Chapter 1
Introduction

In the history of Database Management Systems (DBMSs), interaction between computer and user has played an important role. The incredible power of computers to manipulate data has always been limited by the narrow channel of communication with the user; no matter how fast computers go, users are still responsible for specifying the organization of databases, entering data, making queries, and viewing query results. The success or failure of any DBMS depends greatly on how well its interface allows users to perform these operations.

In order to improve interaction between users and databases, there has been a trend toward interfaces that display information in a visual (as opposed to textual) manner; these interfaces can take advantage of both the high bandwidth available in modern bit-mapped displays, and also the human ability to conceptualize dimensions of space, color, size, texture, and other visual attributes. Visual interfaces are also well suited to the use of direct-manipulation, where the user can modify information through operations on its visual representation (such as mouse clicks).

This thesis examines an important aspect of database interfaces: visualization and direct-manipulation of schemas. Schemas are central to database interfaces because they describe the organization of the database, and thus they are the framework with which the user understands the data. In addition, they are an ideal template for providing context to interface operations such as queries and data specification and display. The core of this thesis is a novel framework for describing two-way mappings between schemas and their visual presentations. This framework provides unprecedented flexibility and customizability in producing visual representations of schemas, and allowing the user to change schemas via direct-manipulation of these representations. This approach has been validated through its use as the foundation for the schema design tool OPOSSUM. OPOSSUM is the front end to an object-oriented scientific database system, and is currently in use by scientists from several disciplines.

1.1 Motivation and Background

1.1.1 Database Interfaces

A DBMS by definition manages data, yet it must also support several kinds of interaction with the user: users specify the schema of the data to be managed, input new instances of the data, and retrieve the data or subsets thereof. There are several traditional text-based approaches to support these types of interaction: Data Definition Languages (DDLs) to describe the database schema, form systems and data manipulation languages (DMLs, such as SQL) to allow input and retrieval of data, and report generators to provide structured output of the data. These traditional approaches have several problems, however. All are text-based, and do not take full advantage of the bandwidth available with modern bit-mapped displays and pointing devices such as mice. Traditional DDLs and DMLs are difficult to learn and use correctly. In addition, textual DDL descriptions are linear, making them less well suited than two or three-dimensional displays for describing very large and/or complex databases. The deficiencies inherent in these approaches have led to the development of visual methods for displaying and/or specifying schemas, queries, and data. These approaches use diagrams, tables, and icons for showing the information; some also permit the user to make changes to the information via direct manipulation of the display. This thesis describes a novel, flexible, extensible method for creating visual

displays of structured information, and allowing the user to edit the information via direct-manipulation of the display; it is discussed primarily in the context of database schemas, though this approach is also applicable to a large extent to queries, data, and other similar information.

1.1.2 Visual Schema Management on the Desk-Top

The visual display and manipulation of schemas is an important area for current database user interfaces. In addition to overcoming many of the limitations of textual DDLs described above, several other factors increase the importance of visual schema management. Novice users in increasing numbers are managing their own DBMSs, requiring more user-friendly ways to specify schemas using computers located on the desk-top. The data in databases are becoming more complex, resulting in schemas that are more difficult to manage. In addition, database graphical user interfaces need visual schema representations to make other database operations (such as querying or browsing) easier; in these interfaces the schema acts as a template, allowing the user to refer to it instead of needing to remember details of the database structure. This thesis describes a formalism that allowed us to create a *Desk-Top Schema Manager* (DTSM). A DTSM is a tool intended for the desk-tops of novice user, supporting flexible and powerful visual representation and manipulation of schemas for the many roles they play.

1.2 Contributions of this Thesis

This research discusses improvements to the interaction between users and databases, specifically examining how schemas are shown to and manipulated by the user. This subject has a major impact on how well users perform all the different interactions with the database, since schemas can be used as templates for query specification and data visualization. It also affects communication of database information between people, because schemas provide an ideal framework for describing database contents.

This thesis provides a framework for declaratively describing the visualization of schemas, allowing schema information to be viewed in many ways. Part of this framework is a novel approach to keeping the layout of a schema in the same general shape as parts of the schema change in size; this approach to layout can be used to support abstraction, whereby schemas can be shown at many levels of detail, with the schema shape similar at all levels. This framework has been implemented as the schema design tool for a scientific DBMS that is currently in use by several groups of scientists.

1.2.1 A Formal Framework for Describing Visual Representations

In order to provide complete flexibility in how information is shown, it is necessary to have a framework describing the process of visualization. This thesis presents an approach to schema visualization that permits declarative descriptions of the abstract data to be displayed, the visual elements of the display, and the mapping between the abstract data and the visual elements. Given such external, declarative descriptions, it is possible to have a generic visualization tool that works with many kinds of data and can display each kind of data in many ways. In addition, the framework allows the visualization of the data to be customized and extended to suit different users and different situations. This flexibility gives users the power to see information in varying ways, and through that to better understand the data. The utility of this framework is demonstrated by its implementation as part of interface to a scientific DBMS, where it is used as the core of tools for visualization

and specification of schemas, data, and queries.

1.2.2 Layout of Abstracted Visualizations

A common problem with viewing data structures is seeing both large scale structure and fine detail. Many data structures are too large to be meaningfully shown in their entirety, regardless of the style of visualization. When the whole is shown, the details are too small to be discerned, and if the details are shown large enough, their context within the whole is not clear. A common approach to this problem is abstraction: allowing a group of details to be seen as a single item. Abstraction has problems of its own: when portions of a display are abstracted, the display becomes more sparse, and when details are shown, the display becomes more crowded. Automatic layout algorithms have been used to maintain display density, but they are unstable, often moving elements from one side of the display to the other. This thesis contains a novel solution to this problem, utilizing user input to determine suggested locations for abstracted and non-abstracted elements of the display, moving elements between these two locations to avoid overcrowdedness or sparsity.

