NPS52--86-014

NAVAL POSTGRADUATE SCHOOL

Monterey, California

A Unified Interface Method
for Interacting with a Database

C. Thomas Wu

January 1986

Approved for public release; distribution unlimited

Prepared for:

Chief of Naval Research
Arlington, VA 22217

FedDocs
D 208.14/2
NPS-52-86-014

NPS 52-80-014

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral R. H. Shumaker
Superintendent

D. A. Schrady
Provost

The work reported herein was supported by Contract from the Office of Naval Research.

Reproduction of all or part of this report is authorized.

This report was prepared by:

VINCENT Y. LUM
Chairman
Department of Computer Science

KNEALE T. MARSHALL
Dean of Information and
Policy Science

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: NPS52-86-014
2. GOVT ACCESSION NO.:
3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): A UNIFIED INTERFACE METHOD FOR INTERACTING WITH A DATABASE
5. TYPE OF REPORT & PERIOD COVERED:
6. PERFORMING ORG. REPORT NUMBER:

7. AUTHOR(s): C. Thomas Wu
8. CONTRACT OR GRANT NUMBER(s):

9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School, Monterey, CA 93943-5000
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 61152N; RR000-01, N0001486WR4E001

11. CONTROLLING OFFICE NAME AND ADDRESS: Chief of Naval Research, Arlington, VA 22217
12. REPORT DATE: January 1986
13. NUMBER OF PAGES:

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office):
15. SECURITY CLASS. (of this report):
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A graphics user interface called GLAD (Graphics Language for Database) is proposed as a unified interface method for a user interaction with a database. GLAD provides a coherent interaction method for all three user interactions with a database: data definition interaction, data manipulation interaction, and program development interaction. In this paper, the features of data manipulation interaction of GLAD are described. Specifically, the method of representing and manipulating generalized/specialized objects and recursively related objects is presented, and the notion of program box (see attached)

which is used for specifying a complex query is introduced.

A Unified Interface Method for Interacting with a Database

by

C. T. Wu

Naval Postgraduate School
Department of Computer Science
Monterey, Ca 93943 USA

The research is partly funded by the Naval Postgraduate School Research Foundation Program.

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

Abstract

A graphics user interface called GLAD (Graphics LAnguage for Database) is proposed as a unified interface method for a user interaction with a database. GLAD provides a coherent interaction method for all three user interactions with a database: data definition interaction, data manipulation interaction, and program development interaction. In this paper, the features of data manipulation interaction of GLAD are described. Specifically, the method of representing and manipulating generalized/specialized objects and recursively related objects is presented, and the notion of program box which is used for specifying a complex query is introduced.

1. Introduction

To attain a wider acceptance and usage, a database management system must accommodate more different types of users. These users range from very sophisticated users, such as database administrators and system designers, to naive users, such as secretaries, clerks, and other non-technical users of database. In this paper, we describe a graphical user interface that accommodates both sophisticated and naive users.

Users of a database management system interact with a database in three different ways. The first is the creation of database[1] via a data definition language. The second is the accessing (i.e. retrieval and update of information) of database via a data manipulation language. And the last is the development of application programs via an embedded host language (i.e. a regular programming language embedded with a data manipulation language). We call these three different user interactions, respectively, *data definition interaction, data manipulation interaction* and *program development interaction.* In a relational database management system INGRES, for example, a query language called QUEL is provided for data manipulation and definition, and an embedded query language called EQUEL/C is provided for developing application programs. To utilize these specialized languages successfully, users must be well versed in computer programming concepts.

In a traditional data processing application of database management system, naive users' interation with a database is usually limited to the accessing of database; thus, the previoulsy proposed "user-friendly" interfaces are concentrated on the data manipulation interaction. A "canned" menu-driven query system, where each choice in the menu is a specific query such as "list all customers with outstanding balance," is a common user interface employed for accessing a database. Problems with the canned menu-driven

[1]By creation of database, we mean the definition of database schema and not the actual loading of data into the database.

query system are that it can only be used by naive users (it is too tedious for sophisticated users) and that it has very limited querying capability (users can only ask what the menu provides).

New proposals for a better data manipulation interface can be classified into three different approaches: a natural language interface, a modified query language interface, and a graphics interface. A natural language interface [BOGU84, CODD74, HEND77, WALTS78] is primarily developed for naive users, and therefore, it may not be suitable for more sophisticated users. Pros and cons for using a natural language interface as a database access method are presented in [PETR76]. The second approach, a modified query language interface [KORT84, MACG85], employs a similar syntax as a regular query language but with more simplified syntax and enriched semantic. It is intended for both sophisticated and naive users. The third approach, a graphics interface [HERO80, LARS84, MCDO74, STON82, SUGI84, WONG82, WU85, ZLOO77], uses graphic objects as a tool for accessing database. This approach is also intended for both types of users.

As a better interface for program development interaction, so called *Fourth Generation Languages* are proposed. These commercial products are advertised as so easy to use that non-technical managers can develop their own application programs by themselves. Although we feel that these Fourth Generation Languages are in general easier to use than embedded host languages, we view these advertisers' claims more of a sales gimmick. A much better approach called *fill-in-the-form* programming for an application program development is reported in [ROWE85]. With this approach, a user develops a complete application by interactively composing a collection of frames, which consists of forms to display/enter data and a menu of operations.

Expressing the conceptual schema of database, once it is designed, in a data definition language is the easiest of all three interactions. And therefore, not much effort has been done to build a good interface for data definition interaction. However, we feel that just providing a syntactic mean to express the conceptual schema of database is not enough. A good user interface for data definition interaction, we believe, should aid users in the design process. GAMBIT [BRAG84] is one such interactive user interface; however, it is intended primarily for database administrators and not for naive users.

A database management system must provide a good user interface for all three interactions to make the system easy to use and learn for both sophisticated and naive users. We should note that simply combining some of the aforementioned proposals is not acceptable, because users must learn three different interaction methods to utilize the database management system. We believe that a single, coherent interation method must be provided for all three interactions in order to gain an acceptance and to increase a usage of database management system by a wider audience.

We believe a graphical interface has the best potential in developing such single, coherent interaction method applicable to all three interactions, and thus, we propose a graphical interface GLAD (Graphics LAnguage for Database). GLAD has two major advantages. First, it provides a single environment where different techniques proposed for different purposes can be integrated. Specifically, in GLAD, techniques proposed in QBE, GUIDE, TIMBER, G-WHIZ, etc are all integrated into a single working whole. Rather than re-inventing a wheel, we provide a framework where different, previously proposed techniques are integrated into one unit, which we believe is a much better approach. Second, GLAD maintains a high degree of data independence. GLAD is not tied to any specific implementation and therefore, it can serve as a front-end to relational or network DBMS. And it is also possible to have a specialized implementation for

GLAD. Since GLAD can be attached to relational or network DBMS, many existing databases can become available (by having GLAD front-end) to all users who learn GLAD.

A major contribution of GLAD to a user interface research is its single, unified interface method for all three interactions. By providing a coherent interface method, GLAD achieves a high degree of ease of learning and using. To our knowledge, there is no other proposal that addresses the issue of a unified interface method for all three user interactions. Specific contributions of this paper are the presentation of simple graphical representations for different types of generalized/specialized and recursively related objects and the introduction of program box concept. GLAD is the first graphical interface that can represent these objects and allow users to manipulate them in a simple fashion.

The paper is organized as follows. The characteristics of a good user interface for data manipulation and an overall description of a data manipulation interaction via GLAD are given in the next section. Section 3 illustrates a data manipulation interaction via GLAD by going through a sample session. Section 4 and 5 discuss the representation and manipulation of generalized/specialized objects and recursively related objects, respectively. Section 6 introduces the notion of program box which is used as an aid in formulating complex queries. And finally, Section 7 concludes the paper.

2. Data Manipulation Via GLAD

A good user interface for data manipulation interaction capable of supporting both sophisticated and naive users must have the following characteristics:

(1) It must be descriptive. It must show what kinds of data (employee, department, equipment, etc) are stored in the database and how they are related to each other.

We call such representation of data and their relationships *database schema*. Also, it should provide a help (i.e. describe more about the database) if requested by users.

(2) It must be powerful. Users must be able to express a complex query by using it.

(3) It must be easy to learn. Naive and uninitiated users should be able to master the interaction method quickly and start accessing the database with a short learning period.

(4) It must be easy to use. It must be easy to use so users rarely make erroneous queries and can formulate complex queries simply and quickly. Also, users should be able to specify a query in different ways; they should not be forced to remember a particular way to pose a query.

Query languages of currently available database management systems lack in all characteristics but (2).

A review of previously proposed graphics user interfaces [HERO80, LARS84, MCDO74, STON82, SUGI84, WONG82, ZLOO77] by using the criteria listed above as a yardstick is given in [WU85]. We shall simply note here that none of them attains all four characteristics satisfactorily. A graphics user interface for data manipulation interaction that we describe here can be viewed as a synthesis of previoulsy proposed graphics user interfaces. By incorporating their good features and eliminating their weak points, we believe that GLAD has achieved a higher degree of descriptivenes, ease of learning and using, and power.

