Pergamon

Comput. Biol. Med. Vol. 26, No. 3, pp. 183–197, 1996

Printed in Great Britain
0010–4825/96 $15.00 + 0.00

0010-4825(95)00053-4

A GRAPH CONCEPTUAL MODEL FOR DEVELOPING HUMAN GENOME CENTER DATABASES

MARK GRAVES, ELLEN R. BERGEMAN and CHARLES B. LAWRENCE

Department of Cell Biology, Baylor College of Medicine, One Baylor Plaza, Houston, TX 77030, U.S.A.

(*Received* 19 *October* 1995)

Abstract—We have developed a representation of genome data which has proven itself useful for describing data at a Human Genome Center. Genomic data have a graph-like structure and representing the concepts and relationships of genetics as a graph simplifies the development of databases for genome laboratories. Graphs are a comfortable communication medium for biologists and computer scientists and graph diagrams assist in the development of databases by facilitating the exchange of expertise. We have tailored a graph language for modeling genomic data and describe our process of using graphs to develop genome databases. Copyright © 1996 Elsevier Science Ltd.

Database development Genome informatics Conceptual modeling Data models Graph-based representation Software engineering Human Genome Project

1. INTRODUCTION

Databases for Human Genome Project researchers must be developed in a way that supports the research effort and blends with the work flow of a Genome Center. These databases should provide the necessary support for capturing and retrieving data, as well as aiding in new research.

Current development of databases relies heavily on the amount of biological knowledge that the database developer possesses. Since many database developers are computer scientists, this can be a problem. We describe a process which brings the biologist into database design and facilitates communication between the biologist and developer, therefore improving the final database.

Conceptual modeling is the process of describing the domain independently of any database management system [1]. Since there is no need for an understanding of databases in the initial phase of conceptual modeling, a biologist is able to describe the domain, using an appropriate scientific vocabulary, in a way that can then be understood by a computer scientist who is responsible for implementing the database. The formal (theoretical) language used for conceptual modeling is called a conceptual model. Conceptual modeling is usually an interactive process and it is simplified when the conceptual model is natural to the domain of genetics and precise enough for the computer scientist to interpret unambiguously.

Database development is a multi-step process, similar to other software development processes. Our database development process consists of conceptual, logical, and physical design phases which, respectively, incorporate into the database design process the concepts of the domain; the data model of the database management system; and implementation decisions which affect the ease and efficiency of accessing data. Understanding how and what to design in each phase of the process increases the likelihood that the final database will meet the future needs of the user as well as immediate needs. Logical and physical design issues are studied heavily in databases research. Conceptual design is more difficult to examine in the abstract because of its

dependence on the domain of interest. We have developed a conceptual model to be used for conceptual design which is tailored to the domain of genetics.

Another advantage of defining conceptual design as a separate stage of development is that the result of the conceptual modeling process can be reused. The result of conceptual modeling is a set of conceptual schemas which can be reused to produce databases in different database management systems, as part of other conceptual schemas, or as a basis for object-oriented application development.

We describe some of the requirements for a genome center database, our conceptual design process for a database, our conceptual model, how to use it for design, and other possible uses for a graph conceptual model.

2. HUMAN GENOME CENTER DATABASE

A genome center database is a central repository for data collection and access within a Human Genome Center. It captures the laboratory data generated by the center, the results published by the center and the data received from other sources. Providing a single database with multiple, project-specific, data entry applications, query capabilities, and other end-user applications supports the work flow of a center. Integration of data within a single database also allows researchers to discern information necessary for their research projects, even when originally discovered by another researcher.

Our conceptual model has been developed while designing several laboratory databases and a central database for a Human Genome Center. Developing one database to capture the data from the several laboratories in a Human Genome Center places an increased burden on conceptual design because the diverse data sources must be captured and integrated within one database. In addition, the design must support smooth transitions as laboratory processes change.

2.1. *Capturing the data flow*

Current genome research is oriented toward developing maps of genome-derived reagents at different levels of resolution. The genome reagents include chromosomes (normal and abnormal), clones of subregions of genomic DNA maintained in a variety of cloning vectors, cDNA clones derived from different tissues, and markers for polymorphic sites in genomes from populations of individuals. Common genome maps include: cytogenetic maps which localize a reagent to a sub-region of a chromosome; genetic maps which order polymorphic genetic markers and can be used to find the approximate chromosomal location of genetic material responsible for an inherited trait; physical maps which provide a physical ordering of reagents relative to a chromosome; and the DNA sequence which is a type of high-resolution physical map. Such maps are being developed for a number of diverse species including human, mouse, Drosophila, *C. elegans*, budding yeast, fusion yeast, Arabidopsis, many prokaryotic genomes, and several agriculturally important plant and animal species.

As these genome projects progress, more attention will be turned to identifying: the location of all genes and the genome components that regulate their expression; the biochemical function of gene products including their role in metabolic and regulatory pathways and interactions with other gene products; the pattern of individual gene expression during development; the subcellular location of gene products; and the evolutionary history of genes. As this data corpus grows, biology-related research will be facilitated by integrating genome and biological data in such a way that it can be explored in a flexible and meaningful way by researchers. Presently, the mapping data from genome project research are organized largely into species-specific and sometimes chromosome-specific databases. There is little integration of information across species boundaries. The most useful integration of data across species occurs through the DNA sequence databases when a sequence from one organism is found to be significantly similar to sequences from other organisms. Yet, currently, even within a single-species database, it is difficult to ask relatively simple questions such as "How many genes on the long arm of chromosome 21 have been sequenced?".

It is difficult to design an effective database to integrate genome and biological data because of the complexity of the data and related concepts. Many significant biological database projects to date have been implemented using relational database technology because the volume of data and the need to assure its safety requires the use of mature, commercial database management systems. This places additional constraints on the design of a genome database because relational databases are optimized for transaction-intensive, record-oriented applications; not applications involving data that have a complex structure. Data models which are more natural for genome research must be used to respond more rapidly to changing requirements, an important attribute in a field that is developing as rapidly as molecular genetics.