1.3 Organization of this Thesis

This thesis is organized as follows: Chapter 2 discusses the history and importance of interfaces to databases. Chapter 3 describes desk-top schema management. Chapter 4 provides a framework for visualizing database schemas. Chapter 5 gives an overview of the OPOSSUM visualization tool, implementing the framework described in Chapter 4. Chapter 6 presents case studies of the system in use, including a discussion of how abstraction is supported. Chapter 7 describes related work. Finally, Chapter 8 offers conclusions and directions for future work.

Chapter 2

Background and Motivation: Database Interfaces, Past, Present, and Future

2.1 Introduction

Funk & Wagnal's Encyclopedia defines a database as “any collection of data organized for storage in a computer memory and designed for easy access by authorized users.” The history of DBMSs has consisted of work toward these two broad goals: improving computer data storage, and improving (authorized) user access. In this chapter, I explore the area of database interfaces in more detail, discussing their history, current state of the art, and the results of a survey on future directions for database interface research. This history demonstrates the importance of visualization in current and future interfaces. The area of database interface work is quite diverse; I do not attempt to survey all work in the area, rather my intention is to paint the “big picture” of what database interfaces are, where they have been, and where they are going, and why visualization is important.

2.2 A Brief History of Database Interfaces

The two goals of database systems have always been to permit computers to store a wide variety of data, and to allow people to access that data. Access entails several different activities: describing the data to be stored and manipulating the data (entering, querying, browsing, deleting, and outputting). These tasks have been supported with increasing sophistication and facility over the history of databases. This history has been shaped by the interactions of three areas: data models, which describe how a DBMS can organize information, interfaces for data manipulation, and interface technology. Table 2.1 gives an overview of these three areas, discussed in detail in this section.

Time Period	Models	Data Definition	Data Manipulation	Interface Technology
1960s	Hierarchical Network	Declarative Textual	Procedural Textual	Key Punch Line Printer Teletype
early ’70s	Relational	Declarative Textual	Declarative, Textual (SQL)	CRT, Keyboard
late ’70s	Semantic	Diagrammatic (E-R)	Template-based (QBE)	
early ’80s	Deductive	Direct- Manipulation	Browsing Diagrammatic	Bit-mapped Display Mouse
late ’80s- ’90s	Object- Oriented	Icons	Interactive Visualization	3-D pointing devices Virtual Reality Multimedia Voice recognition & synthesis

Table 2.1. Advances in Data Models, Data Definition, Data Manipulation, and Interface Technology

The *user interface* may be defined as that part of the computer system with which a human interacts. The common conception of a user interface is usually limited to windows and menus on a screen, or some kind of textual language, with which users tell the computer what to do. In reality, the interface includes means of input (e.g., tools for specification of schemas), output (e.g., tools for data display), and conceptual tools for the user. In the case of DBMSs, the data model falls into the third category; it is a conceptual view for users of how the database system stores information (which does not necessarily correspond with what the DBMS actually does). While users do not interact with a data model the same way they interact with a pull-down menu, the data model is the framework with which users can think about the data. If a data model can more naturally capture the organization of the real world, it is easier for a designer to create a database that accurately models the data, and that users can understand. In addition, the data model affects the functionality of data manipulation tools with which the user does interact directly. As a result, the history of data models is an important part of the history of database user interfaces, along with input/output tools and interface technology.

The earliest DBMSs, in the 1960s, were mainly based on either the Hierarchical or Network data models (an excellent discussion of these models may be found in the textbook by Date [Dat77]). The Network model is oriented toward complex interrelated data; it captures information in terms of record types and bidirectional links between record types. Consider a database used to keep track of scientists, laboratories, and projects. Each scientist has a name, a specialty, a lab, and a one or more projects. Each lab has a name and an address. Each project has a name, and a lab with which it is associated. Scientists can work on projects outside of their labs. An example of such a database organized using these records and links is shown in Figure 2.1 (it should be noted that Network databases did not display the data as seen in Figure 2.1, rather the figure represents the organization of the data).

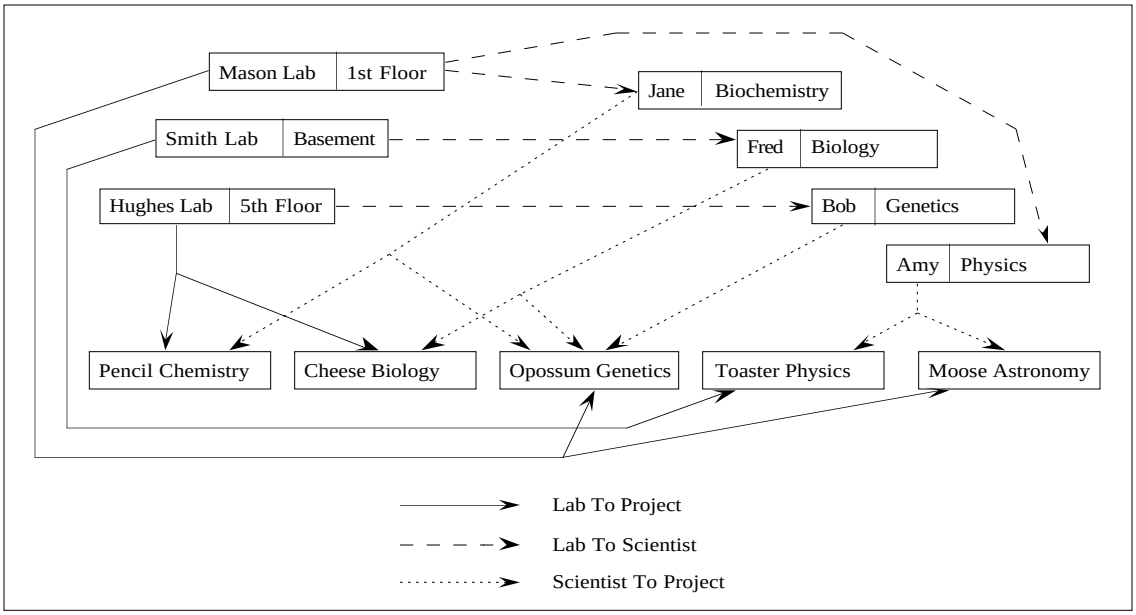

Figure 2.1. A conceptual view of data in the Network model.