For ease of reference, we shall call the part of GLAD that deals with the data manipulation interaction GLAD *DMC* (*D*ata *M*anipulation *C*omponent). In the following, we describe the salient features of GLAD DMC. These features are categorized into four characteristics mentioned above.

2.1. GLAD DMC is descriptive

GLAD DMC displays a diagram of the database schema. This GLAD diagram is semantically rich, which means that it is capable of capturing a real world semantics (how the information stored in the database are related to each other) naturally and precisely. Conventional data models, such as relational, network, and hierarchical data models are semantically poor compared to new data models, such as SDM, TAXIS, E/R, etc. A GLAD diagram is applicable to many semantic data models[2] , because it provides an elegant diagrammatic representation of real world abstraction concepts those semantic data models employ. The abstraction concepts we are dealing with here are: aggregation, generalization, and classification. In the following, we discuss each of them and show how each is represented in a GLAD diagram.

Aggregation:

An object[3] is an *aggregation* of (sub)objects. For example, a student object could be an aggregation of name, address, ssno, gpa, and dept (sub)objects. An aggregate object is represented as a rectangle in a GLAD diagram, see Figure 2.1a. A user can see the sub(objects) of an aggregate object by expanding it, see Figure 2.1b. Notice that the dept object is not shown in Figure 2.1b because it is not an atomic object. Only first four objects are *atomic*; that is, they are an aggregation of exactly one system-defined or a user-defined *base* object (string, number, enumeration, subrange and boolean). Since the dept object is an non-atomic aggregate object, say, an aggregation of name, set of students, set of courses, and school, it was not shown in Figure 2.1b. Each non-atomic object in the database is displayed[4] , and an association between objects is represented as a line between these objects, see Figure 2.1c. Notice that the dept object is shown as a

[2]Although we feel that a GLAD diagram is applicable to almost all semantic models, there may be some data models that we are unaware of or do not fully understand. Therefore, we shall refrain ourselves from making such claim until further study is made.

[3]We do not define the term object here; we intuitively view an object as a "thing" (both tangible and intangible) that exists.

[4]A database designer can modify it so a non-atomic object is not displayed.

separate rectangle. Notice also that an association is not labelled. We discuss the issue of labelling an association in Section 5.

Generalization:

Individual objects, such as faculty, secretary, and technician, can be grouped together to form a *generalized* object, say, employee. Faculty, secretary, and technician objects are called *specialized* objects of employee. Generalization abstraction can be characterized as an IS-A relationship (faculty IS-A emloyee, etc.).

A generalized object is represented in a GLAD diagram as a nested rectangle, see Figure 2.2a. A user can expand the nested rectangle and view the specialized objects, see Figure 2.2b. These specialized objects can themselves be the generalized objects of yet further specialized objects. A faculty object, for example, can be a generalized object of full, associate, and assistant professors. Graphical representations of different types of generalized/specialized objects are given in the next section.

We have already mentioned in the previous subsection that an association between objects is represented as a solid line in a GLAD diagram. When one or both of the associated objects are specialized objects, a dotted line is used. Figure 2.3 shows how the dotted lines are used in a GLAD diagram.

An aggregate object can have a disjunctive association, which means that an object can have either one (sub)object or another. We use a small circle to show a disjunctive association. Figure 2.4 says that an equipment can belong either to a department or to a school.

Classification:

Each data item stored in a database is information about some object. For example, data item [bruce springsteen, n. j., 123-45-6789, 3.2, music] is an information

about student. and it is *classified* as a student object. We call each data item of an object a *member* of that object.

2.2. GLAD DMC is easy to learn

The number of concepts a user has to learn in order to access a database via GLAD DMC is few, and the interaction method is consistent throughout the interface. Circle, regular and nested rectangles[5] , and solid and dotted lines are the only concepts that a user need learn to understand the GLAD diagrams. Morever, the HELP and DESCRIBE commands are always available to a user. The interaction method is also very straightforward. A user will select the operation by first moving the mouse to the desired operation and then clicking the mouse button. A user can also select the operation by pressing the corresponding function key or by typing the operation name. After selecting a desired operation, a user must select an argument(s) (by moving the mouse to the desired argument and clicking the mouse button) that the chosen operation will be executed on. For example, a user will first select an operation LIST MEMBER and then select an argument **subject** to list all subject matters stored in the database. The result is shown in Figure 2.5. The complete result cannot fit into a window and therefore, only seven subject matters are displayed in the window. A user can browse the data by moving the small square vertically or by positioning the mouse at the arrow and pressing the mouse button. In GLAD DMC, a user may reverse the sequence of interaction, that is, a user selects argument(s) first and then selects the operation. This flexibility helps users learn the interaction method easily.

For more complex retrieval operations, a QBE-like interface is used. The notable difference between ours and the original QBE is that ours does not need the use of variables for queries involving more than one objects (relations in the original QBE).

[5] The third type, repeated rectangle, is used to represent a recursively related ojbects.

The results of various psychological studies that demonstrate QBE's ease of learning and using are well documented in [THOM75]. By avoiding the use of variables and by displaying the database schema, we believe ours is easier to learn and use.

2.3. GLAD DMC is powerful

GLAD DMC's power to express a complex query is a direct consequence of adopting a QBE-like interface for specifying a query. QBE is relationally complete, which guarantees that it is capable of expressing any query that can be expressed in relational algrebra or relational calculus. This relational completeness is a general criteria used to prove the expressive power of a query language. Because GLAD DMC has inherited all the querying capabilities of QBE, GLAD DMC is also relationally complete.

2.4. GLAD DMC is easy to use

GLAD DMC's flexibility of allowing a user to formulate a query in different ways and in incremental, piece-by-piece fashion makes it easy to use. By allowing a user to pose a query in different ways, it appeals to the wider range of users. If there is only one way to pose a query, it probably would not appeal to all users, because the allowed query specification may be too difficult to naive users or may be too cumbersome for sophisticated users. Each user has his own preferred way of specifying a query, and the user interface should allow different ways of specifying the same query as much as possible.

Incremental query specification would also appeal to the wide range of users. Rather than writing the specification for a complete query and executing it, GLAD DMC users can retrieve the result of the complete query in a piece-by-piece, incremental manner. A user formulate the complete query by first (mentally) decomposing the query into smaller subqueries. After specifying each subquery correctly, he combines them to

retrieve the result for the complete query. In this way, a user can formulate a complex query in an expedient manner with very few errors. A sample session in the following section will illustrate the sequence of incremental querying.

The ability to browse the result also improves the ease of use. The result may actually contain more than what a user really wanted. Instead of reformulating the query, the user can simply browse through the result and get information. If the user wants to save the desired result, then he can delete the unwanted records while browsing through the retrieved result.

3. Sample Session

We now illustrate the features of GLAD by going through a sample session. Specifically, we show how the user can retrieve the answer to "List all classes that deal with the subject of probability and that are taken this quarter (Fall '85) by the students who are majoring in Computer Science and whose gpa's are better than 3.5."

Figure 3.1 is the GLAD diagram for a university database schema. Figure 3.2a shows the hierarchical structure of GLAD commands.[6] Associated with each command is a positive integer which corresponds to the function key number. A user can select the command either by using the mouse, by pressing the corresponding function key, or by typing the operation name. The top level menu is shown in Figure 3.2b. The menu will be placed on one part of a screen.[7]

To get the answer to the query, the user enters the QUERY mode by selecting the QUERY command from the second level menu. The third level menu is now displayed on the screen. He decomposes the query into two part: one that lists all courses that deal

[6]Not all commands are shown here and some of the commands shown here will not be discussed in this paper.

[7]Exact position is not determined yet. We anticipate to position it either on the top or on the right hand side of a screen.

with probability and another that lists all classes taken in the Fall '85 quarter by the Computer Science students whose gpa's are better than 3.5.[8] The user SPECIFYs the subject, see Figure 3.3. the course, see Figure 3.4. and the line connecting the two to get the result of the first subquery. An alternative method, where the user specifies every condition within the course object, is shown in Figure 3.5. The subject column is appended to the course object when the user selects the SHOW CONNECTED OBJECT command.

To actually get(retrieve) this intermediate result. the user issues the CREATE RESULT command, see Figure 3.6. Notice that the result icon and the subject and course objects are shaded identically. He can now request SHOW RESULT to verify that the intermediate result is what he wants. Similar to the LIST MEMBER command, the SHOW RESULT command will allow users to browse the intermediate result. The SHOW RESULT command gives an immediate feedback to users enabling the early detection of erroneous query specification. As users become more proficient, they can bypass the SHOW RESULT command. Having these two separate commands, GLAD can help naive users (if asked) by providing an immediate feedback, but it will not force such help on sophisticated users. By going through a similar query specification, the user creates the result for the second subquery, see Figure 3.7.

When a user formulates many subqueries, he may forget what the (sub)results were about. He may prompt GLAD to describe them. For example, the description of

```
contains class information where
dept.name = 'Computer Science' and
student.gpa > 3.5
```

will appear under **result 2** if the user requests to DESCRIBE **result 2**. A user may change the environment by executing the SETUP command so the description of a result

is displayed automatically.

