to foreign key to primary key references in a relational database) must be representable in the data structures and traversable by the system software. It must be possible to traverse such links without mandatory human intervention (e.g., without mandatory mouse clicking).

Server Requirements

FOSM servers will need to provide actual data to satisfy queries and also metadata to support building and operating the client interface and other automatable tools. Servers will also need to provide some server–to–server information to help maintain external references.

FOSM servers must:

- provide full–function anonymous data serving. That is, their services should be fully available to clients unknown to the server until the first query arrives.

- support negotiation with clients regarding the details of protocols, data, and formats. For example, a client might specify the maximum amount of data it could receive in one transaction or negotiate handshaking protocols. In addition, clients might inform the server what methods the client can support or what services it will request of the server.

- support both value–based queries and identifier–based queries.

- serve several different kinds of objects: (1) "type objects" that document the structure of the data objects so that the client software can produce an appropriate query and retrieval interface; (2) "data objects" that contain the actual data of interest; or (3) "help objects" that contain help messages to be used by the client to provide context–sensitive help messages.

- support remote domain and referential integrity in external servers. That is, if objects in one FOSM server reference objects in another server, the second server should provide specific support to assist in maintaining the integrity of references towards it. This might take the form of an

EXISTS() function that would allow a server to verify the existence of an externally referenced object in its collection.

Client Requirements

To support the needs of database users, the FOSM client will need to be able to maintain more *customizable* functionality than does a Mosaic or other WWW browser. FOSM clients:

- will need to "negotiate" with FOSM servers regarding the format and structure of objects requested and regarding the parameters and protocols of exchange.

- must be able to build dynamically custom forms–based or graphical interfaces to allow the interrogation of any FOSM server. To do this, clients will obtain metadata describing the structure of objects served by a particular FOSM server.

- must allow users to manipulate the structure of data objects from one server, or combine structure objects from different servers, to build single, virtual objects against which unified queries may be dispatched. It is this functionality that would allow users to specify queries that are similar to relational PROJECT or JOIN operations.

- must support "batch" as well as interactive, retrieval operation. That is, users must be able to create and store queries and the software must be able to execute stored queries automatically at specified times or intervals, outputting the retrieved data automatically into local files or into local analytical software.

- must allow user customization of the local–software configuration and of the configuration of interfaces into particular databases.

Resource–Discovery Requirements

The FOSM approach assumes that users will need assistance in identifying relevant FOSM objects and servers. It also assumes that a key part of resource discovery is resource filtering—i.e., the explicit rejection of data objects from undesirable sources. Therefore, the FOSM approach supports the free development of "editorial" activities, so that editorial bodies may indicate approval for individual FOSM objects, or for individual FOSM servers, or for sets of objects or servers. Editorial annotations could be hierarchical. That is, an editorial board might wish to assign its approval to all of those objects already approved by editorial boards A, B, C, and D.

Resource discovery tools must be easy to locate and use. Therefore, access to FOSM resource–discovery tools should be a built–in component of the FOSM client. Whether the discovery information should be provided by a central, known source; by distributed search engines (like veronica); or by some significant extensions to self–propagating name systems (like DNS) is an open question.

Third–party Development Requirements

In a manufacturing economy, materials travel along extensive pathways of value–adding activities: e.g., ores are mined, metals extracted. parts fabricated, objects constructed, etc. A successful "information economy" must also support unlimited chains of interlocking value–adding activities.

Many desktop software packages now explicitly support value–adding plug–in modules from third–party developers, and some of these interfaces have become sufficiently generic that they have been adopted by competing manufacturers. For example, the same third–party graphics manipulation filters can be used to augment the functionality of either Adobe Photoshop or Corel PHOTO–PAINT.

Because FOSM recognizes the importance of value–adding developers, all aspects of the FOSM architecture must be designed either to provide explicit support for third–party activities or to avoid hindering third–party activities. For example, FOSM resource–discovery services should be designed to allow *any* third–party to provide value–adding classifications of FOSM servers or FOSM objects. Extended chains of value–adding activities should also be supported, such as allowing third parties to classify classifications developed by other third parties.