This example will be used throughout this section in comparing data models, data definition, and data manipulation. There are three instances of the laboratory record type, shown in the upper left. Four scientists are described in the upper right, and five projects are shown along the bottom. Arrows of different kinds are used to portray the links between the various records. Figure 2.1 shows how the Network model can capture

many-to-many relationships, such as the fact that a scientist can work on many projects and a project can involve many scientists.

The Hierarchical model is similar to the Network model in that it also captures information in terms of record types, and links between record types. The difference is that in a Hierarchical schema, each record type may be the destination of at most one link. Thus Hierarchical schemas have a similar structure to that of trees, with root record types linked to possibly many branches, and each branch being linked to by only one root. The Hierarchical model naturally captures hierarchical data from the real world, though it works less well with more complexly interrelated information. Each record type can only appear once in a hierarchical schema, so the only way to represent many-to-many relationships is by defining additional "logical" schemas that arrange the information differently. For example, in order to capture the relationships between scientists, projects, and labs, a hierarchical database would require two logical schemas, one to show which scientists work in a given lab and the projects on which they work (Figure 2.2), and another to show which projects are part of a given lab and the scientists that work on those projects (Figure 2.3).

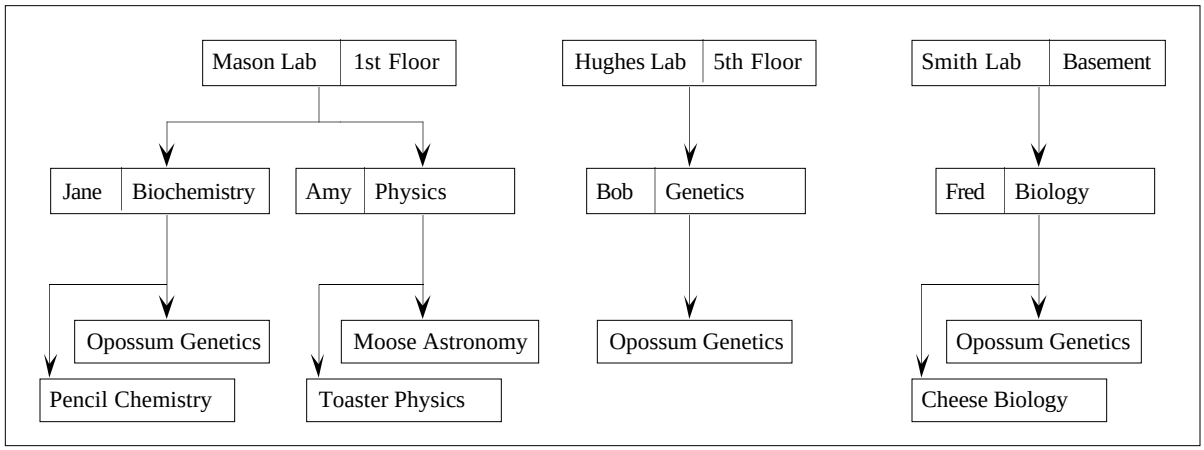

Figure 2.2. A conceptual view of one organization of a Hierarchical database.

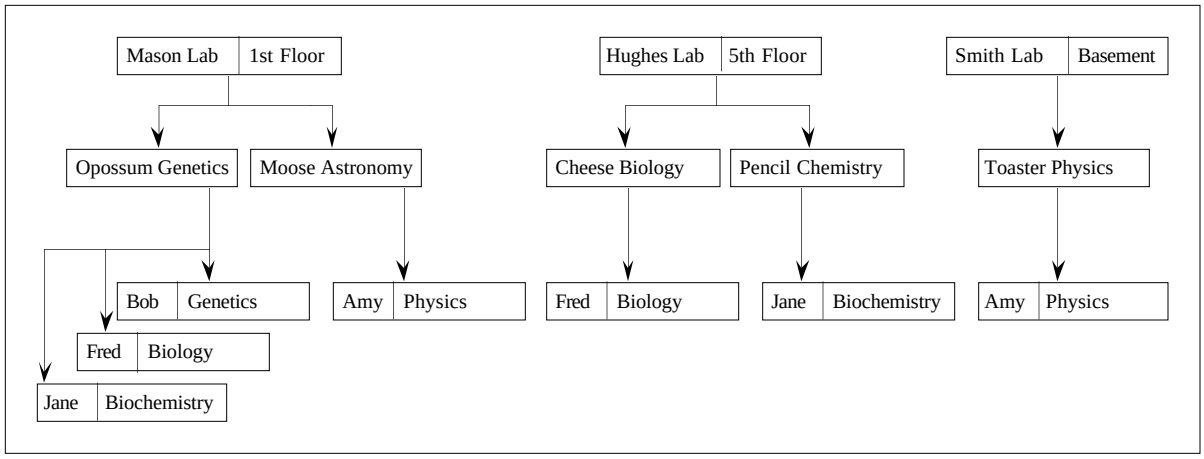

Figure 2.3. A conceptual view of another organization of the same Hierarchical database.