Finally, the user retrieves the complete answer by combining two (sub)results by executing the COMBINE RESULTS command, see Figure 3.8, and selecting the association between them. If the user desires to have a permanent copy of the answer, he can save the answer by selecting SAVE RESULT.

4. Representation and Manipulation of Generalized/Specialized Objects

In this section, we describe the graphical representations for different types of generalized/specialized objects. We start with some definitions. Let G be a generalized object and $\Gamma(G) = \{ S_1, S_2, \cdots, S_n \}$ where each $S_i = \{ S_{i_1}, S_{i_2}, \cdots, S_{i_m} \}$. S_i is called *category* and S_{i_j} *specialized object of G in category* S_i. For example, we may have

Γ(employee) = { {engineering, business}, {faculty, secretary, technician}, {male, female} }

for an employee object.

In [WU85], the following assumptions, or conditions, are made:

(a) $0 \leqslant n \leqslant 1$ (note: n = 0 means G is not a generalized object)

(b) $S_{i_j} \cap S_{i_k} = \phi$ for $j \neq k$, $1 \leqslant i \leqslant n$, $1 \leqslant j,k \leqslant m$

(c) $\bigcup_{j=1}^{m} S_{i_j} = G$ for $1 \leqslant i \leqslant n$.

The above assumptions are made to simplify the preliminary design of GLAD DMC. The first assumption says that an object can have at most one category. The second and third assumptions together say that when an object has a category, a member of the (generalized) object must be a member of exactly one specialized object. Above

[8]He may, of course, decompose in other ways.

Γ(employee) violates condition (a). Suppose an employee can be a faculty and a technician at the same time, then it violates condition (b). And suppose there is an employee who is not a faculty, secretary, or technician (say, he is an administrator), then it violates condition (c).

In the following, we show how these assumptions can be removed and yet have elegant and effective graphical representations of generalized/specialized objects.

4.1. Specialization in more than one category

One possible way to graphically represent Γ(employee) given above is shown in Figure 4.1. This method of showing all specialized objects in every category is not acceptable in the following accounts: (1) it cannot be extended nicely to handle the situation where the other two conditions are removed, (2) it contradicts our general philosophy of good user interface, which is to show the minimal amount of information (i. e. highest level of abstraction) initially and to show the detail only when requested by users, (3) it becomes unwieldy when the number of categories is large, and (4) it does not lend itself to a easy formulation of query which involves more than one category.

The method we adopt here is based on the fact that specialized objects inherit all attributes of the generalized object. When a user request to expand the generalized object G, which has more than one category, GLAD DMC will prompt the user to select which category to expand. When a user issue the EXPAND operation on the employee object described above, a pop-up menu will appear on the screen as shown in Figure 4.2. A user has an option of expanding the employee object in any one of the categories listed. If a user selects the job category, then the screen will show the employee object expanded along the job category, see Figure 4.3. Since the specialized objects inherit attributes from the generalized object, faculty, secretary, and technician objects all have two categories, i.e. sex and school categories. So, if a user requests to EXPAND the

secretary object in Figure 4.3, then the situation shown in Figure 4.4 will result. Notice that with this method, users have a wide variety of expanding a generalized object: they expand the object in a way that fits most naturally to a query that they are formulating.

4.2. Specialized objects are not disjoint

When specialized objects are not disjoint, then there is a member x of G such that $x \in S_i$ for more than one i. In other words, specialized objects overlap (if we view object as a set). Figure 4.5 depicts the situation where there exists a member x of G such that $x \epsilon S_1$ and $x \epsilon S_2$. Notice that the LIST MEMBER operation executed on S_1 and on the overlapped region of S_1 and S_2 will result differently. Examples in Figure 4.6 show the versatility of this approach.

4.3. Union of specialized objects is not equal to a generalized object

If $\bigcup_{j=1}^{m} S_{i_j} \neq G$, then there is a member x of G such that x is not a member of any of S_{i_j}.

This situation is graphically depicted by creating a unnamed rectangle, see Figure 4.7. This unnamed rectangle clearly shows that there is a member who does not belong to any specialized object. This approach fits perfectly with other solutions given in two previous subsections and the overall design of GLAD DMC.

5. Representation and Manipulation of Recursively Related Objects

We call an object *recursively related* if there is an association between the members of the same object, and such association is called *recursive*. For example, there may be associations such as "married" and "parent-of" between the members of the people object. Recursively related objects are graphically represented as a repeated rectangle, see Figure 5.1. Notice that in Figure 5.1, there are two associations on the people object,

and they are not labelled. Associations are not labelled in a GLAD diagram even if there are more than one association between the same objects. Then, how does a user know which association is which?

Automatic labelling of associations will cause the cluttering of a screen and therefore, we decided not to label them.[9] Normally there is only one association between objects, and once a user asks GLAD to EXPAND it (or DESCRIBE it if more detailed explanation is desired), he can usually remember its meaning for the duration of a on-line session. However, if a user is a very forgetful person, he may end up asking GLAD repeatedly for the expansion of the same association. Also, if there are more than one association between objects, it may be difficult to keep track of which association is which. To aid users in these situations, GLAD DMC allows users to label associations (in fact, they can label almost anything in a GLAD diagram). So there are two levels of flexibility. First, a user has a choice of labelling an association or not. And second, he can label it any way he wants to once he decides to label it. He can use a word(s) that helps him remember the semantic of an association best. This may be viewed as a another form of semantic relativism.

The method for the manipulation of recursively related objects in GLAD is modelled after G-WHIZ [HEIL85]. We modified their method so that it integrates nicely into the GLAD framework. Recursive query is formulated in GLAD DMC by specifying the beginning (root) member of the hierarchy,[10] the direction and depth of the traversal, and the regular attribute qualifications. For example, to list the grandfathers of 'Walter', a user puts the specification as shown in Figure 5.2a and then selects the appropriate recursive association (in this case, it is parent-of). The term BU3 in Figure 5.2a stands for *B*egin hierarchy *U*pward for *3* levels. The downward traversal for *n* levels

[9]By default, there is no label; but users can change this default so a label always appears on a screen.

[10]Expanding a recursively related object from one starting point will end up in a hierarchy of members.

is expressed as BDn. The BU operation is unique to GLAD: in G-WHIZ, only BD is available. The specification 'Walter' under the name column states that the traversal begins from 'Walter'. The END specification under the sex column states that the condition '=male' applies to the members in the last (i.e. end) level of the hierarchy. The END specification under the people column states that only the members in the last level of the hierarchy are to be retrieved. The other two options are BEG and ALL. These options are used to specify the stated conditions are applied to the members of *which* levels of a hierarchy. The specification in Figure 5.2b retrieves all grandfathers of the person named 'Pat' who is a female. When the desgination of which members that the stated conditions are applied to becomes more complex, then the PATH option is used. The value of PATH designates the specific member of a hierarchy. For example, PATH = 2.1 of designates the first member at the level 3 who is a parent of the second member at the level 2. If the hierarchy is an ordered tree, and the first member is father, while the second is mother, then the member at PATH = 2.1 is Walter's maternal grandfather.

6. Program Box

It is helpful to have program control structures to formulate a complex query, because it may not be possible to specify some complex queries just by the features of the GLAD DMC discussed so far. To provide a further assistance to users in formulating a complex query, GLAD supports the feature called *program box*. It is an extension of the condition box employed in the QBE system. A program box is a window on a screen, where a complex query is formulated by combining the control structures and the results obtained through the normal GLAD DMC interaction.

We use the query "list all companies located in San Diego where three generations of a family all work for" as an elucidative example of using a program box. The **result1**

and **result2** boxes in Figure 6.1 are, respectively, for a father and grandfathers of **person**, where **person** is the variable used in the query formulation. The query formulations for these results are specified similarly as those described in the previous section, except that here the variable is used. The user attaches more meaningful names **fatherOf** and **grandfatherOf** to the **result1** and **result2** boxes by executing the LABEL command. The program box is invoked by the PROGRAM BOX command, and the query is specified as shown in Figure 6.2. A program box will be placed either on top of GLAD diagram as a separate window or on the side of GLAD diagram; if possible, the system will always put the program box on the side of GLAD diagram for better visibility. Notice that the functional notation similar to DAPLEX [SHIP81] is used. Also, notice that the list of control structures is provided.[11] Instead of actually typing in the whole program, a user can also create it by copying the result boxes and objects into the program box.

Here we treated a program box strictly as an added database accessing tool. But it is also a tool for program development interaction, because a user can create an application program by using a program box. So there is actually no clear distinction between program development interaction and data manipulation interaction in GLAD, which is exactly what we have hoped for. In other words, the GLAD DMC with a program box capability serves the dual role of query language and embedded host language. Data definition interaction via GLAD, moreover, is essentially a creation of database schema by arranging graphical objects such as rectangles, lines, and circle and specifying their associations, attributes, and integrity contraints. Therefore, in GLAD, all three user interactions have a consistent, unified interface method.

[11]Not all choices are shown in the figure.

7. Conclusion

We have presented in this paper a motivation behind a unified interface method for interactions with a database. And we have proposed a graphics user interface GLAD as an ideal candidate for such unified database interface method. We believe the GLAD's coherent interface method with its high degree of descriptiveness, ease of learning and using, and power will appeal to both sophisticated and naive users.