While developing a database to address these needs, we have found that a simple and fundamental abstraction based on graph theory forms the basis for naturally representing genome and biological data. This abstraction extends traditional design approaches by improving their capability to represent both complex objects and their interrelationships. Graphs facilitate communication between system designers and system users because they are easy to comprehend. They are also easily modified or extended as system requirements evolve.

2.2. *Extensibility and flexibility*

Because of the rapid progress of genome research, the requirements of a genome center database change frequently. A database may require changes because user needs change or the science changes. Most fields only have to deal with changes in user requirements and do not have to be concerned with changes in the domain. In genetics the most common techniques today were not well known 5 years ago and they may be obsolete in a couple of years. A laboratory database which takes a year to build is outdated before it is brought online.

The uses of the database can change quickly. As new laboratory techniques evolve and new projects are brought into a genome center, the database must be changed to support that research. This requires that the database schema be able to evolve gradually, without rebuilding the database each time. For the applications to change rapidly, the database must be flexible and easily extensible.

The meaning of a genome concept can change drastically with the discovery of new information. An example is the definition of a locus. As large-scale sequencing has progressed, and location information has become more precise, the term locus has remained, though it means different things to different researchers. These different schemas must be captured in the database. Changes in the science also require the database to be flexible and extensible.

Graphs provide flexibility and extensibility to support the requirements of an evolving genome center database. Because graphs are nodes connected by edges, adding a new concept or relationship does not involve redesigning the whole database. It is simple to add a new node to a graph or a new arc coming off a node.

2.3. *Database architecture*

The architecture of our genome center database includes a single shared database which supports multiple applications. A single shared database allows for the integration of data across a genome center while providing data integrity and security and allowing access to all data which are publicly available.

End-user applications include laboratory-specific data entry applications, query interfaces, editor applications, browsers, and report generation applications. The architecture of the database should support rapid development of these applications, allowing developers to create project specific products which access the database. Browsers should provide a way that users can wander through the data, discovering information as necessary. Query interfaces should provide 'canned' or defined query interfaces as well as *ad hoc* query capabilities. In addition, applications such as map editors, contig

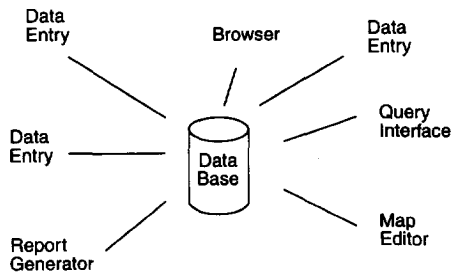

Fig. 1. Schematic diagram of a genome center database.

builders, or graphical display programs should be able to interface easily to the database. A schematic diagram of a general architecture is given in Fig. 1.

2.4. *End user applications*

Every data-intensive application developed to support an experimental process must meet a set of minimum requirements. A user requires that the application be robust, developed rapidly, be extensible, be able to evolve, provide data security, provide an intuitive, familiar user interface, and be run-time efficient. System requirements are that an application should be modular, cost-effective to build and maintain, re-usable, and not be limited to one database management system. The use of object-oriented software engineering methodologies, combined with domain analysis and user-interface design yields an application which can meet the set of user and system requirements.

Object-oriented software engineering provides a framework of specification, analysis, design and implementation. Emphasis is placed on defining objects within the application domain which capture the essence of the requirements. This works well for most applications, but in database development the addition of conceptual modeling to the engineering process provides a means of defining the data along with the application.

Domain analysis is a branch of software engineering where the emphasis is on discovering aspects of the domain which can be used in multiple software applications [2]. Our conceptual model can be used for domain analysis to capture the essential concepts without making premature decisions about the importance of each concept.

User interface prototypes are useful for describing an application concretely. When working prototypes cannot be developed, the interface may be described as a schematic diagram or a non-working user interface. Many user interface development environments provide a palette of standard user interface parts which can be manipulated easily and are useful for developing several variations of a prototype interface that can be developed quickly and displayed to the user for comment. Describing the user interaction with the system is important and is helped by the prototype being functionally complete.

3. CONCEPTUAL DESIGN

A database is developed within the context of a database system [3]. A database system may be thought of as a computerized laboratory notebook. It stores information, makes it available for later access, and keeps it safe and secure. Laboratory notebooks work well but are not intended to capture the vast quantities of data which are generated by large, high-throughput labs with automated equipment. A database system automates information tracking by storing and organizing data in persistent and secure media. A database system consists of minimally a database (a collection of data) and a database management system (a software system which supports access of the data). A database is developed by designing the database, then implementing it in a database management system.

Database systems are especially limited by data representation because they are, by nature, data intensive, although representation of data is a limiting factor in any software system. A good representation models the data in a manner which makes the needed tasks efficient and robust. The representation should not model more than what is necessary, for this is inefficient, and it should not model less than what is necessary, for this is not robust.

A design process for databases has been developed [4] which is useful for genome databases because it separates knowledge of the domain and knowledge of databases. Thus, the design process is amenable to group database development where one individual or individuals is more familiar with the biology and another is more familiar with databases. This situation is the most common dynamic in genome informatics. The database design process can be used in a realistic setting to develop a genome database which captures the data efficiently and can be accessed efficiently.

In the next four subsections, we describe our modifications to the design process, discuss cooperative conceptual design, present background on conceptual models, and work through an example.

3.1. *Database design process*

In our database design process, the three stages of database design are conceptual design, logical design and physical design. After the design process, the database design is implemented in a database management system. The conceptual design captures the concepts and relationships of the domain which are to be modeled within the database. The logical design process translates the conceptual design into the data model supported by the database management system. The physical design process determines how the data are to be stored in the data structures provided by the database management system.

In developing genome databases we have discovered two revisions to database design which make the process more viable. The first revision is to borrow from software engineering the spiral model [5] to replace the waterfall model usually used for database design [4]. The second is to delineate areas of responsibility in cooperative work.