The interface technology of the 1960s provided punch cards, teletypes, and line printers as the major means of interaction between user and computer. In many cases, "users" never came in direct contact with the

computer at all, instead delivering a stack of punch cards to the technicians who ran the computer, returning at a later time to pick up line printer output. At best, users would communicate with the computer via a teletype printing text on paper at speeds of twelve characters per second. Given the limitations of this kind of interaction, database definition and manipulation occurred through text-based languages. An example of the specification of a simple Network model schema appears in Figure 2.4. An example query in the Network model language DBTG is shown in Figure 2.5, asking for the names of scientists whose specialty is Biochemistry. (The syntax for these examples was derived from the textbook by Date, chapters 20 and 22 [Dat77].)

```
Schema Name Is Sci-Lab-Proj.

Record Name is Scientist;
   Location Mode Is System-Default;
   Identifier is ScientistID in Scientist.
   02 Name               ; Type is Character 25.
   02 Specialty          ; Type is Character 30.
   02 ScientistID        ; Type is Fixed Decimal 3.

Record Name is Laboratory;
   Location Mode Is System-Default;
   Identifier is LaboratoryID in Laboratory.
   02 Name               ; Type is Character 30.
   02 LabID              ; Type is Fixed Decimal 3.

Record Name is Project;
   Location Mode Is System-Default;
   Identifier is ProjectID in Project
   02 ProjectName        ; Type is Character 30.
   02 ProjectID          ; Type is Fixed Decimal 3.

Set Name is Sci-Proj;
   Owner is Scientist;
   Member is Project;

Set Name is Lab-Sci;
   Owner in Laboratory;
   Member is Scientist;

Set Name is Lab-Proj;
   Owner is Laboratory;
   Member is Project;
```

Figure 2.4. A Network model schema describing the database on scientists, labs, and projects.

```
      Move "Biochemistry" To Specialty in Scientist.
NEXT. Find Duplicate Scientist.
      If Notfound="YES" GO TO DONE.
      Get Scientist.
      (print Name In Scientist)
      GO TO NEXT.
DONE.
```

Figure 2.5. A DBTG Network model query to find the names of scientists whose specialty is biochemistry.

The complexity of the Network and Hierarchical models is the major source of both their advantages and disadvantages. On the positive side, they allow users to design databases that capture intricate relationships from the real world (the Network more easily than the Hierarchical, however). On the negative side, their complexity limits their usability with respect to data manipulation; the vast majority of systems based on these models only permit manipulation through "record-at-a-time" languages, where the user must write a procedural program that iterates through records in the database. Such an approach requires the user to specify exactly how to achieve the intended manipulation, as seen in Figure 2.5. In addition, the user is responsible for ensuring that the manipulation program runs efficiently. It is not impossible for DBMSs based on these models to provide a more declarative approach to data manipulation, but their complexity makes the task sufficiently difficult that very few systems have done so.

Scientist

Name	Specialty	ScientistID	LabID
Jane	Biochemistry	S37	L1
Bob	Genetics	S39	L2
Amy	Physics	S32	L1
Fred	Biology	S33	L3

Laboratory

Name	LabID	Address
Hughes Lab	L1	1st Floor
Mason Lab	L2	5th Floor
Smith Lab	L3	Basement

Project

ProjectName	ProjectID	LabID
Cheese Biology	P11	L1
Toaster Physics	P12	L3
Pencil Chemistry	P13	L1
Opossum Genetics	P14	L2
Moose Astronomy	P15	L2

WorksOnProj

ScientistID	ProjectID
S32	P12
S33	P11
S37	P13
S33	P14
S39	P14
S32	P15
S37	P14

Figure 2.6. How data can be organized in a Relational database.

A higher-level approach to data manipulation arrived with the 1970s and the introduction of the Relational data model. It is simpler than the Hierarchical and Network models, modelling information only as tables (a.k.a. relations) of each record type. It is not possible in the Relational model to directly represent links

between record types. Instead record types must share a common field to indicate a link. Figure 2.6 shows an example of how the same data might be arranged in a Relational database. Note how there are no explicit links. Instead, for example, the link between scientists and labs is through common values in the "LabID" field.

This simple approach provides a great advantage with respect to data manipulation: the Relational model is sufficiently simple that it possible to define declarative DMLs for it. Given a declarative language, users need only specify what the result of a manipulation should look like (instead of writing a program to generate that result). Users do not have to be concerned with the efficiency of the manipulation because the system undertakes that task. The most common declarative DML for the relational model is the language SQL. Figure 2.7 shows the sample database defined for the Relational model using SQL. Figure 2.8 shows a sample query in SQL.

The Relational model does have a disadvantage, however, in the area of database design. Because links between record types are specified as values common to different tables, a Relational database must undergo a process called *normalization* to ensure consistency and eliminate redundancy. This process is complex and difficult for most average users. On balance, however, the Relational approach is better for the user, as data manipulation is a more common task than database design. The Relational model has proved very successful, and it still dominates the database market. The interfaces to querying and schema design could use further improvement, however, as evidenced by the fact that as late as the mid-1990s, the database group at the University of Wisconsin-Madison considered forming Relational queries and normalizing Relational schemas to be sufficiently difficult areas that they were always included in the qualifying exam for database graduate students.

```
Create Table Scientist        (  Name VarChar(25),
                                 Specialty VarChar(30),
                                 ScientistID Char(3),
                                 WorksInLab Char(3));

Create Table Laboratory       (  Name VarChar(30),
                                 LabID Char(3));

Create Table Project          (  ProjectName VarChar(30),
                                 ProjectID Char(3),
                                 PartOfLab Char(3));

Create Table WorksOnProj      (  ScientistID Char(3),
                                 ProjectID Char(3));
```

Figure 2.7. A Relational schema, specified using SQL statements.

```
Select Name
From Scientist
Where Specialty = "Biochemistry"
```

Figure 2.8. An SQL query asking for the names of scientists whose specialty is biochemistry.