The result we have presented here is an outgrowth of the work reported in [WU85]. We originally viewed a graphical interface is suitable only for data manipulation. But we soon realized, after we reported our preliminary design of a graphical interface for accessing a database in [WU85], that our approach can be extended to the other two user interactions. Our work on data definition and program development interactions will be reported in forthcoming papers.

We have already initiated an implementation effort using the ISI workstations with graphics terminals. We anticipate that the kernel portion of GLAD DMC with a program box capability as described here will be implemented within the next six months. Our implementation goal is to make it portable so that it can be adapted as a front-end for any DBMS without much conversion effort.

Since there are too many areas of possible future research for GLAD, we shall mention here only some of those which are directly related to the subjects discussed in this paper. First is the representation and manipulation of an object which is a specialization of more than one generalized object. A work-study student object, for example, is a specialized object of both employee and student objects. We would like to add an elegant graphical representation of an inter-specialized object to a GLAD diagram. Second is the representation and manipulation of a generalized/specialized association such as a parent-of association which can be viewed as a generalized

association of father-of and mother-of associations. Just as we have a generalized/specialized object, it may be superior (as far as data modelling is concerned) to have a generalized/specialized association. And last is the graphical, syntax-directed program box. Instead of a user invoking a program box and typing in a program, we would like to have a user creates a program by utilizing a graphical, syntax-directed editor. With this editor, icons for the control structures are provided and a user creates a program by arranging these icons in a program box. We believe a technique similar to the one reported in [GLIN84] can be employed for this purpose.

References

[BRAG84] Gragger, R. P. , Dudler, A.. Refsamen. J.. Zehnder. C. A.. Gambit: An interactive database design tool for data structures, integrity constraints and transactions. In *Proceedings of IEEE International Conference on Data Engineering* (Los Angeles, 1984), 399-407.

[BOGU84] Boguraev, B. K. and Jones, K. S. A natural language front-end to data bases with evaluative feedback. In *New Applications of Databases*, Gardarin, G. and Gelenbe, E., Eds., Academic Press, London, 1984, 159-182.

[CODD74] Codd, E. F. Seven steps to RENDEZVOUS with the casual user. In *Proceedings IFIP TC-2 Working Conference on Data Base Management Systems*, North-Holland Publishing Co., Amsterdam, 1974, 179-200.

[GLIN84] Glinert, E. P. and Tanimoto, S. L. Pict: An interactive graphical programming environment. *IEEE Computer.* Vol 17, No 11, (Nov 1984), 7-25.

[HEIL85] Heiler, S. and Rosenthal, A. G-WHIZ, a visual interface for the functional model with recursion. In *Proceedings of 11th Conference on Very Large Data Bases*, (Stockholm. 1985), 209-218.

[HEND77] Hendrix, G. G. Human engineering for applied natural language processing. In *Proceedings of Fifth International Joint Conference on Artificial Intelligence*, (Cambridge, 1977, 183-191.

[HERO80] Herot, C. F. Spatial management of data. *ACM Transactions on Database Systems.* Vol 5, No 4 (Dec. 1980), 493-514.

[KORT84] Korth, H. F., Kuper, G. M., Feigenbaum, J., van Gelder, A., and Ullman, J. D. System/U: A database system based on the universal relation assumption. *ACM Transactions on Database Systems.* Vol 9, No 3 (Sept. 1984).331-347.

[LARS84] Larson, J. A. The forms pattern language. In *Proceedings of IEEE International Conference on Data Engineering* (Los Angeles, 1984), 183-191.

[MACG85] MacGregor, R. M. ARIEL -- a semantic front-end to relational DBMSs. In *Proceedings of 11th Conference on Very Large Data Bases*, (Stockholm, 1985), 305-315.

[MCDO74] McDonald, N. and Stonebraker, M. CUPID - the friendly query language. Memo ERL-M487. ERL. University of California. Berkeley, CA. October. 1974.

[PETR76] Petrick, S. R. On natural language based computer systems. *IBM Journal on Research Developments.* Vol 20, No 4 (July, 1976), 314-325.

[HEIL85] Rowe, L. A. "Fill-in-the-Form" programming. In *Proceedings of 11th Conference on Very Large Data Bases*, (Stockholm, 1985), 394-404.

[SHIP81] Shipman, D. W. The functional data model and the data language DAPLEX. *ACM Transactions on Database Systems.* Vol 6, No 1 (March, 1981), 140-173.

[STON82] Stonebraker, M. and Kalash, J. TIMBER: a sophicticated relation browser. In *Proceedings of 8th Conference on Very Large Data Bases* (Mexico City, 1982), 1-10.

[SUGI84] Sugihara, K., Miyao, J., Kikuno. T. and Yoshida, N. A semantic approach to usability in relational database systems. In *Proceedings of IEEE International Conference on Data Engineering* (Los Angeles, 1984), 203-210.

[THOM75] Thomas, J. C. and Gould, J. D. A psychological study of query by example. In *Proceedings of the National Computer Conference* 44, (1975), 439-445.

[WALT78] Waltz, D. L. An english language question answering system for a large relational database. *Communications of ACM.* Vol 21, No 7 (July, 1978), 526-539.

[WONG82] Wong, H. K. T. and Kuo, I. GUIDE: graphical user interface for database exploration. In *Proceedings of 8th Conference on Very Large Data Bases* (Mexico City, 1982), 22-32.

[WU85] Wu, C. T. A graphical user interface for accessing a database. Submitted for publication.

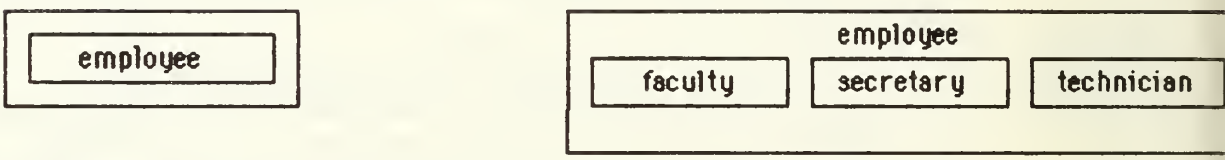
[ZLOO77] Zloof, M. M. Query-by-example: a data base language. *IBM Systems Journal* , 4 (Dec. 1977), 324-343.

a) student object

b) expanded object

c) association between
dept and student objects

FIGURE 2.1



a) generalized object

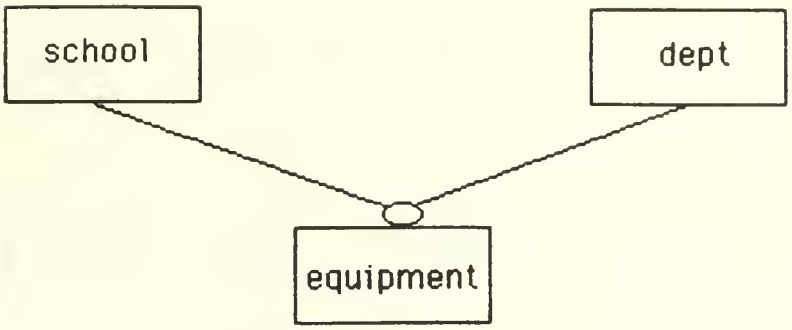
b) expanded generalized object

FIGURE 2.2

Either i) sub-object of A is related to B or
ii) sub-object of B is related to A or
iii) sub-object of A is related to sub-object of B.

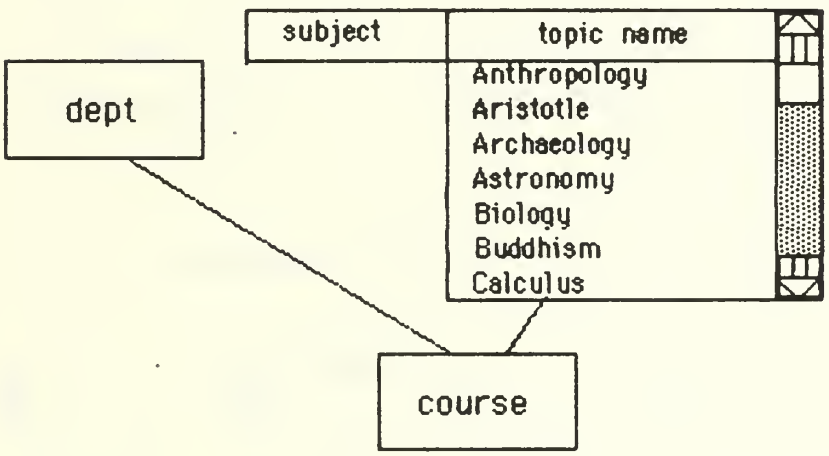
If a user is interested in this relation, he can either expand A and/or B, or prompt GLAD to describe the relationship between A and B. This is a very important point; GLAD only provides more information when asked, it will not force information on a user.