Data–Structure Requirements

Just as the HTML data structure is the key to WWW functionality, so an appropriate syntactic data structure will be required for handling structured data. The FOSM model does not specify or constraint the semantics of participating databases, just as HTML does not specify or constrain the contents of WWW documents. Thus, two FOSM databases might well choose to publish similar data objects in semantically different forms. This is acceptable in a FOSM environment, provided that both data servers published their data in the FOSM syntax.

FOSM data structures:

- (or some consistent representation of them) must be reasonably easy to understand. (This would facilitate the development of virtual objects by users and/or third–party developers.)

- must be able to represent considerable (arbitrary?) complexity.

- must be able to offer meaningful representations of data objects extracted from different underlying DBMSs (e.g., RDBMS, OODBMS, etc.).

- must be readily parsable.

- should be closed under basic retrieval and manipulation operations.

- must robustly and unambiguously support the ability of data objects to contain, as attributes, references to data objects published elsewhere.

- must be *self–describing*, so that almost anything can be represented, yet *constrained*, so that generic client tools can be developed.

FOSM data structures could exist at both a physical (as represented internally by the system) and a conceptual (as perceived by users) level. In this essay, we will consider only the conceptual aspects of the data structure.

FOSM Architecture

FOSM architecture is based on a generic client–server approach, with explicit support for middleware and other development by third–parties. A registry of FOSM information would support both direct queries and resource discovery activities. Whether the registry should be a central database, or a system that supports duplicated information propagation (such as domain name servers) is an open question. The registry would hold information about FOSM servers, FOSM objects (and versions), FOSM links, FOSM subfederations, FOSM editorial records, FOSM methods, FOSM names, FOSM cataloging, etc.

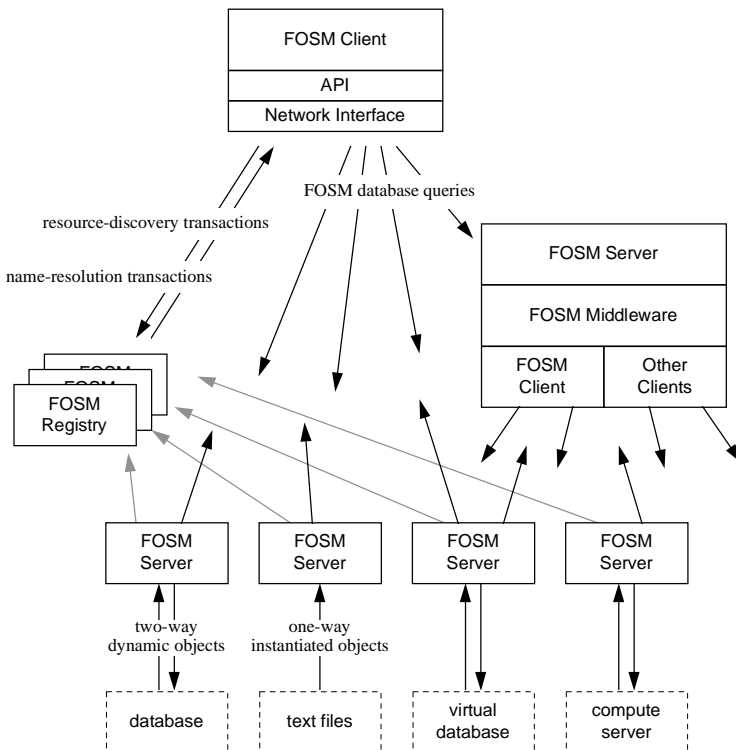An overview of the FOSM architecture is given in Figure 4.

Figure 4. FOSM clients interact with FOSM servers and with a FOSM resource registry. Servers publish holding information to the registry (gray arrows) and respond directly to client queries (black arrows). Explicit support for n[th]–party developers is provided, through the encouragement of middleware development.

The FOSM client (Figure 5) is built around a central kernel, the FOSM user–interface manager (UIM), which interacts with various local programs and remote servers. The UIM would probably be some kind of script interpreter, possibly a generic script interpreter so that more than one scripting language could be used. The UIM core is surrounded by a variety of other programs, which are invoked to call the local execution of "methods" associated with remote data objects, and files, which provide appropriate metadata and caches.
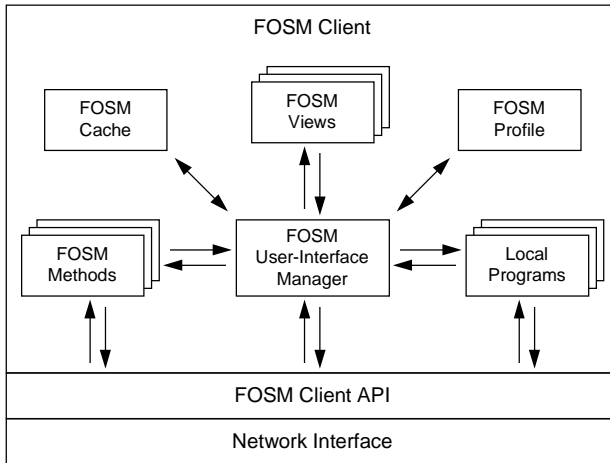