Two major technical advances in the 1970s were the advent of minicomputers, which helped bring users physically closer to their computers, and the increasingly widespread distribution of video display terminals, which permitted a marked improvement in interactivity over teletypes, punch-cards and line printer output.

These two developments allowed people to access their databases directly, without having to go through human intermediaries (though the databases were usually managed by specialized database administrators). Video terminals also helped bring about the first truly user-friendly interfaces for data manipulation: Query By Example (QBE [Zlo75]) and CUPID [MS75]. QBE relies upon the simple, regular structures of the Relational model to present the user with simple one or two row tables for each relation in the database. These tables act as templates into which the user can type qualifications for different attributes. Each column is labelled, so the user need not remember the names of all the attributes. An example of a simple QBE query on the above database appears in Figure 2.9. CUPID takes a similar approach, allowing users to draw lines connecting one-row tables to values indicating selections, or to each other indicating links between tables.

Scientist	Name	Specialty	ScientistID	LabID
	P.	Biochemistry		

Figure 2.9. A QBE query to print the names of scientists whose specialty is Biochemistry.

These templates can also be constructed by the user as a means for defining database organization. The primary advantages of QBE are that: 1) it provides labelled templates, so the user does not need to remember the names of every attribute and can see which attributes are part of the same relation, and 2) it has a two-dimensional syntax, so that the parts of a query can be specified in any order (unlike linear textual query languages, which usually have a linear syntax that requires strict ordering).

The second half of the 1970s saw further improvements in data models. The Hierarchical and Network models are limited because their complexity makes data manipulation difficult, and the Relational model has problems because its simplicity made database design tricky. Ideally some middle ground could be found with the advantages of both. Relational databases were more popular, so the most common approach was to extend the Relational model enough to improve its modelling capabilities, but not so much as to lose its declarative manipulation ability. This trend began with Semantic models in the ’70s, and continued with the Nested-Relational and Object-Relational models in the ’80s and ’90s.

One important aspect of Semantic models is their attempt to more closely represent the real world through the concepts of entities (items in the real world), properties of entities, and relationships between entities. This seems a departure from the Relational model, which is based on sets of values, though the early Semantic systems were built on top of Relational databases. The most significant of the Semantic models (with respect to user interfaces) is the Entity-Relationship (E-R) Model [Che76]. It allows the user to specify database organization in terms of these entities, properties, and relationships, and there is a well defined set of rules for transforming E-R schemas into the Relational, Hierarchical, or Network model schemas. In addition, the E-R model is significant in that it includes a diagrammatic language for specifying these schemas. E-R diagrams consist of rectangles for entity sets, ovals for properties, and diamonds representing relationship sets, with lines connecting the relationships to the related entities, and the entities to their properties. Figure 2.10 shows the sample database as an E-R diagram. The characters “1”, “M”, and “N” next to the relationship-diamonds indicate the cardinality-ratio of each part of the relationship, for example the Works-on-Project relationship has a cardinality ratio of M:N. Figure 2.10 demonstrates the advantages of specifying a schema as an E-R diagram over the SQL and DBTG schema specifications in showing the structure of the database. Like QBE, this two-dimensional language allows users greater flexibility with no syntax-required ordering constraints. Initially, E-R diagrams were created on paper and converted to (mainly) the Relational model by hand, but with the advent

of bit-mapped displays and pointing devices by the early 1980s, tools were developed to allow the user to create E-R diagrams on the screen via direct-manipulation. One early example of this is GUIDE [WK82], which also permits specification of queries through manipulations of the E-R diagram; many other systems followed suit [ACS90, EL85, LSN89, Miu91, SCT91, ZM83]. Thus, it became possible for users to design and manipulate databases in a diagrammatic, non-textual manner. Figure 2.11 shows an example of how a query might be expressed visually using pieces of an E-R diagram.

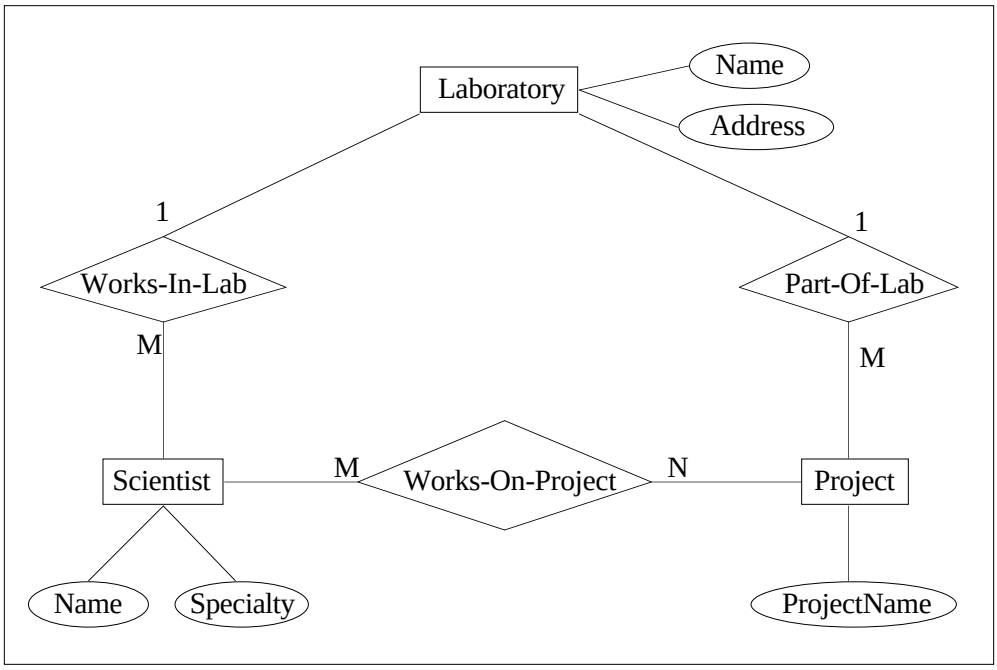

Figure 2.10. A sample E-R diagram.