The spiral database design process is more flexible than the waterfall model. The waterfall model in software engineering states that all requirements specification should be completed before any of the analysis, which should be completed before any design, which should be completed before implementation begins. The spiral model is more flexible and proceeds more like: requirements, analysis; requirements, analysis, design; requirements, analysis, design, implementation. In addition, rather than beginning with the requirements for the entire system, often some core part of the system is chosen, and less central subsystems are added later in development. Database design parallels the software engineering process for application development. Rather than create a complete conceptual model, it may be more useful to create a conceptual model of the central concepts and then attempt to design a logical schema which incorporates it. The prolems and inconsistencies noticed in the logical design step can then be corrected in the second conceptual design step. The conceptual schema can be extended and the process repeated. When a central part of the logical schema appears to be stable, physical design can begin.

Each stage of design requires different areas of expertise. Currently, there are few, if any, individuals with the expertise to create a genome database which accurately captures the domain at the current state of the art and also develop a robust and efficient implementation. The database design process usually assumes that the design will be performed by one individual or a homogeneous group. Given the current state of genome informatics, it is more likely that the design will require at least one person from two distinct groups: those with a deep understanding of the biology and those with a solid background in databases. The lack of individuals with solid foundations in both databases and genetics necessitates the cooperation between genome researchers and informaticians.

3.2. *Cooperative conceptual design*

Conceptual design is an interactive process between biologists and informaticians. In general, the primary responsibility for conceptual design lies with the biologist and the responsibility for logical and physical design with the informatician. The biologist examines and understands the logical design enough to ensure that the conceptual design is being correctly captured and the informatician assists the biologist in creating the conceptual schema, possibly suggesting alternative representations for complex constructs. The physical design is the responsibility of the informatician and the biologist does not make any physical design decisions.

Definition of the concepts and relationships to be stored in a database is the responsibility of the domain expert. This person has a better understanding of the domain than an informatician could because the expert works within the domain on a daily basis. Conceptual modeling moves the need for understanding of databases by the domain expert from the initial design phase and allows the expert to define an initial conceptual model for the domain which can be used as a basis for a conceptual schema.

The informatician ensures that the conceptual schema is complete enough to specify the domain knowledge of the database. Initially, the developer may use personal knowledge of the problem to define a partial conceptual schema of the objects to be stored in the database, though this is primarily a mechanism for teaching the domain expert the conceptual modeling process.

Good communication between informatician and domain expert ensures that each domain object is accurately captured. Using the schema as a tool for communication, the developer and domain expert refine the schema to accurately capture all domain concepts and their interactions. Discussion involves defining potential queries, necessary operations, and constraints. Questions may show that additional concepts should be added or that another relationship between concepts is necessary. The schema continues to evolve until both the informatician and biologist are satisfied that the domain has been accurately captured in the conceptual schema.

3.3. *Conceptual models*

A conceptual model is a language for describing the concepts of a domain in a database-independent manner. Choosing a language in which to speak or write is not usually a human option, but in conceptual modeling there are several options from which to choose. For genome data, graphs appear to be the most appropriate representation. Some work has already been done in defining graph languages for databases. We describe the requirements for a conceptual model, existing graph languages, and how graphs have been used in databases.

3.3.1. *Requirements*. For concepts to be represented easily in the schema and form the foundation of a useful database, the conceptual model should be expressive, natural to the domain, easy for the biologist to use and capable of being manipulated effectively in a computer. The conceptual model must be able to express the data in the domain, or the database will not capture all of the data. For a conceptual model to be natural, the constructs in the model should closely correspond to the domain, and small changes in the status of domain objects should be captured by the representation and require only small changes in the representation. A natural model of the domain should be easy for the biologist to use, and it should avoid introducing obscure functionality or cryptic notation which unnecessarily restricts its ease of use. In addition, because the conceptual model will be translated into the logical data model it should have a foundation which is computationally perspicuous.

There are many different conceptual models which could be used to describe a domain, such as a natural language or an object-oriented one. A natural language, such as English, meets most of the requirements for a conceptual model. Natural languages are expressive, natural and easy to use. Unfortunately, natural languages are also vague and imprecise and thus not easily modeled on a computer. Another data model which

could be used for conceptual modeling is an object-oriented language, but object-oriented languages appear to be too behavior-oriented for capturing complex structures.

We do not need the expressiveness of natural language to model genome data, but if a formal language developed for computational linguistics were natural to the domain of genetics and easy for biologists to use, it would make a useful conceptual model. Graph languages have been developed within computational linguistics to effectively represent natural languages on a computer. We have found graph languages to be useful for capturing genome data, since they are expressive, natural for genomics, easy to use and have a firm computational foundation.

3.3.2. *Graph languages.* Graphs capture the structure of genome data. The simple language of graphs also allows a biologist and a computer scientist to communicate, encouraging change and improvement of the domain objects as the development process evolves. A graph conceptual model is a language for describing the concepts (including relationships) of a domain as a labeled, directed graph. The graphs in the language are called conceptual schemas, which are graphs with labeled nodes and edges in which the labeled nodes represent concepts and the labeled edges represent the relationships between nodes. A conceptual schema describes the concepts and relationships in a particular domain. An example conceptual schema is shown later in Fig. 2.

Our graph conceptual model is based on feature structures [6], ψ-types [7], conceptual graphs [8], and terminological description logics [9] which are representation formalisms used in computational linguistics. Entity-relation diagrams [10] are similar to a graph conceptual model, but they require more decisions to be made earily in design. In our conceptual design process, the essential concepts are captured without making decisions about the importance of each concept. Entity-relation diagrams are useful for the logical database design phase and have been used successfully in the design of scientific databases [11, 12]. Graph conceptual models are more useful for conceptual design because they are not as dependent on the relational data model and require fewer decisions to be made early in the design. Using only vertices and edges gives the developer the freedom to capture all concepts without being forced to make premature decisions about relative importance of the concepts.

3.3.3. *Data models.* A conceptual model is a data model used for conceptual design. Data models are a way of describing a formal language for manipulating data. Data models are often used to describe the logical design language of a database, but they can also be used to describe the conceptual design language.