Figure 5. The FOSM client provides much of its functionality through its component–based design. All aspects of the FOSM system are intended to facilitate the value–adding activities of third–part developers. That is, it should be easy for users to install locally FOSM methods or views or profile components created elsewhere.

FOSM Data Model

A generic tree–shaped data structure provides a conceptual data representation that meets FOSM requirements, since a tree can capture the minimum essential subset of structure from relational, object–oriented, and other database systems. Each type of FOSM tree would represent one class of real–world objects and each individual FOSM tree would correspond with one member of that class.

Any data model that can be represented in an extended entity relationship (EER) schema can have read–only data objects extracted from it into tree–shaped configurations. Figure 6 shows how tree–shaped data objects may be extracted from a portion of an EER schema. Multiple occurrences in the tree of the same entity from the EER diagram indicates participation in different semantic roles. For example, the faculty data–object tree is rooted on the faculty entity and also includes "faculty" at two sublocations, one corresponding to the role of "departmental colleagues" and the other of "departmental chair." Individual FOSM

trees are one–to–many downward, and lower nodes can be considered as sets of sub–objects, related in some role as attributes of the next higher node.
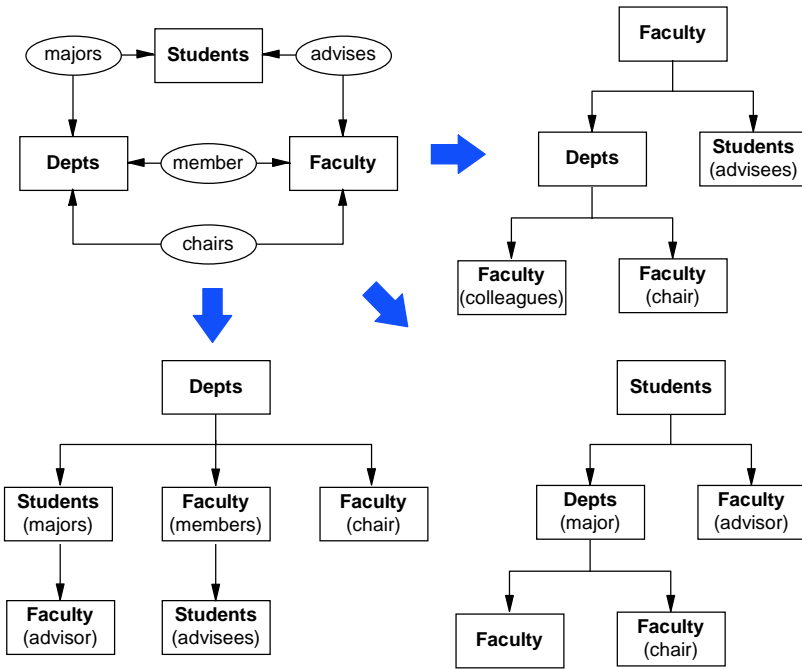
Figure 6. Tree data objects can be easily extracted from EER schemas. Here *faculty*, *department*, and *student* tree objects are all extracted from the same portion of a university database schema (represented as a directed graph). Notice that one node in the original schema may appear several times in a particular tree.

Individual tree–shaped data objects could be "selected" from a data server either through value–based or key–based queries. Once obtained, the data objects could be manipulated using operators such as "prune" and "graft" (Figure 7). These operators are similar to those of the "project" and "join" operations in relational databases. Prune and graft are "closed" in that they are defined to have well–formed trees as inputs and to produce well–formed trees as outputs.