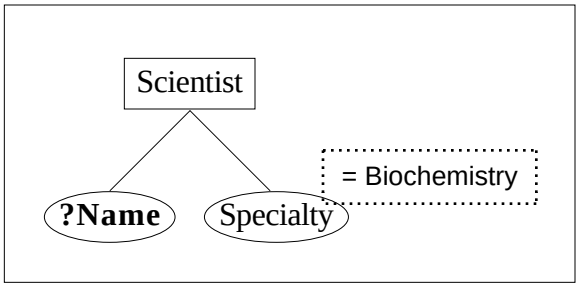

Figure 2.11. A sample query to print the names of scientists whose specialty is Biochemistry, expressed using an E-R diagram fragment.

The 1980s and ’90s saw further developments in data models and interface tools. In the realm of interface tools, Form systems appeared in research prototypes [KN87] and most commercial systems, providing functionality similar to QBE, but with templates that appear like paper forms found in an office, instead of tables. Browsers were developed to allow users to interactively move through the data, combining query and output functions [SK82]. In addition to diagrams, constructs such as icons, tables, and dialog boxes were also used to provide interactive access to schemas and queries. Developments in Artificial Intelligence led to interfaces accepting natural language as input [Nat88], or using it as output to confirm user intentions for

queries specified in another way [EL85].

Data modelling saw two important advances in the 1980s: Object-Oriented (O-O) models and Logic models. Object-Oriented models evolved from Semantic models (they use the term *object class* in the place of *entity set*); the primary distinction (with respect to interfaces) is that O-O models support *data encapsulation*, which permits the database designer to specify the interface to the data, potentially hiding some aspects of the data, and limiting what operations the user can perform on it. To design the database, some O-O DBMSs use E-R schemas [LSN89], or other directed graphs (SNAP [BH86], ENIAM [Cre89], VIMSYS [GWJ91], SKI [KM84], [LCT+90], GOOD [PdB92], and others [YDS+87]). A sample schema in the MOOSE Object-Oriented model [WI93] appears in Figure 2.12, representing object classes as nodes in a directed graph, and Relationships as edges.

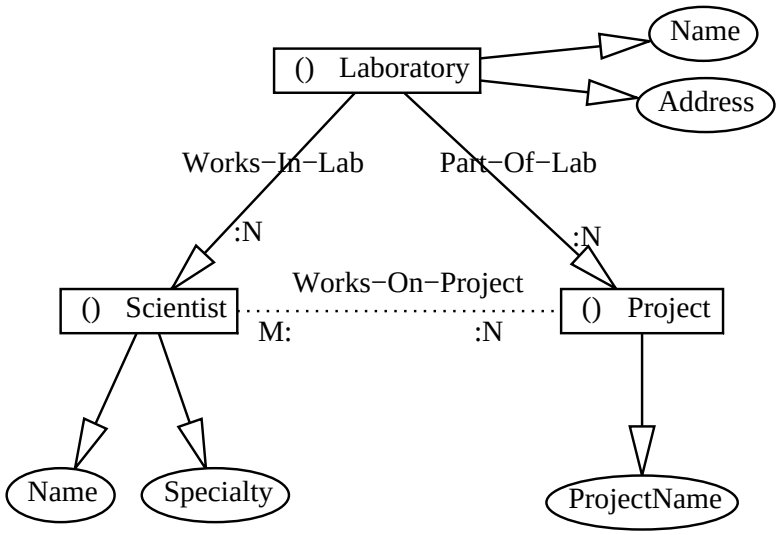

Figure 2.12. A sample schema in the MOOSE object-oriented data model.

The Logic data model captures information as a set of facts, and a set of rules for determining new facts that are implied by existing facts. This approach is very powerful for capturing and retrieving certain types of information, such as those involved in expert systems (e.g. a system to diagnose illnesses, where different sets of symptoms imply different illnesses). Queries are expressed declaratively as questions about facts (e.g., is a fever a symptom of chicken pox?), or partial facts (e.g., what diseases are implied by a high fever?). This approach provides users with the ability to think about data in terms of facts and implications instead of tables and rows, or entities and relationships. Traditionally, logic databases and their queries were specified textually. Some systems provide more advanced interfaces; one example is Hy+ [CM93], which can display data as a very large directed graph of related facts, and permits query specification through drawing a graph which describes a pattern to search for in the data.

One of the most significant technological developments of the late ’70s and early ’80s was the advent of the personal computers and workstations. In terms of price and size, this brought computing to the desk-top of many users. Using such machines, people not only accessed their databases directly, they also managed them, increasing the demand for user-friendly interfaces. The late ’80s and early ’90s brought the introduction of multimedia, 3-D displays and pointing devices, and virtual reality. 3-D displays have appeared in a few products for data visualization, but on the whole the research community is still trying to figure out how best to apply this technology.