FIGURE 2.3



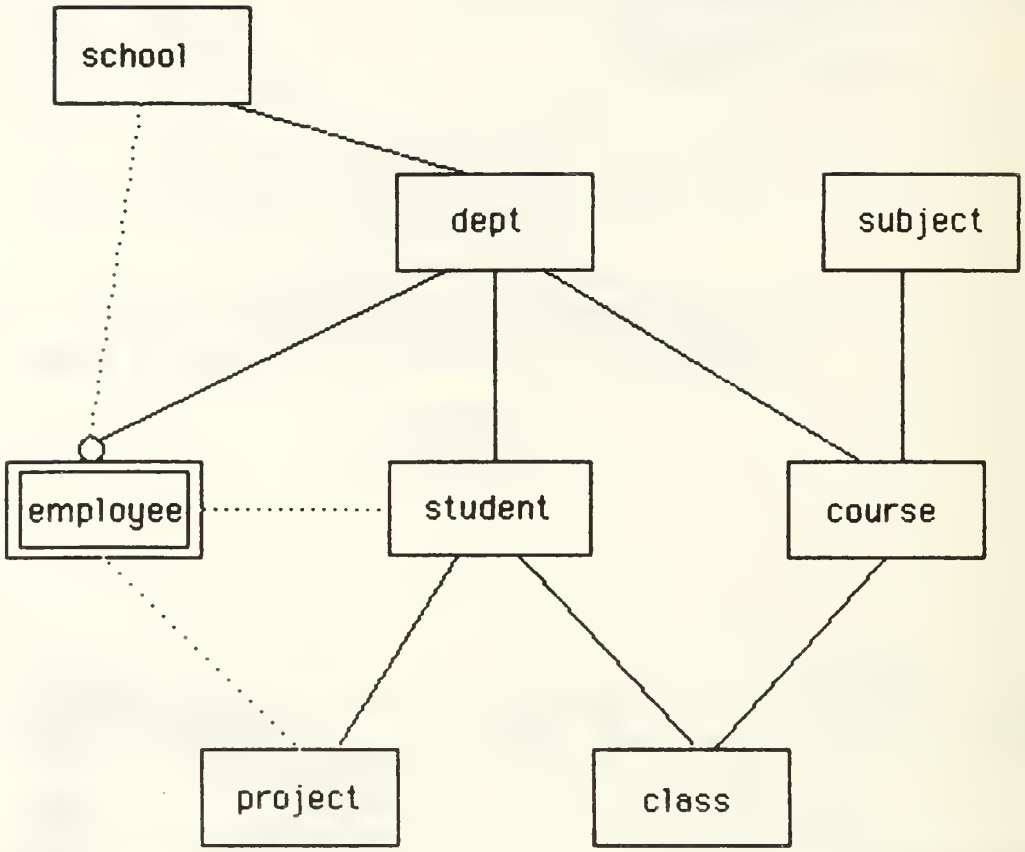
Disjunctive relation; above shows that an equipment either belongs to a school or a dept