Prune and graft could be combined to give a "promote" operation that could move nodes higher up the tree, eliminating intermediate nodes (and requiring some role definition refinements). The FOSM client would allow the user to create custom trees by pruning and grafting server–provided type trees, then store them locally to be used in driving queries to underlying data resources. This would give the ability to operate within a custom–tailored environment, while sparing servers from the need to maintain profile information on individual users.
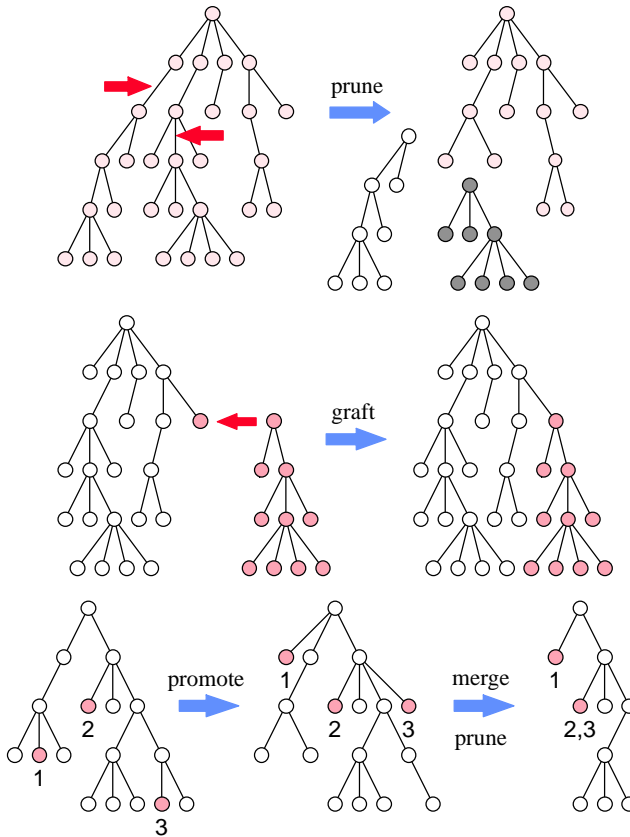
Figure 7. The "prune" operator is similar to the relational "project" operation. The "graft" operator is similar to the relational "join" operation. The "promote" operator allows the movement of nodes to higher positions in a tree, through a combination of pruning and grafting. If promotion results in multiple nodes defined over the same domain being attached at the same point in the tree, the "merge" operator combines them.

FOSM Data Identifiers

To be "federation ready" a FOSM server would have to provide absolutely stable, unambiguous identifiers for every rooted object in its published collection. Similarly, every external reference in a FOSM server would be in the standard format for global FOSM names. All rooted FOSM objects must be unambiguously identifiable in a global FOSM name space of arbitrary identifiers. Although biological names are too volatile to serve as primary FOSM identifiers, value–based queries of FOSM objects must also be supported so that researchers can interrogate the system using familiar terms. Indeed, one might expect that most key–based FOSM queries would be produced by software, not human users.

In a single copy of a stand–alone database, object identity is a fairly simple concept. However, in a FOSM system, *copies* of objects will be distributed from servers to clients where they may be stored for local use. Occasionally, then, clients will need to compare object copies to determine their equivalence. This raises subtle notions of identity.

For example, each FOSM object can be subdivided into five components: (1) a database identifier that specifies the information resource from which the object may be obtained, (2) a class identifier that specifies the class of objects to which the object belongs, (3) an associated type tree that specifies what attributes objects of that class *could* have (each FOSM *class* has one of these trees), (4) an object identifier that provides a unique identifier for the individual object, within the information resource, and (5) an associated data–value tree that specifies what attributes the particular object *does* have and gives their values (each FOSM *object* has one of these trees). (Note: because new findings sometimes lead to reclassifications of real–world objects, FOSM object identifiers should be unique within FOSM servers, not merely within FOSM classes, so that object identity could be preserved across category reclassification.)

This allows for several different concepts of equivalence, of which we will discuss four: *object equivalence*, *class equivalence*, *version equivalence*, and *value equivalence*. In all cases discussed below, it is assumed that the objects come from the same information resource.

- Two FOSM data objects exhibit *class equivalence* if they are from the same FOSM object class.
- Two FOSM data objects exhibit *version equivalence* if they are class equivalent and share the same type tree.
- Two FOSM data objects exhibit *value equivalence* if they are version equivalent and have identical data–value trees.
- Two FOSM data objects exhibit *object equivalence* if they refer to the same real–world object and they have the same object identifier. This is the most fundamental component of identity and it persists across value updates to the object's attributes and possibly even across schema updates to the object's type tree.