The history of database interfaces has been marked by the interaction of several important trends: changes

in data models to improve user access, the development of improved approaches to database definition and data manipulation, changes in the technology of artifacts for human-computer interaction, and changes in the role of the user. It is impossible to definitively determine cause and effect, but there are some clear trends. Data models were initially complex, powerful, and difficult to use. The Relational model marked a move toward simplicity in access, though at the price of increased difficulty in expressing relationships. Since that time, there has been a general trend of increasing model complexity to better capture the real world, while maintaining simplicity of access. In the area of tools for database manipulation, the trend has been toward improved usability and less textual representations, from the procedural queries in the Network model to SQL, to queries based on direct-manipulation of diagrams or natural language input. Part of this trend is due to advances in interface technology; e.g., visual display terminals are necessary to support systems like QBE, and bit-mapped displays and pointing devices are needed to allow direct manipulation of directed graphs. Another part of this trend is driven by changes in data models, for example, with the Network model it was necessary to write queries as procedural programs, whereas the less complicated nature of the Relational model simplified the development of a declarative query language. The change in the role of the user has also affected interfaces: as computers have transformed from rare, expensive items sequestered in machine rooms, to common, inexpensive tools on the desk-tops of average people, the user has seen increasing interaction with and management of the database. This has increased the demand for user-friendly interfaces; the interface technology and data models required for a natural language interface are not necessarily different from that required for an SQL interface, but the former is intended to be easier for average users than the latter.

2.3 Current Database Interface Research

The current state of database interface technology is quite diverse: virtually all the data models described above are available in commercial systems, accessible via many of the data manipulation tools. Most such systems provide some kind of form-based interface for querying, browsing, and data entry, as well as a declarative, textual query language. The state of research on database interfaces is also fairly broad, it can be broken down into the following categories, which will be discussed in this section: interfaces to improve traditional database functions, interfaces for providing new functionality to databases, and studies of database users.

2.3.1 Improving traditional functionality

One of the most important roles of database user-interface research is to make existing database functions easier to use. Recent research in this area includes topics such as querying, schema management, and data display and browsing.

Query interfaces have received the most attention in recent research, probably because forming correct queries is a difficult task, occurring more frequently than database design, and often undertaken by people with less knowledge and experience than database designers. One of the most common approaches to queries is expressing them visually as some kind of directed graph, often based on the schema [BFG94, CS94, CT94, Nor92, PK94]. Some of this work is oriented toward producing a diagrammatic query language as expressive as SQL (or more so). This diagrammatic approach does not require users to remember a complicated textual syntax, though they still must understand the meaning of the query diagrams. A somewhat more ambitious approach is the use of a formal visual language for query and data display specification [Cru93]. Alternatively,

to aid more novice users, other recent work has examined the possibility of mixing natural language or icons with diagrams [Fer94, MPS94, SM94].

Another important area of current research is schema management. Simple diagrammatic definition of schemas, such as E-R diagrams, is well established. Current research examines new types of representations, such as 3-D displays [RK94], and flexible, extensible, user-defined representations such as those described in this thesis. Another topic under study is tools for visually managing and describing versions of schemas [LCC94, Mon94].

Output and browsing of data is also receiving some attention in current research. The major topic is browsing the complex interrelated information found in Object-Oriented databases. Approaches include sorting and clustering tables [SP94] or form-like dialog boxes [dO+94].

2.3.2 Providing new functionality

One important aspect of current database interface research is the development of interfaces that expand the boundaries of database functionality. One branch of research has examined new paradigms for the query process, including incremental queries based on direct manipulation of the results of previous queries [IS94], permitting literal query-by-example through asking the user whether representative tuples match the search criteria [DP94], providing immediate visual feedback about query results as the query qualifications are progressively specified [AS94], and allowing the user to form queries using high-level categories of information instead of the logical schema [HGR94].

Another fairly active branch of research involves database user interface tools, which allow either automatic [MP94, OZL94] or manual [BMP+92, KN92, RKS+90] definition of user interfaces with built-in connections to an underlying database.

Two other topics are worth noting. One is the integration of computer supported cooperative work (CSCW) and databases, breaking the tradition of always hiding the existence of other users. An example is allowing multiple users to navigate a 3-D information display, with the ability to see what data other users are looking at or changing [BM94]. The other is the extension of database interfaces to support manipulation of new kinds of data, such as sound [EV94] and video.

2.3.3 User studies

Study of database users is an area of database interface research that deserves more attention than it has received. In order to determine which research paths to follow, a better understanding is needed of what users need to do, and how they go about it. Examples of this include studies of how users approach the process of schema design [BCD94], evaluating speed and accuracy of various input devices [EV94], and studies of the evaluation process itself [PaQK94].

2.4 A Survey on Future Directions for Database Interface Research

In order to further demonstrate the importance of visualization in database/interface research, this section describes a survey on the subject. In 1989, and again in 1993, Michael Stonebraker surveyed panels of database researchers about which research areas they considered promising, and also which areas they considered the least likely to produce significant results [SAD+93]. The panels' conclusions were somewhat controversial because of their dismissal of several active research areas as unimportant. On the positive side, however, both

panels chose user interfaces as the most promising area. In order to refine the results of these panels with respect to user interfaces, a survey was conducted of the participants at the 2nd International Workshop on User Interfaces to Databases (IDS94) in Ambleside, U.K. The survey's goal was to obtain a snapshot of work in the area, and highlight important subareas and problems. Unlike the surveys by Stonebraker, the results are intended to *describe* rather than *direct;* the sample size is too small and geographical distribution too narrow to mark any topics as the most important, or as unsuitable for further work. In the fall of 1994 this survey was expanded to include both the larger database and human-computer interaction communities. This section presents the results and further analysis of these surveys, suggesting possible directions for database interface research.