A data model defines the types of data the database stores and the operators which manipulate them. Data models as a theoretical tool were first proposed by Codd [13] to describe the relational database in an abstract, formal manner. A data model describes and restricts how data is to be represented and manipulated. A data model is a formal means of representing and manipulating the structures of a database and is usually defined in terms of type constructors, operators, and integrity constraints [1, 14, 15]. Common data models include the relational data model for relational databases and F-logic [16] or complex objects [17] for object-oriented databases [18].

Other data models include the network, functional (or semantic), and logic data models. The network data model is a physical data model (like hierarchical). It stores data as binary, many-to-one relationships, but is tied to the storage mechanism used. Logic data models are defined using mathematical logic and often use a subset of first-order predicate logic. Semantic data models originated from semantic networks that were developed as tools in artificial intelligence for representing the semantics of natural language. They often have a graph-like structure built from functional arcs or a small set of type constructors.

Graph data models formalize the representation and manipulation of graph data structures for database systems by defining a collection of graph-oriented type constructors and operators which create and access graph data structures. All graph data models have as their foundation the mathematical definition of a graph, which is a collection of

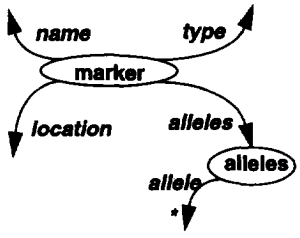

Fig. 2. Basic conceptual schema of a marker.

nodes and edges. The data models are often used to support an application, such as visual querying [19], end-user interfaces [20], hypertext systems [21] or extensible databases [22]. Unlike the relational data model which has one incarnation, there are several extensions to the definition of graphs which form foundations for different graph data models. Most graph data models usually add labels to the nodes or edges or both. Sometimes the basic definition of a graph is changed by allowing nodes to encapsulate graphs [21] and allowing edges to relate to other edges [19]. Related data models include GOOD [23], G+/GraphLog [19], and Hyperlog [21].

3.4. *Domain description*

One application of conceptual modeling is in the development of laboratory databases. Laboratory mapping databases store information, such as markers, probes, clones, primer pairs, PCR conditions and allele information. This should all be captured in a conceptual schema.

There are many types of markers that are useful in finding the location of genes. One useful type is the *dinucleotide repeat marker*. At many locations in the genome, two nucleotide bases, such as CA, are repeated. The number of repeats can vary between people and the pattern of inheritance of these different *alleles* can be examined within a family to estimate how likely a marker is to be inherited along with a gene being searched for. These probabilities can be used to estimate the location of the gene with respect to other markers whose locations are known by converting the probability to a distance using a mapping function [24].

There are various factors which make some dinucleotide repeat markers more useful than others, and discovering useful markers is a fairly involved process, which can be supported by databases capturing the appropriate laboratory data. A central concept in the domain is a marker, and we will describe that concept first.

A marker is defined by a name, a location on a chromosome, a marker type, and a set of alleles. Our basic conceptual schema for a marker is shown in Fig. 2.

We have chosen to represent the set of alleles as a separate graph node because of the complexity of information which may be associated with an allele set (especially when multiple populations are being studied).

Other information is also important for certain types of markers or particular applications. For example, a dinucleotide repeat marker is also defined by the oligonucleotides which surround it and variability measurements of a marker, such as heterozygosity, are very important in choosing markers to be used for linkage analysis. Figure 3 captures this information.

Other factors may also be important. For example, to ensure that the appropriate repeat counts are read from a gel, reference sizes which correspond to commonly available sources of DNA are necessary. While not an integral part of the definition of a marker, reference sizes are required for some applications.

One conceptual schema for representing the relations between markers and the clones which are used to discover them is presented in Fig. 4, which shows the relationship of a PCR marker to a clone which contains it. The clone is related to the marker through the oligonucleotide which occurs in the sequence of the clone and which serves as the PCR

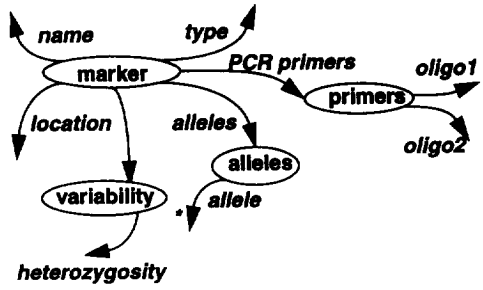

Fig. 3. Conceptual schema of a PCR marker.

primers for the marker. A clone may have many oligonucleotides. This also shows that the graph representing the structure of a concept may not always have a tree structure.

4. DESCRIPTION OF MODEL

A graph conceptual model is a formal system which represents the concepts and relationships of a domain as a graph. We use a form of the graph data model for capturing the concepts and relationships of genetics.

A relation is defined mathematically as a set of n-tuples. Graphs are defined in graph theory as a collection of vertices and edges $G=(V,E)$ [25]. The vertices in a graph refer to genomic objects or n-ary relations and the edges refer to binary relationships (links) between them. We have extended the basic theoretical definition of a graph to create a conceptual model.