Combinations of these three equivalences lead to different kinds of identity:

- Two copies of FOSM objects are *semantically identical* if they exhibit class and object equivalence.
- Two copies of FOSM objects are *computationally identical* if they exhibit class, object, and version equivalence. However, computationally identical objects could have different values stored for the object's attributes.
- Two copies of FOSM objects are *truly identical* if they are computationally identical *and* they exhibit value equivalence.

Additional identity concepts could be derived from these. For example, we might want to say that two objects are *apparently identical* if they are class equivalent, with identical type and value trees, but different object identifiers.

To facilitate different kinds of identity comparisons, a FOSM object could carry two *computed* identifiers, a *type identifier* (defined over the object's type tree) and a *value identifier* (defined over its value tree), in addition to its already *assigned* class and object identifiers. These computed identifiers would be calculated on the fly, whenever an object is provided by a FOSM server, much as check sums are calculated anew each type an IP packet is placed on a physical medium. These calculated FOSM identifiers would also be useful for detecting corruption in local copies of FOSM objects.

Type identifiers could also be used to associate particular computational methods with FOSM objects. For example, semantically identical DNA sequence objects could be represented in computationally different FOSM trees that are equivalent to flat–file, ASN.1, BLAST, etc., formats. Each format would have a specific type identifier and this could be used automatically by software to determine the appropriate parser to be used in analyzing the data.

Schema version changes would also be reflected in type–identifier changes. To allow ready detection of specific versions, perhaps the type identifier should contain two parts: one specifically giving the version number and the other a computed value derived automatically from the contents of the type tree itself.

A major goal of FOSM is providing a scalable, automatable system for delivering structured data objects across a federation of autonomous resources. Achieving this will *require* that type identifiers contain a computed component so that software can check automatically to determine if it knows how to read and process the data. Data resource developers will differ in their personal notions of what changes are sufficiently significant to constitute a change in the designated version of the database. However, some third–party software may rely upon the precise configuration of data from a particular resource and would break in the face of even tiny changes in the schema. The only way to ensure that type identity is genuinely preserved is through the use of check–sum–like computed identifiers.

In the short term, care must be given toward the specification of appropriate global naming conventions to enable a global information infrastructure for biology. In the longer term, efforts by the overall networking community to modify network protocols to support transparent interactions among networked information resources, not just networked hosts, will likely provide a more complete solution [17]. Until such functionality is delivered, those developing federated biological systems should take care to communicate their naming requirements to the appropriate organizations and developers.

Data–Level Integration Across Multiple FOSM Servers

FOSM would support data–level integration across data objects from multiple servers. For example, information on mammalian genes could be published by

several different FOSM servers. Each server would have the local responsibility and autonomy for formatting and publishing its own holdings in the form of trees. Leaves on the trees published by one data server could contain "tokens" that represent the roots of specific data trees available from other servers (Figure 8).
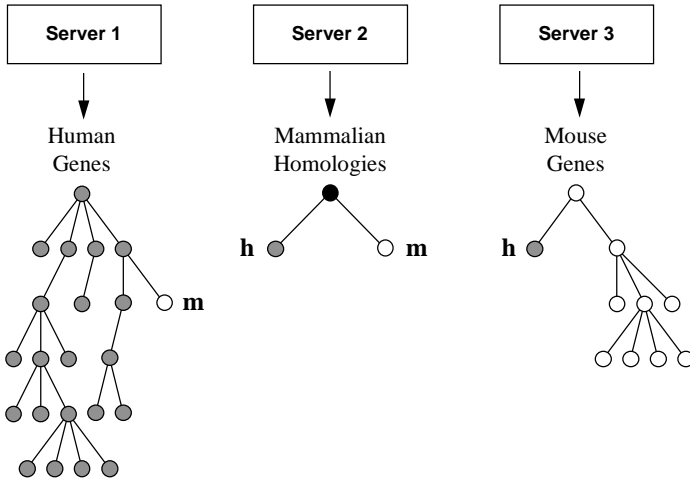

Figure 8. In a FOSM environment, individual data resources would publish their holdings to the network in a standard tree–structure format, according to standard protocols. Nodes marked with "m" and "h" represent sets of tokens that would correspond to the root nodes for mouse–gene and human–gene objects respectively. The inclusion of these external references as leaf nodes indicates that the designer of the local database believes that these external objects are related to the database's primary objects in some role (which is defined in the local database). The decision to include such references, and the populating of them with values, would be the responsibility of the local FOSM server.