2.4.1 The Survey

This survey was initially performed during IDS94 to provide data for a panel discussion. The survey results and the discussion that followed were sufficiently interesting to warrant a follow-up survey by e-mail, providing more comprehensive results. A similar questionaire was later sent to the subscribers of the DBWORLD and Visual-L mailing lists, which deal with database and interface issues respectively. Participants were asked to list the areas of their own research, and outside those areas to vote for three topics at the intersection of Databases and Human-Computer Interaction (DB/HCI) that they consider important. In order to maximize the breadth of responses, the survey did not list specific topics (though general categories were listed to suggest the breadth of the field). The survey also asked respondents to specify if they believe that a formal or empirical approach is needed for their chosen topics. From the IDS workshop, 20 people responded, representing about half of the workshop participants. There were 25 respondents from the DBWORLD list, and 11 from Visual-L.

2.4.2 The Results

Survey respondents voted for about 50 distinct topics within DB/HCI. To organize this large number of topics, I have created a framework of categories, subcategories, and topics. These are listed in the following outline; each is followed by the number of votes it received. Some answers covered two topics, and thus are listed as half votes for each. Votes varied in generality; some votes were for a specific topic, some were for a subcategory, and some were for a category as a whole. In the case of the User Issues category, two votes were cast for that category as a whole, so the total for that category is greater than the sum of the votes of its subcategories. In addition, two votes were cast by the IDS group for formal approaches to all aspects of database/user interface work.

Topic	Subcategory Total	IDS	DBWORLD	Visual-L
Better Interfaces for Traditional DB Tasks				
Data Visualization	16% (18)			
Data Visualization in General		6	6	2
Interfaces to Manage Visualization		1	1	0
Formal Approaches to Visualization		0.5	1	0
New Techniques for Visualization (3-D, VR, etc.)		0.5	0	0

Querying	12%	(13.5)			
Query interfaces in general			3	1	1
Querying based on user's view of the data			1	0	0
Metaphors for data access/querying			1	0	0
Query visualization & direct manipulation			1	2	2
Formal approaches to query interfaces			0.5	0	0
Querying Large Databases			0	1	0
Schema Design/Viewing	7%	(8.5)			
Interfaces for schema design			2.5	2	0
Schema visualization			0.5	0	0
New Techniques for visualization (3-D, VR,etc)			0.5	0	0
Formal approaches to Schema Interfaces			0	1	0
Interfaces for Querying the Schema			0	1	0
Schema Browsing in Distributed Databases			0	1	0
DB administration tasks	4%	(5)			
DB administration			1.5	1	0
DB distribution			0.5	2	0
Better interfaces for new types of DB or new DB functions					
Interfaces for new data types	9%	(10)			
Multimedia			4	0	3
Formalizing unusual data types			0	1	0
Managing GIS data			0	2	0
Interactive Data Visualization	7%	(8)			
Interactive visualization in general			1	1	1
Direct manipulation of data			1	0	1
Interactive 3-D and multimedia views			1	0	1
Visualization of hypertext information			1	0	0
New kinds of databases/interfaces	7%	(8)			
New types/applications of DB in general			2	4	0
Interfaces for heterogeneous multi-DBs			1	0	0
Extending interaction for multi-modal systems			1	0	0
CSCW and DBs	6%	(7)			
Interfaces for cooperative DBs in general			4	0	3
New kinds of query interfaces	2%	(2)			
Justifying results from complex queries			0	0.5	0
Rule-based query formulation			0	0.5	0
Interfaces for fuzzy pattern matching in queries			1	0	0
User Issues					
Evaluating Usability	11%	(12.5)			
Evaluating usability in general			3	4	1
Empirical usability evaluation			1	0	0
Defining evaluation criteria for DB interfaces			1	0	0
Formal usability evaluation			0.5	0	0
User Behavior	5%	(6)			
Models of User Behavior			2	2	1
User Studies			1	0	0
Different Types of User	4%	(5)			
Walk up and use DB interfaces			1	1	0
Formal basis for defining visualizations for different types of users			1	0	1
Maintain functionality for different users across different interfaces			1	0	0
Interface Design Issues					
Database Interface Tools	5%	(6)			
DBs to manage Interface information			0	3	0
Flexible Interface Generators			0	2	0
Facilitating the production and integration of multiple interfaces			1	0	0

Other Issues	5%	(5.5)		
Architecture of distributed interfaces		1	0	0
Standardizing interface behavior		1.5	1	0
Formalize properties of user interfaces in the context of DBMSs		1	0	0
Integration of DB interfaces with those of other software		1	0	0

Table 2.2. Results of the survey on directions in database interface research.

2.4.3 Survey Conclusions

This survey has a sample size too small to state with any certainty the opinions of the DB/HCI community. Nevertheless, it does give a good picture of the area and its popular topics. DB/HCI is a broad area, stretching from mainly HCI topics, such as user studies, to mainly DB topics. In general, research into improving traditional DB tasks is popular, but novel DB types and tasks are also receiving attention. In addition, the database user is being studied to understand how these interfaces should be oriented. Considering specific subcategories, data visualization is the clear favorite, with 26 votes spread between read-only and interactive visualization. After that, the topics are much closer together: with the most popular as Query Interfaces (traditional and new together at 22%), Usability Evaluation (11%), Interfaces to Manage New Data Types (9%), Schema Design and Viewing (7%), and New Types of Databases and Interfaces (7%). Given the small sample size, looking at the numbers for individual topics is not worthwhile, but their breadth gives a good picture of the range of current research. On the whole, visualization of data, queries, and schemas is an important part of research toward better interfaces.

2.5 Summary

This chapter has considered the past, present, and future of database interfaces. It is clear from this discussion that interfaces play an important role in the evolution of databases. It is also clear that in the current era of database interfaces based on direct-manipulation of bit-mapped displays, visual representations of database information are a crucial part of user interfaces. Whether displaying schemas, data, or queries, visualization is critical. This thesis presents a framework with which flexible, extensible visualization tools can be built to help interfaces in these areas.