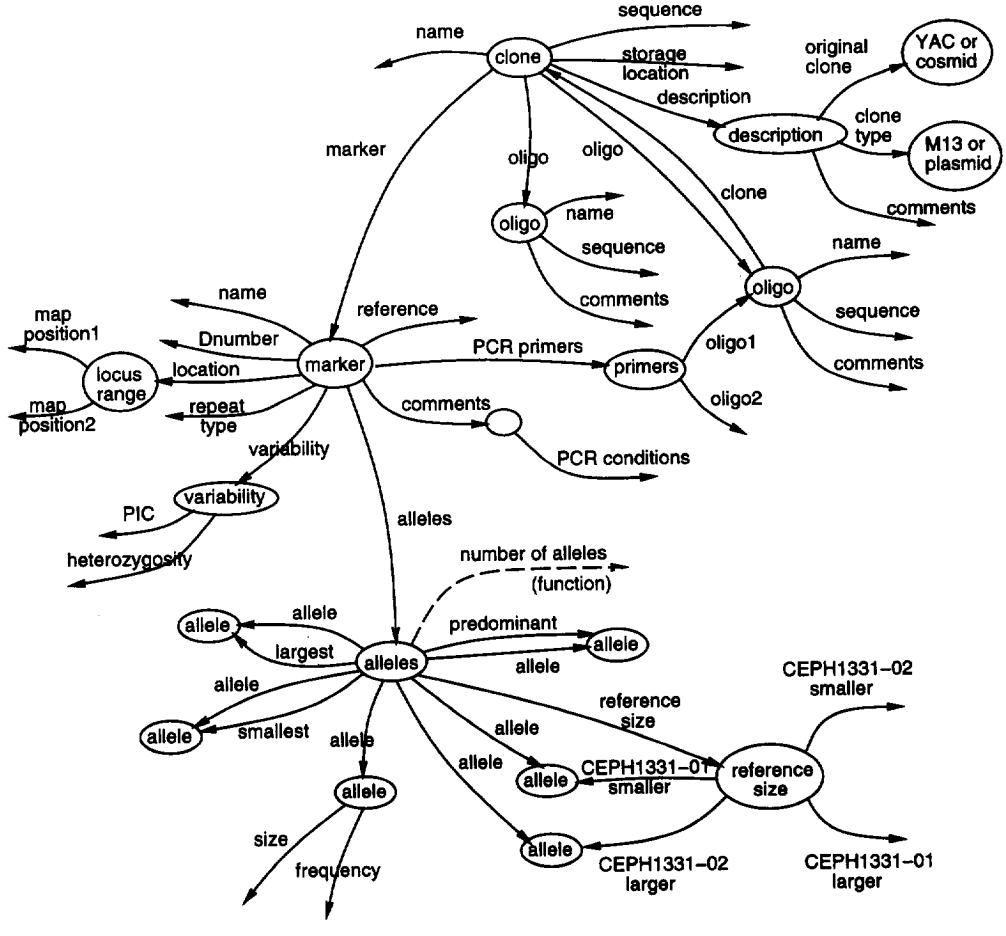

Fig. 4. Conceptual schema of a PCR marker and a clone which contains it.

The graph model which we use to capture genome concepts consists of three extensions to the basic definition of a graph. The first extension is the definition of a concept as a vertex of a graph. Each vertex is labeled with the name of a concept. The type of link between two genomic concepts is also important, thus the second addition to the definition of a graph is a collection of edge labels. The edge labels specify the characteristics of one of the objects, a relation between the two vertices or the role that one of them has with respect to the other. For exampe, valid relations between a cosmid and a YAC would include "hybridizesTo", "generatedFrom", or "sharesSTS", or a plasmid might be in relationship to sequence or location objects. The third extension is the addition of cardinality constraints to the edges.

There are four data types in our graph representation: concepts, edges, edge labels, and cardinalities. A graph is a collection of concepts, edge labels, cardinalities, and edges where:

- concepts are the nodes of the graph and model the simple concepts and n-ary relations of the domain;
- edge labels on the edges describe the relation which holds between the two vertices and are uniquely named;
- cardinalities are either a positive integer or "many"; and
- edges connect two vertices. There can be multiple edges between two vertices (with different edge labels). There is a cardinality for each vertex of an edge. Thus, an edge is a relation between two concepts, one link name, and two cardinalities.

For example, in Fig. 2 a marker has one name, one type, one location and many alleles. The "many" cardinality is denoted by an "*", and the "one" cardinalities are omitted.

There are an additional four extensions to the graph representation which are useful in practice, but which do not describe formally: views, constraints, operations and queries. A graph representation may be broken up into several diagrams, or views, which represent the various configurations in which the links may occur. Constraints include cardinality and type, and operations describe how data are stored and accessed.

A conceptual schema is a collection of conceptual diagrams that define views of the domain. Concepts and links may be shared between diagrams. A conceptual schema should include enough diagrams to show all the links that a concept may have and the various configurations in which the links may occur, but it is not necessary to show each state in the concept's lifecycle. For example, a marker allele may have a name and a frequency in a certain population, or it may have a size and a frequency in a certain population, but alleles do not typically have both names and sizes. When the database is developed, there may be a user view which corresponds exactly to a single (or a few) conceptual diagrams.

A diagram should be augmented with constraints which must hold between the concepts and links. Cardinality constraints are included in our formal model, but other constraints provide additional information. For example, one could state that the length of each cosmid sequence is between 10 K and 100 K bases. Although some work has been done on graphical representation of constraints, we have found it best to describe the constraints as text.

A conceptual model is more than a static description of the data and should also describe the transactions which are to be performed on the database and the constraints which must hold on the data. It is also useful to describe queries which are likely to be asked. We have not found a general way of describing transactions in a manner independent of the database management system used, so we list the operations which may be performed on the data.

Queries are best represented as a corpus of English (natural language) queries which are likely to be asked. They should describe a variety of queries, but do not need to be exhaustive.

5. CONCEPTUAL SCHEMAS

In natural languages, such as English, concepts are captured in sentences or paragraphs. In conceptual modeling, concepts are captured in schemas, which can be combined together to capture more complex concepts. Design is the process of moving from a rough draft to a final product. When writing a paper, one usually begins with an outline which captures the major concepts and describes the flow of the paper. In conceptual modeling, a preliminary schema is used to capture the major concepts and as a basis for discussion. When one is satisfied with the final draft of a paper, it is submitted for publication; in the same way, when one is satisfied with a schema, it is implemented in a database. We describe our approaches to developing a preliminary schema and translating it into a database.

5.1. *Developing a preliminary schema*

The first step in conceptual modeling is to develop a preliminary schema. By creating a preliminary schema, the biologist defines the important concepts in the domain and the relationships that must be captured in the database, without the need to understand any database management system. After the preliminary schema is developed, it can be refined with the help of the informatician to provide a baseline for discussion and understanding of the domain.

The four steps to develop a preliminary schema of a domain are: listing the domain concepts; creating simple sentences which describe relationships in the domain; drawing major concepts as nodes in a graph; adding relationships as edges in a graph.

The first step in creating a graph representation is to list the concepts in the domain. Concepts are real world objects, relationships, and events, such as Experiment, YAC, or Hybridization.