FIGURE 2.4



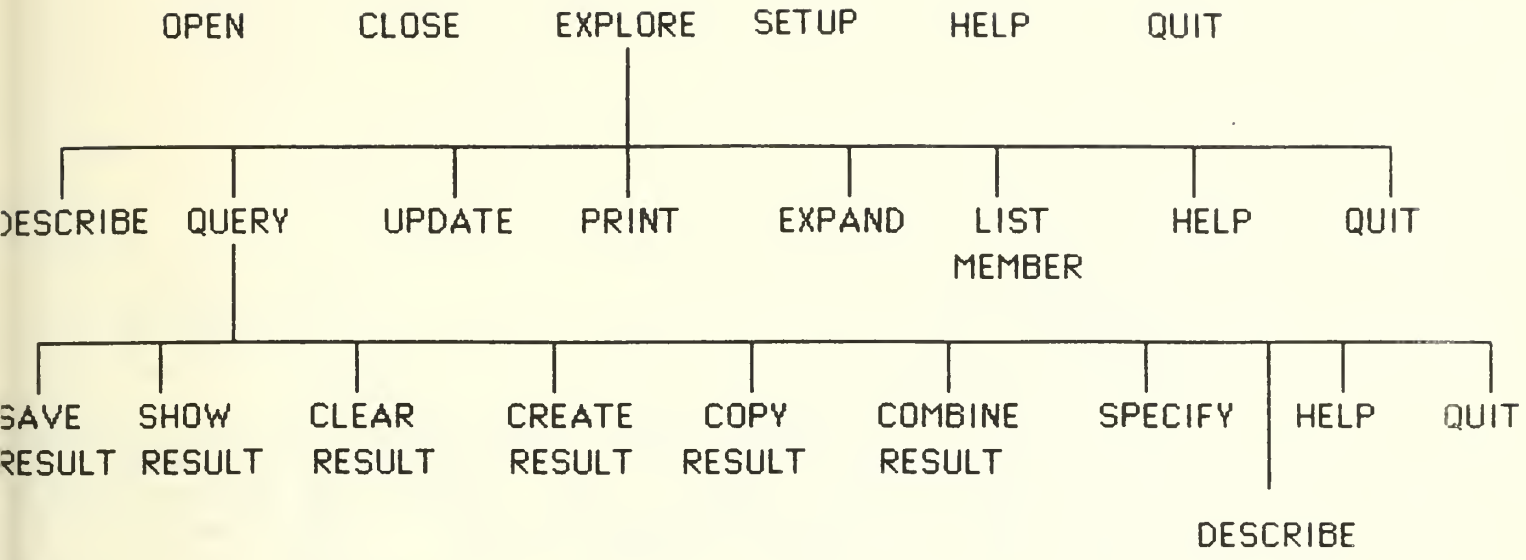
Listing members of subject object

FIGURE 2.5

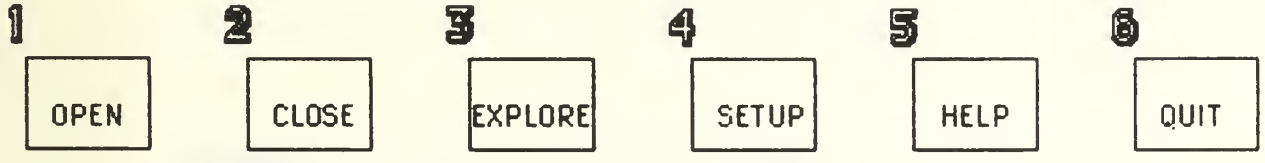


University Database

FIGURE 3.1

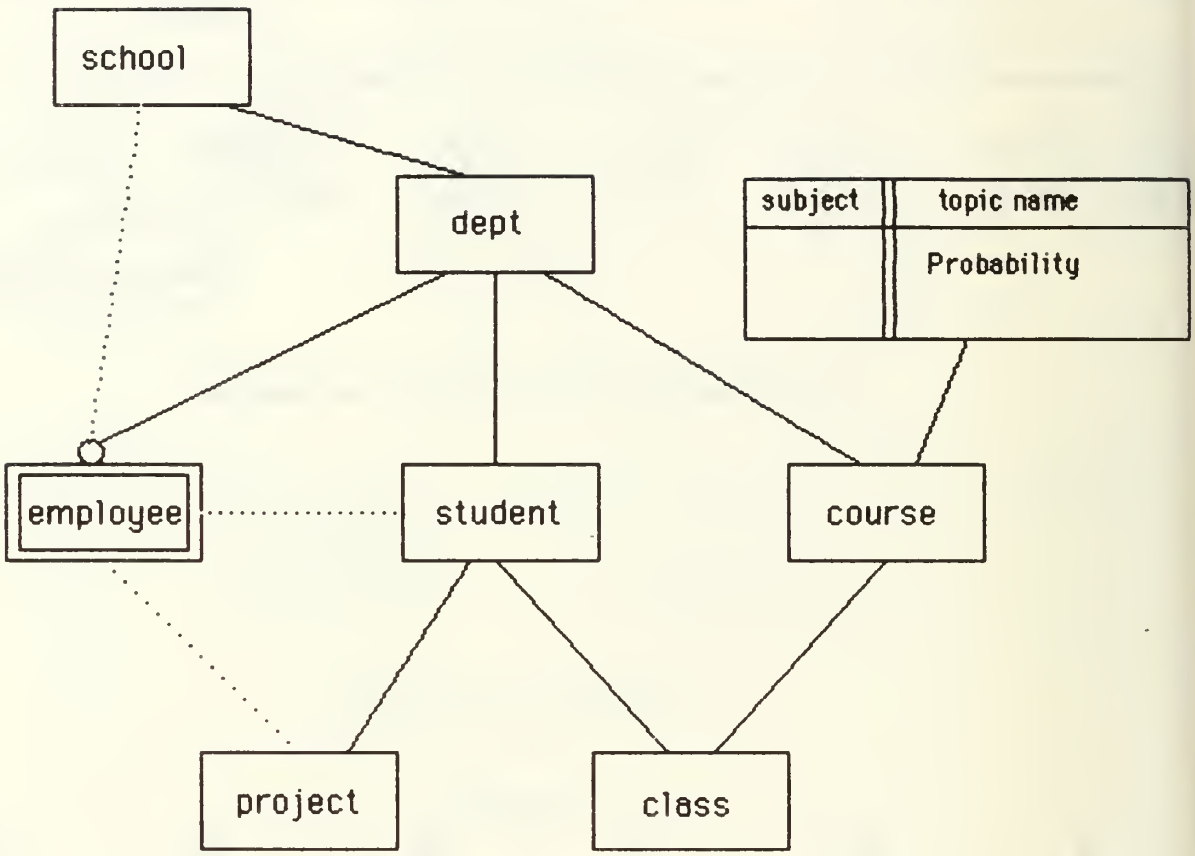


a) Hierarchical structure of commands



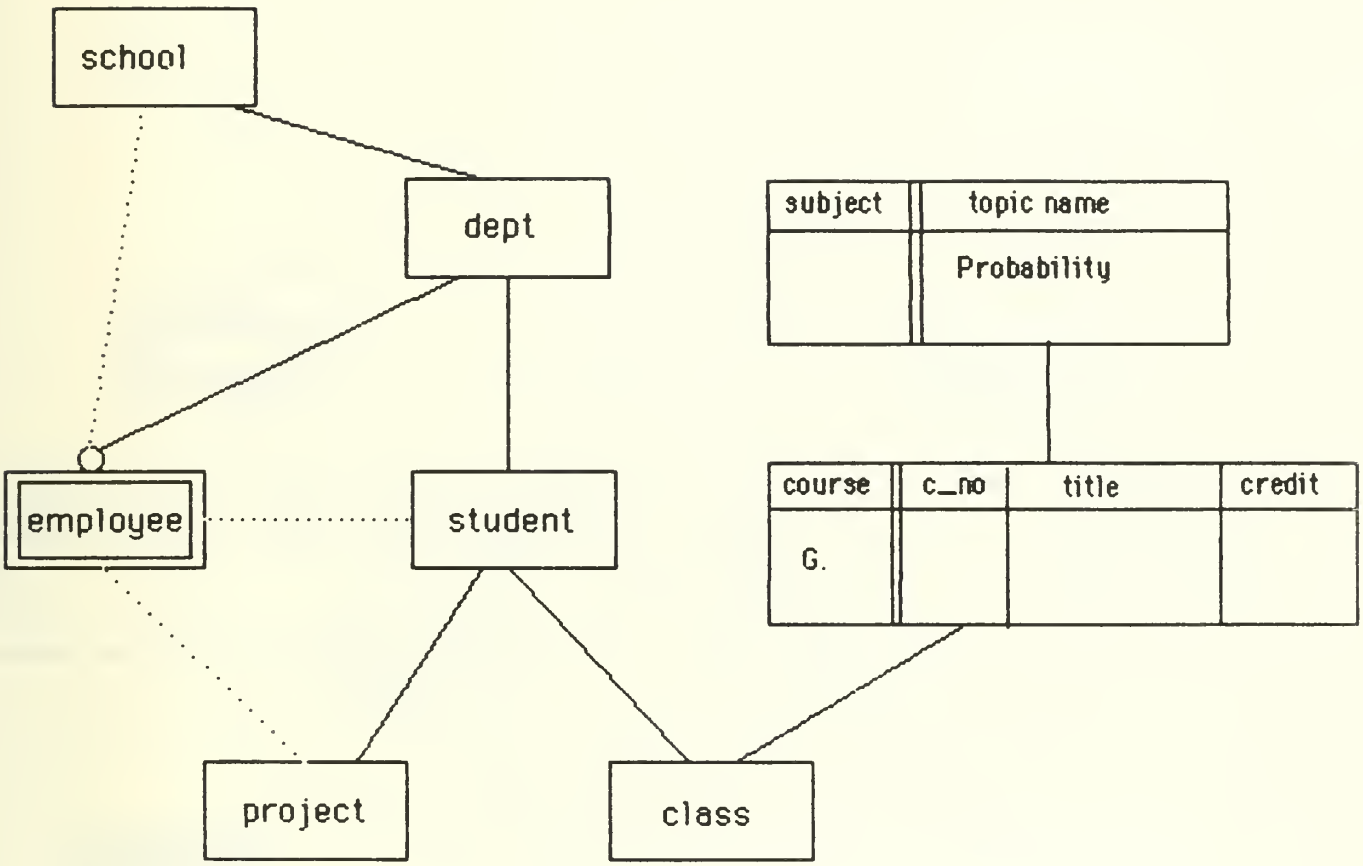
b) Top level menu

FIGURE 3.2



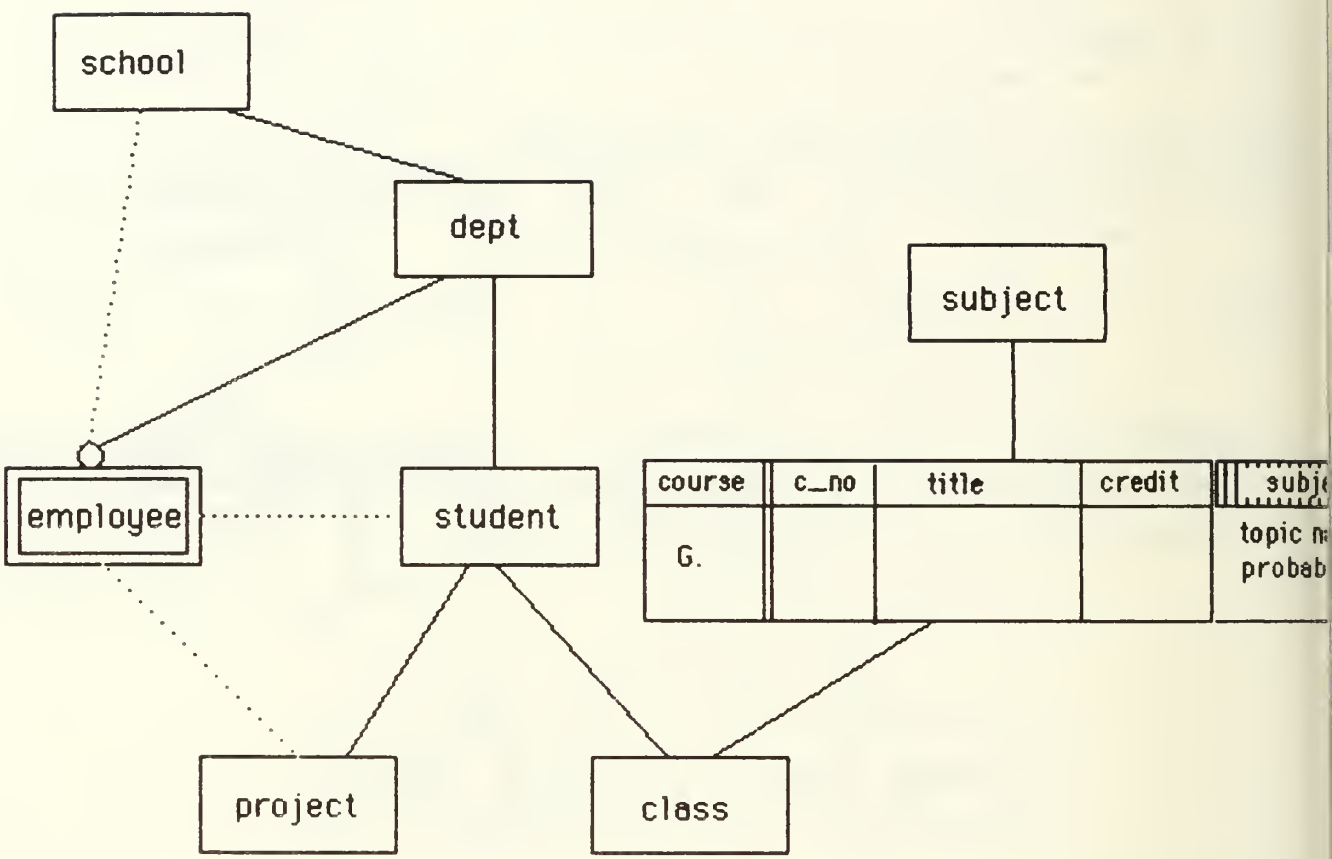
Specifying the desired topic probability