Although one might expect data structures describing human and mouse genes to be semantically very similar, or even identical, here it is assumed that they are semantically distinct. Social pressures might exist on data resources to provide physically similar trees for semantically similar objects. However, these pressures would be external to FOSM itself, which only requires that servers adhere to the FOSM tree syntax.

As long as all participating data servers followed these simple guidelines, and providing that a global naming system offered access into a stable, unambiguous naming space for FOSM objects, generic client software could allow users to navigate easily among related data items from different servers.

If data from different servers are combined using the "graft" operator, new trees are produced. For example, Figure 9 shows human–gene objects extended to include mouse genes as attributes, and vice versa. Mammalian–homology data objects could be extended to include both human and mouse genes as attributes.
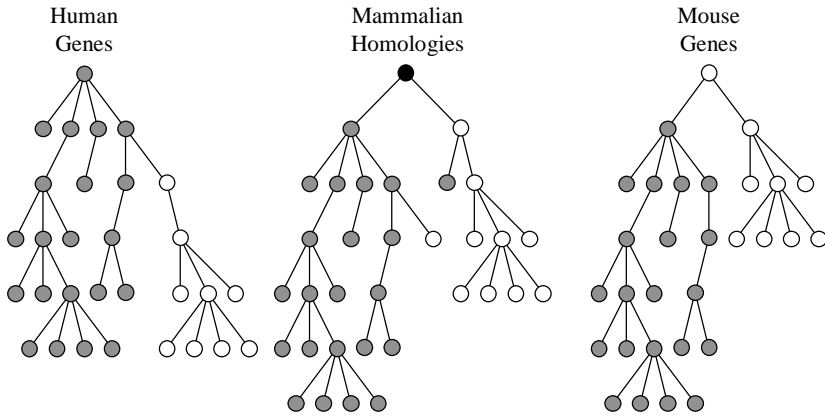
Figure 9. Related data objects may be obtained from different FOSM servers, then grafted together to give new, compound objects. All three of these grafted data objects represent homologies between human and mouse genes, but each from a different perspective: that of the human gene, the mouse gene, and the homology itself. In a DBMS, such inconsistency might be seen as a problem. In a DBPS, the ability to represent diverging viewpoints while maintaining syntactic consistency is a feature.

If data about human genes, mouse genes, and their possible homologous relationships were contained in a single database, obtaining the set of asserted homologous gene pairs would involve a simple, unambiguous join. In the FOSM model, however, individual data providers may offer data objects that reference objects in other databases. Different data providers would be free to publish logically equivalent, but not necessarily content–identical linkages among data objects, as there would be no formal requirement of identity. This freedom to diverge is necessary to allow the information resources to act as scientific literature, which must be able to support differences of opinion

SUMMARY

Biological databases, having survived a crisis of data acquisition, now face a crisis of data integration. Meeting this challenge will require the development of technical and sociological processes that will allow multiple databases to interoperate functionally, while still maintaining much of their individual managerial autonomy. Horizontal partitioning of data, as is the case across some genome data resources, makes the challenge of interoperability especially acute, since achieving good interoperability under these circumstances will require the development of considerable semantic consistency among participating sites.

Computer solutions that, from initial design onwards, are aimed at meeting the specific needs of some particular problem rarely evolve into generic interoperable

systems. Solutions that are based on minimal generic components are more likely to evolve gracefully into specific systems, especially if the specificity is added as layers on top of the underlying generic foundation. Networking architectures have followed this pattern and the evolution of database systems from file–based approaches to cutting edge object–oriented databases show a similar trend.

To be truly useful to the widest range of potential users, on–line genome information systems should be capable of functionally interoperating, at some minimum basic level, with many different information systems (such as nucleotide sequence databases, clinical phenotype information systems, metabolic databases, systematics databases, etc.). Successful interoperation among a large, diverse, and autonomous set of independent data sites can only occur if all sites use equivalent, generic tools to publish their holdings according to common protocols and syntaxes. Gopher and World–Wide Web offer examples of the power in this generic client–server approach to information distribution, but they do not meet all of the needs of those interested in publishing structured data.