The second step is to list simple sentences containing two domain concepts and a linking word or phrase. Linking phrases are descriptions of the interaction of the two domain concepts which describe the relationship clearly. They include: "has a name", "contains as an element", "hybridizes to", "probes". If the phrase describes a complex relationship, the relationship should be treated as a concept. For example, "YAC hybridizes to an STS" is a complex concept which is important in the experimental physical mapping domain. It is necessary to create the concept of Hybridization and create the sentences "YAC is target in a Hybridization" and "STS is probe in a Hybridization" and "Hybridization has Experimental Evidence" instead of the single sentence "YAC hybridizes to an STS".

After the concepts have been linked using simple sentences, the third step is to select major concepts from the concept list and draw them as nodes in a conceptual schema. The major concepts should be selected based on each concept's relative importance in the domain.

The fourth step is to draw the simple sentences as edges between the two concepts and label the edge with the linking phrase. The direction of the edge should be the same as the sentence. To make the schema diagram more readable, we have chosen to drop "is a" and "has a" parts of the linking phrase from the label but leave the rest of the phrase. This results in some edges having the same label as the 'receiving' node.

After a preliminary schema has been developed, it should be used as a starting point for refining and elucidating the conceptual schema. Refining the schema is a cooperative process between the domain expert who created the preliminary schema and the informatician who will implement the database.

Another graph schema is shown in Fig. 5. The graph describes a filter hybridization experiment for a laboratory process concerned with identifying cosmid clones with regions homologous to cDNA clones. The process consists of a series of hybridization experiments between cDNA and cosmid clones. Within the process, multiple experiments are completed which work together to isolate a relationship between an individual cDNA clone and cosmid clone. Because each experiment can produce multiple results,

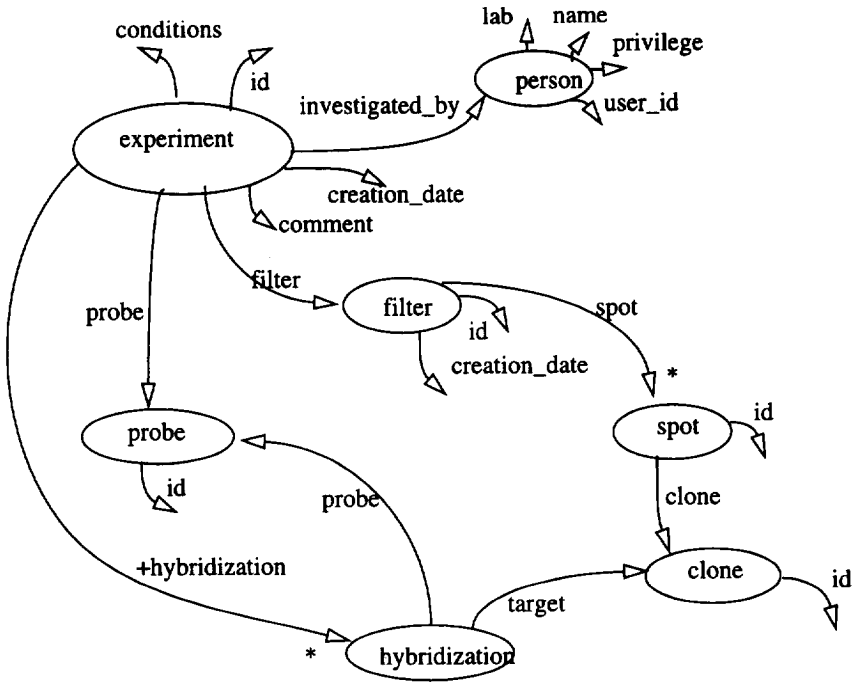

Fig. 5. Conceptual schema for a filter hybridization experiment.

the software should help eliminate redundant experiments, record experimental data, and release final results to other researchers.

5.2. *Translating from a conceptual schema*

We have developed strategies for mapping a graph conceptual schema to a relational or object-oriented schema. To map a graph conceptual schema to a relational database the conceptual schema is translated to an ER diagram, then an ER diagram is transformed to a relational database [4]. To create an object diagram from a conceptual schema, decisions are made about which concepts are central to the design and which relationships are required for an object to exist. An additional diagram is created after each decision and these two diagrams are used with the conceptual schema to create an object diagram, which documents the schema of an object-oriented database.

5.2.1. *Relational database.* To develop a relational database from a graph conceptual schema, the strategy is to make use of an Entity-Relation (ER) diagram [10] as an intermediate result. The conceptual schema is first translated into an ER diagram. The second stage is to develop the relational database from the ER diagram. This translation process is well understood and is clearly presented in [4].

Our conceptual model does not distinguish between concepts which might be considered principal, concepts which have a fundamental dependence on other concepts for their existence and concepts which are not as important as others but merely add descriptive details. These distinctions are made in the ER diagram and form the basis of ER-driven design. An ER diagram describes a database in terms of entities, relationships, and attributes. To translate a graph conceptual schema to an ER diagram, the concepts from the conceptual schema are separated into the three classes:

(1) principal concepts are described as entities;
(2) concepts which do not have their own essential independence in the domain are describe as relationships; and
(3) concepts which add detail are described as attributes.

5.2.2. *Object database*. The conceptual schema captures all the objects in the domain. To create an object model from a conceptual schema, two decisions are made. The first is to decide which concepts are central to the design. The second is to distinguish the relationships which are required for an object to exist. An additional diagram is created after each decision: a domain object diagram and an aggregation diagram, respectively [26]. These two diagrams are used along with the conceptual diagram to create an object model.

A domain object diagram shows the concepts which are to be modeled as objects. After the conceptual schema has been drawn, a decision about the relative importance of each concept is made. Objects are created for the central concepts, and the less important concepts are dropped from the diagram. A domain object diagram shows the relationships between the objects in the domain. One heuristic is to consider all nodes which have no outgoing arcs to be attributes of the node on the other end of the arc. Eliminating the attribute nodes from the graph leaves the domain objects. The domain object diagram is basically a subgraph of the entire conceptual schema. No decisions are made in this step other than eliminating attribute nodes.