FIGURE 3.3



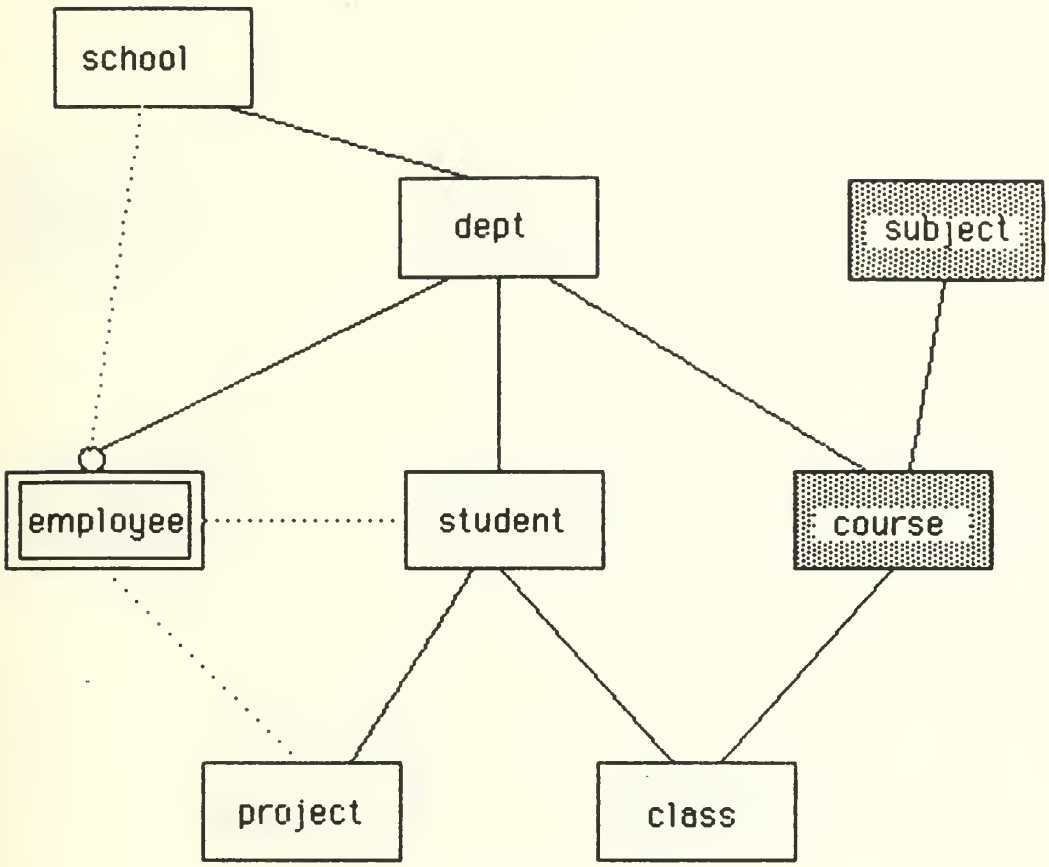
Request to retrieve (G.) all information on course is made in the course object skeleton