An extended data–publishing model, perhaps related to the FOSM concept discussed here, will be required if these needs are to be met in a generic fashion. In such a model, local sites would still be free to manage their data internally according to whatever methods seem best. More importantly, collections of sites would be free to react to scientific needs for convergence upon similar methods for internal data management, as well as upon common consensus data models and semantics for external data publication, while at the same time using generic methods, protocols, and syntaxes for data publication. The adoption of generic client–server methods for data distribution is purely an enabling technology. By not requiring common semantics of anyone, it allows for unrestricted syntactic interoperability. By permitting the adoption of common semantics by some, it facilitates unrestricted semantic interoperability.

The genome community could attain the best of both worlds if they achieve interoperability by sandwiching generic data–distribution methods between converging internal data–management systems on one hand and common public consensus data models and semantics on the other. This would yield a unified conceptual model for genome data, delivered in a system capable of generic interoperation with non–genome resources.

BIBLIOGRAPHY

1. Brooks, FP, Jr: The Mythical Man–month. Addison–Wesley Publishing Company, Reading, MA, 1982.

2. Chorafas, DN, and Steinmann, H: *Solutions for Networked Databases: How to Move from Heterogeneous Structures to Federated Concepts.* Academic Press, Inc., New York, 1993.

3. GeSTeC Directors: Report: NCHGR GeSTeC Director's meeting on genome informatics, 1994. (Available electronically from Johns Hopkins WWW server, http://www.gdb.org/Dan/nchgr/report.html.)

4. Hurson, AR, Bright, MW, and Pakzad, SH (Eds.): *Multidatabase Systems: An Advanced Solution for Global Information Sharing*. IEEE Computer Society Press Los Alamitos, CA, 1994.

5. Lewin, R: DNA databases are swamped. *Science*, 232:1599, 1986.

6. Murray, JC, Buetow, KH, Weber, JL, Ludwigsen, S, Scherpbier–Heddema, T, et al: A comprehensive human linkage map with centimorgan density. *Science*, 265:2049–2054, 1994.

7. Robbins, RJ: Database and computational challenges in the human genome project. *IEEE Engineering in Medicine and Biology Magazine*., 11:25–34, 1992.

8. Robbins, RJ: Genome informatics: Requirements and challenges. In: Lim, HA, Fickett, JW, Cantor, CR, and Robbins, RJ (eds). *Bioinformatics, Supercomputing and Complex Genome Analysis*. World Scientific Publishing Company, Singapore, pp 17–30, 1993.

9. Robbins, RJ: Biological databases: A new scientific literature. *Publishing Research Quarterly*, 10:1–27, 1994.

10. Robbins, RJ: Representing genomic maps in a relational database. In: Suhai, S. (ed). *Computational Methods in Genome Research*. New York: Plenum Publishing Company, New York, pp 85–96, 1994.

11. Robbins, RJ (Ed.): Genome informatics I: Community databases. *Journal of Computational Biology*, 3:173–190, 1994.

12. Robbins, RJ: Genome Informatics: Toward a Federated Information Infrastructure (keynote address). The Third International Conference on Bioinformatics and Genome Research; Tallahassee, Florida; 1–4 June 1994.

13. Schatz, BR, and Hardin, JB: NCSA Mosaic and the World Wide Web: Global hypermedia protocols for the internet. *Science*, 265:895–901, 1994.

14. Sheth, AP, and Larson, JA: Federated databases systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22:183–236, 1990.

15. United States Department of Energy. 1990. *Understanding Our Genetic Inheritance. The U. S. Human Genome Project: The First Five Years.*

16. United States National Academy of Sciences, National Research Council, Commission on Life Sciences, Board on Basic Biology, Committee on Mapping and Sequencing the Human Genome: *Mapping and Sequencing the Human Genome*. National Academy Press, Washington, DC, 1988.

17. United States National Academy of Sciences, National Research Council, Commission on Physical Sciences, Mathematics, and Applications, Computer Science and Telecommunications Board, NRENAISSANCE Committee: *Realizing the Information Future: The Internet and Beyond.* National Academy Press, Washington, DC, 1994.