From the domain object diagram, an aggregation diagram can be developed. When one object depends upon another for its existence in the domain, the relationshp is captured in an aggregation diagram. Aggregation is a relationship between an object and other objects in the domain, where the description of an object is based on the other objects. The aggregation diagram describes domain objects which play an attribute role in the conceptual diagram but must be modeled as separate objects because they have their own attributes. For example, an experiment has a relationship with a filter, a probe, and an investigator, and an experiment cannot exist in the database without probe and filter. Therefore an experiment is an aggregation of probe and filter. The same experiment could be described without knowing the person who investigated it, so the person is not part of the aggregation.

Aggregation diagrams, domain object diagrams, and conceptual schemas are used together to create the object model, as shown in Fig. 6. Sometimes additional objects may need to be defined as helper objects or subclasses for complex concepts, but this can take place as a normal part of object-oriented development. Our object model is

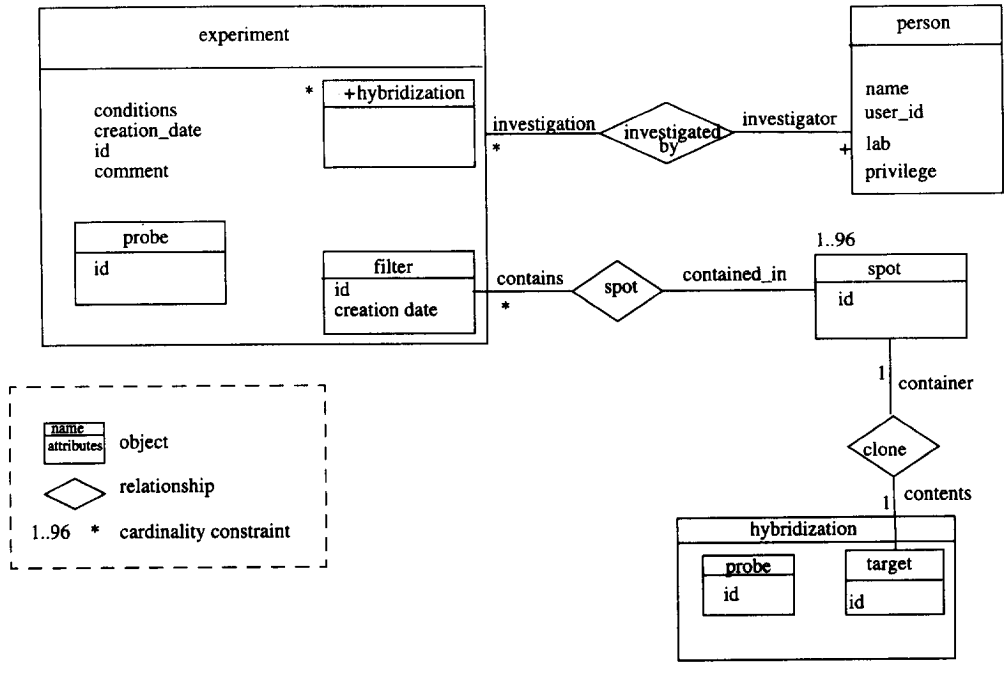

Fig. 6. Object model for filter hybridization experiment.

described using Fusion notation [27]. Concepts of multiplicity in relationships, aggregation of objects and attributes are combined into a single set of diagrams representing all domain objects in the application. The object model is used in the rest of analysis and design.

6. OTHER USES OF CONCEPTUAL MODELS

Conceptual models are not just useful for database design. Other uses are:

(1) development of complex representations (data structures) for application programs;
(2) the logical data model of a new kind of DBMS;
(3) representation language for data exchange.

Databases are not the only software systems that depend upon data representation. All systems are affected by representation to some extent. Designing appropriate data structures are important for any data-intensive application. Many data structures exist, but they are not always sufficient for new application areas. Conceptual modeling can contribute by providing a mechanism to discuss and develop appropriate data structures and to provide a language for domain analysis [2] for software development.

Most advanced database management systems are founded on a data model which at one time was appropriate for conceptual modeling. The relational model helped create a level of abstraction above the files and pointers that most databases were built around. Objects originally arose in artificial intelligence as a mechanism for structuring data and evolved into a useful programming technique. Graphs may also be used as a foundation of a new kind of database management system that stores genome data directly in a more natural framework [28, 29].

Database integration across databases is made more difficult because of differences at the physical and logical design layers. Exchanging data in the common language of a conceptual model can simplify the task by centering the exchange of data around the fundamental concepts of each database.

7. CONCLUSION

We have investigated using graph theory as a foundation for conceptual models and have found that graph conceptual models simplify development of robust genome databases. Three advantages of graph conceptual models are that they

(1) help capture and document laboratory and Genome Center work flow;
(2) facilitate communication between biologists and informaticians; and
(3) support database development which accurately represents biological data.

Graph conceptual models do not appear to be limited to the genome domain, but they are well suited to it because of the graph-like structure of genome data.

Acknowledgements—The authors thank Bob Cottingham, Dan Davison, Wayne Parrott, Joanna Power and Randy Smith for frequent discussions of the ideas in this paper. Thanks also to Jeff Chamberlain at the University of Michigan Human Genome Center, whose laboratory was modeled in the example presented in section 4. This work was supported by the W.M. Keck Center for Computational Biology, The Baylor Human Genome Center funded by the NIH National Center for Human Genome Research, a grant to C.B.L. from the Department of Energy, and a fellowship to M.G. from the National Library of Medicine. M.G. was also supported by an appointment to the Human Genome Postdoctoral Fellowships sponsored by the U.S. Department of Energy, Office of Health and Environmental Research, and administered by the Oak Ridge Institute for Science and Education.

REFERENCES

1. M. Brodie, J. Mylopoulos and J. Schmidt, Eds, *On Conceptual Modeling: Perspectives From Artificial Intelligence, Databases, and Programming Languages.* Springer-Verlag, New York (1984).
2. R. Prieto-Diaz, *Domain Analysis and Software Systems Modeling.* IEEE Press, Los Alamitos, CA (1991).
3. C. J. Date, *An Introduction to Database Systems,* 6th Ed. Addison-Wesley, Reading, MA (1995).
4. T. Teorey, *Database Modeling and Design: Fundamental Principles,* 2nd edition, Morgan Kaufman, San Francisco (1994).

5. B. Boehm, A spiral model of software development and enhancement, *IEEE Computer*, 61–72 (1988).
6. R. T. Kasper and W. C. Rounds, A logical semantics for feature structures. In *Proceedings of the 24th Annual Conference of the Association for Computational Linguistics*, 235–242 (1986).
7. H. Ait-Kaci, A lattice-theoretic approach to computation based on a calculus of partially-ordered type structures. Ph.D. Thesis, Computer and Information Science, University of Pennsylvania, Philadelphia, PA (1984).
8. J. Sowa, *Conceptual Graphs*. Addison-Wesley, Reading, MA (1984).
9. B. Nebel and G. Smolka, Representation and reasoning with attributive descriptions. In *Sorts and Types in Artificial Intelligence*, K. H. Blasius, U. Hedstuck and C. R. Rollinger, Eds, volume 418 of LNAI, pp. 112–139, Springer-Verlag, New York (1990).
10. P. P. Chen, The entity-relationship model: toward a unified view of data, *ACM Transactions on Database Systems* **1**, 9–36 (1976).
11. I.-M. Chen and V. Markowitz, An overview of the Object Protocol Model (OPM) and the OPM Data Management Tools. Tech report LBL-PUB-33706, Lawrence Berkeley Labs, Berkeley, CA (1994).
12. R. Ochs, A relational database model for metabolic information. In *Proceedings of The Third International Conference on Bioinformatics and Genome Research*, H. Lim, Ed., pp. 251–264, World Scientific, Singapore (1996).
13. E. F. Codd, A relational model of data for large shared data banks, *Communications of the ACM* **13**, 377–387 (1970).
14. C. J. Date, *An Introduction to Database Systems*, Volume II. Addison-Wesley, Reading, MA (1983).
15. J. Ullman, *Principles of Database and Knowledge-based Systems*, Volume 1. Computer Science Press, Rockville, MD (1988).
16. M. Kifer and G. Lausen, F-logic: a higher-order language for reasoning about objects, inheritance, and schema. In *Proceedings of the ACM SIGMOD Conference*, pp. 134–146 (1989).
17. R. Hull, A survey of theoretical research on typed complex database objects, *Databases*, J. Paredaens, ed., chapter 5, pp. 193–256, Academic Press, New York (1987).
18. S. B. Zdonik and D. Maier, Eds, *Readings in Object-oriented Database Systems*. Morgan Kaufmann, San Mateo, CA (1990).
19. M. Consens and A. Mendelzon, The G + /GraphLog visual query system. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, H. Garcia-Molina and H. V. Jagadish, Eds, p. 388 (1990).
20. M. Gyssens, J. Paradaens and D. Van Gucht, A graph-oriented object model for database end-user interfaces. In *Proceedings of 1990 ACM SIGMOD Conference on Management of Data*, pp. 24–33 (1990).
21. M. Levene and A. Poulovassilis, The hypernode model and its associated query language. In *Proceedings of Fifth Jerusalem Conference on Information Technology*, pp. 520–530 (1990).
22. M. Graves, Theories and tools for designing application-specific knowledge base data models. Ph.D. Thesis, University of Michigan. University Microfilms, Inc. (1993).
23. M. Gyssens, J. Paradaens and D. Van Gucht, A Graph-oriented object database model. In *Proceedings of ACM Conference on Principles of Database Systems*, pp. 417–424, 1990.
24. J. Ott, *Analysis of Human Genetic Linkage*. Johns Hopkins University Press, Baltimore, Maryland (1991).
25. K. Thulasiraman, *Graphs: Theory and Algorithms*. Wiley, New York (1992).
26. E. R. Bergeman, M. Graves and C. B. Lawrence, Viewing genome data as objects for application development. In *Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology*. AAAI/MIT Press, Cambridge (1995).
27. D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes and P. Jeremaes, *Object-oriented Development, the Fusion Method*. Prentice Hall, Englewood Cliffs, NJ (1994).
28. M. Graves, E. R. Bergeman and C. B. Lawrence, A graph-theoretic data model for genome mapping databases. In *Proceedings of the Hawaii International Conference on System Sciences-28*, Biotechnology Computing Track, pp. 32–41 (1995).
29. M. Graves, E. R. Bergeman and C. B. Lawrence, Graph database systems for genomics, *IEEE Engineering in Medicine and Biology* **11**, 737–745 (special issue on Managing Data for the Human Genome Project) November (1995).

About the Author—MARK GRAVES received a B.S. degree in mathematics and in computer science from the University of Miami, Coral Gables, Florida in 1985, a M.S. in information and computer science from Georgia Institute of Technology, Atlanta, Georgia in 1987, and a Ph.D. degree in electrical engineering and computer science from the University of Michigan, Ann Arbor, Michigan in 1993. He is a postdoctoral associate at the Department of Cell Biology, Baylor College of Medicine, Houston, Texas, where he has been since 1993, and is supported by a Department of Energy Human Genome Distinguished Postdoctoral Fellowship. In 1988–89 he was a software engineer with Intelligent Business Systems, Milford, Connecticut.

About the Author—ELLEN R. BERGEMAN received a B.S. degree in computer science from Texas A & M University, College Station, Texas in 1993. She is currently a software developer at Baylor College of Medicine, Houston, Texas, where she has been since 1993.

About the Author—CHARLES B. LAWRENCE received a B.S. in Life Sciences from M.I.T. in 1970 and a Ph.D. in Biological Chemistry from Washington University School of Medicine (St. Louis) in 1975. Following postdoctoral research studying adenovirus gene expression at the Salk Institute for Biological Sciences, he joined the faculty of the Department of Cell Biology at Baylor College of Medicine in 1979, where he is currently Associate Professor.