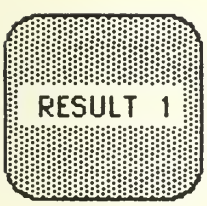
FIGURE 3.4



Alternate method to specify the first subquery

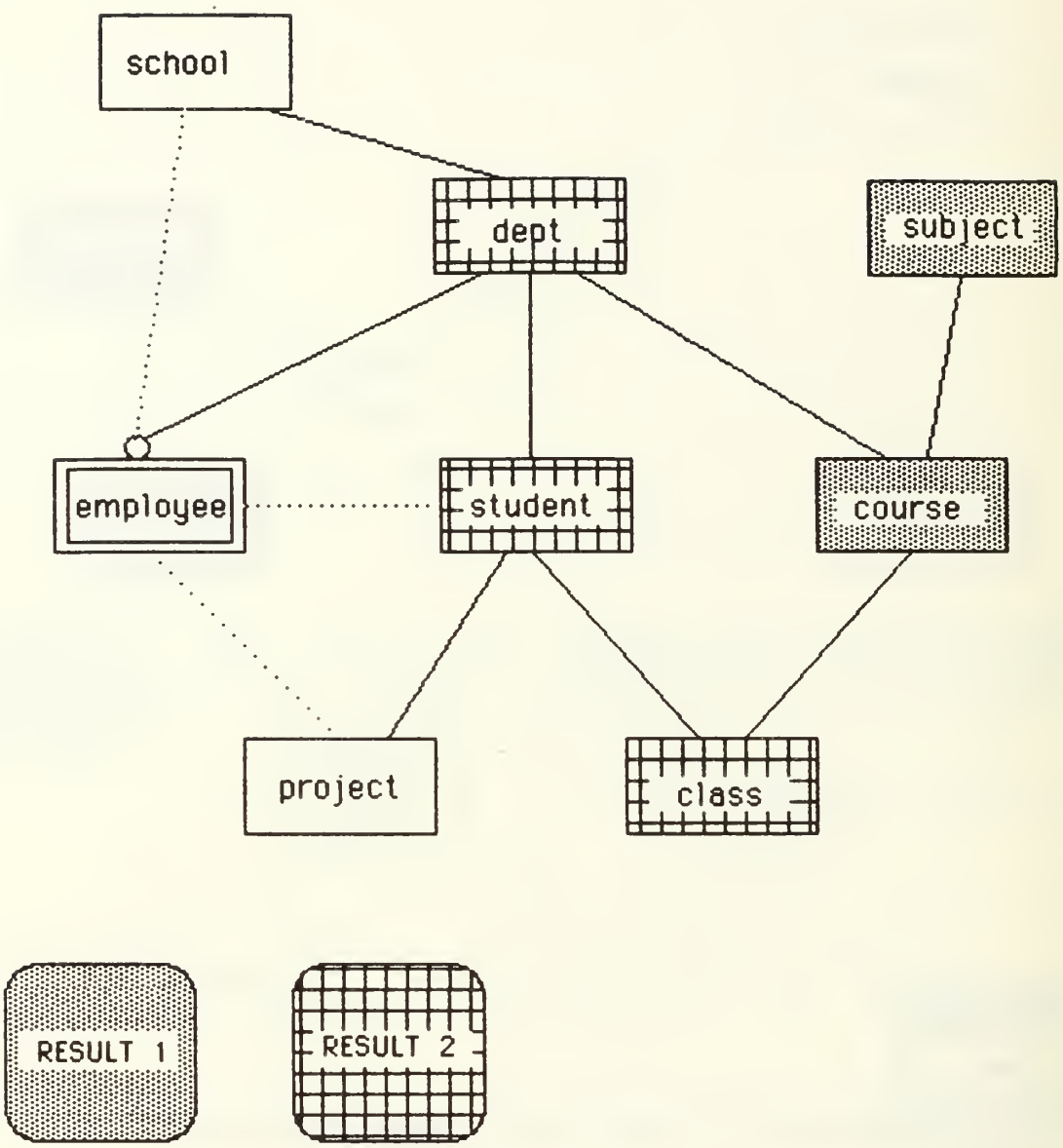
FIGURE 3.5





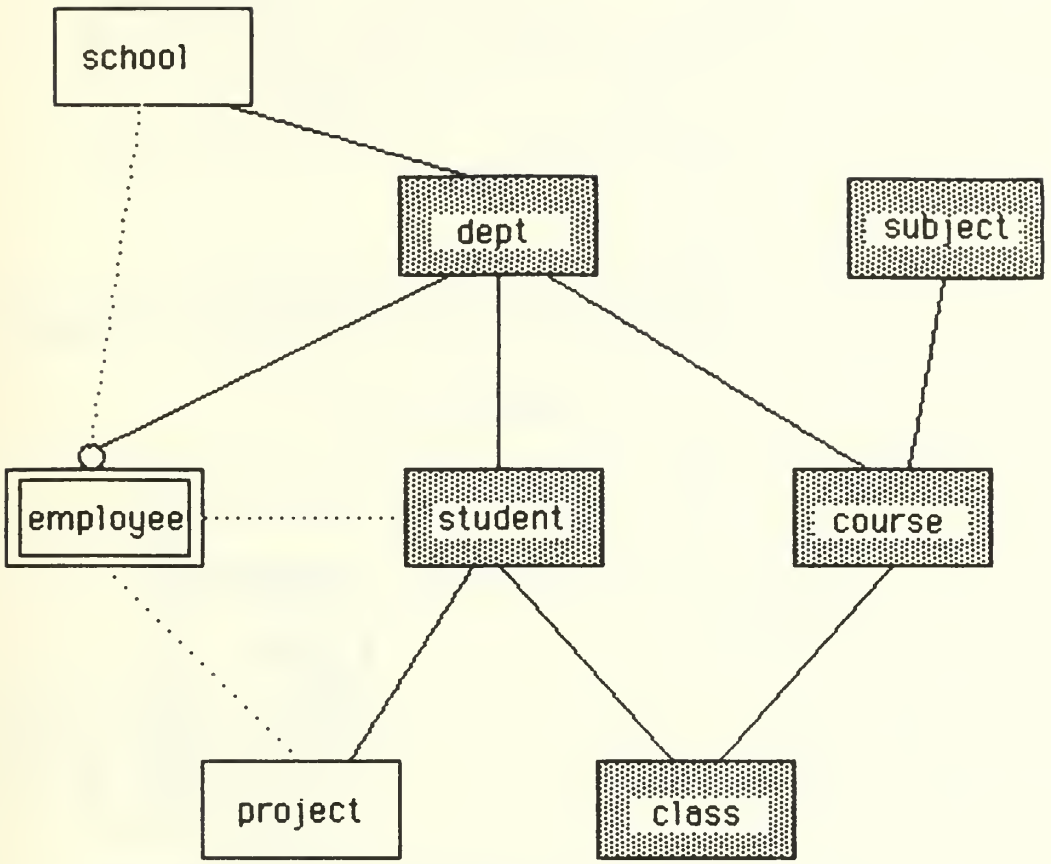
Result of the first subquery is created

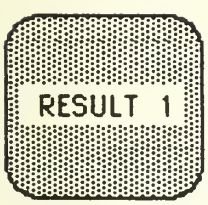
FIGURE 3.6



Result of the second subquery is added

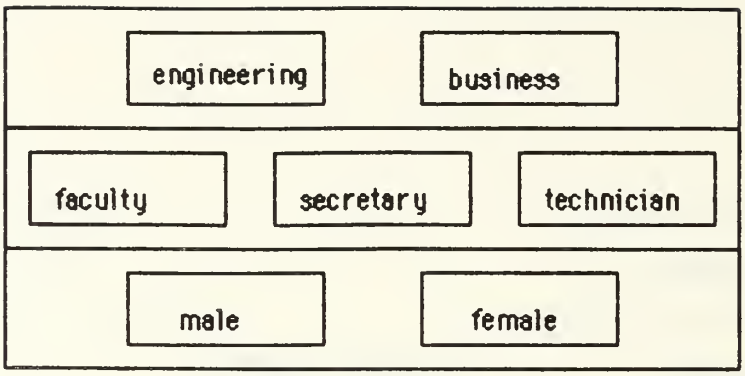
FIGURE 3.7





Result of the complete query is now created

FIGURE 3.8

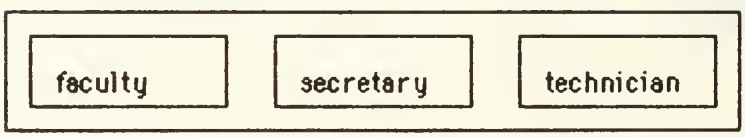


One possible way to represent
specializations in more than one category

FIGURE 4.1

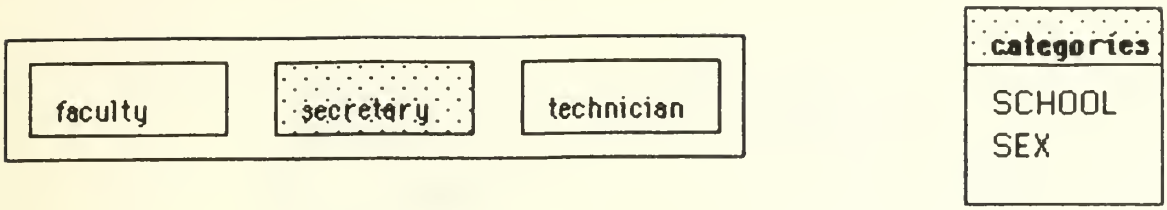
A pop-up menu for specialization categories

FIGURE 4.2



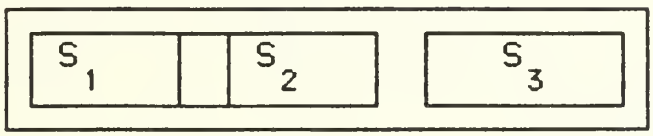
An employee object expanded along
the job category

FIGURE 4.3



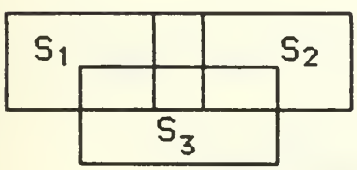
After a user requested to EXPAND the secretary object

FIGURE 4.4

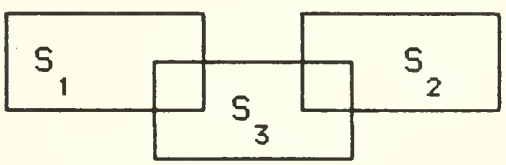


Specialized objects S_1 and S_2 are not disjoint

FIGURE 4.5



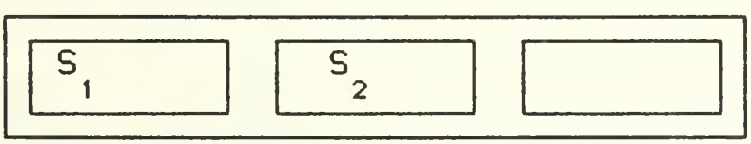
a) all three overlaps



b) No overlap between S_1 and S_2

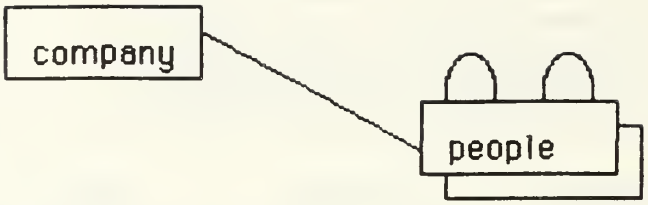
Overlapping of specialized objects

FIGURE 4.6



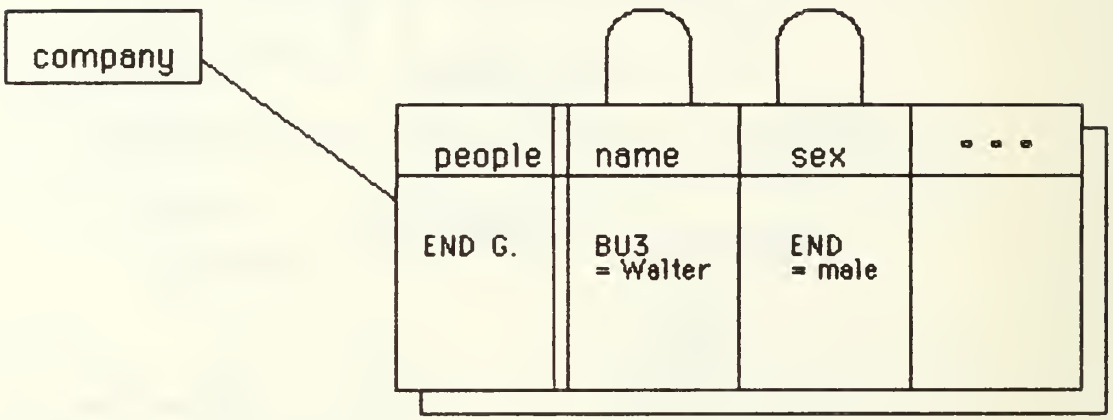
Unnamed rectangle signifies that there is an element that does not belong to S_1 and S_2

FIGURE 4.7

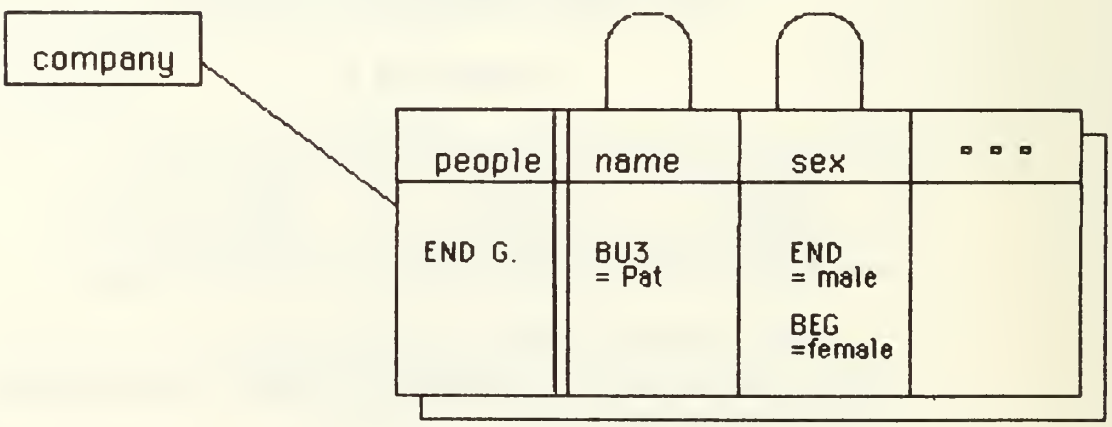


The people object is repeated to denote that it is a recursively related object. There are two recursive associations.

FIGURE 5.1



a) Retrieval of Walter's grandfathers



b) Retrieval of (female) Pat's grandfathers

FIGURE 5.2

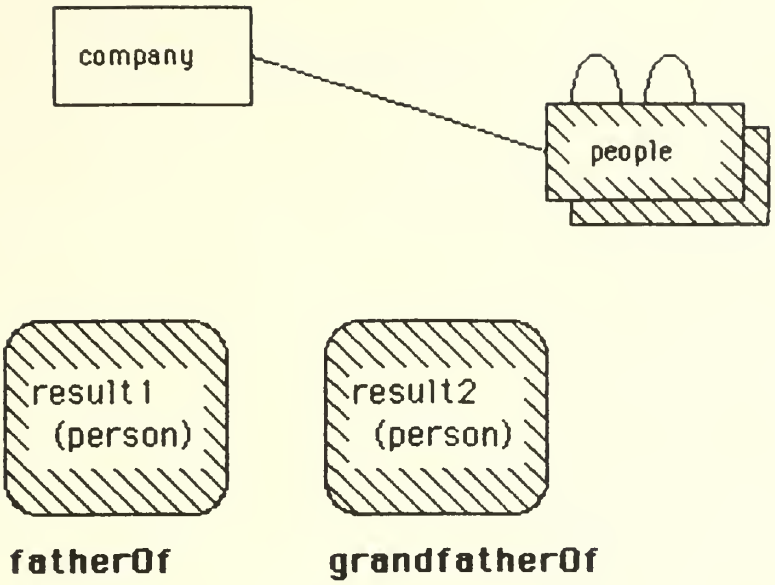
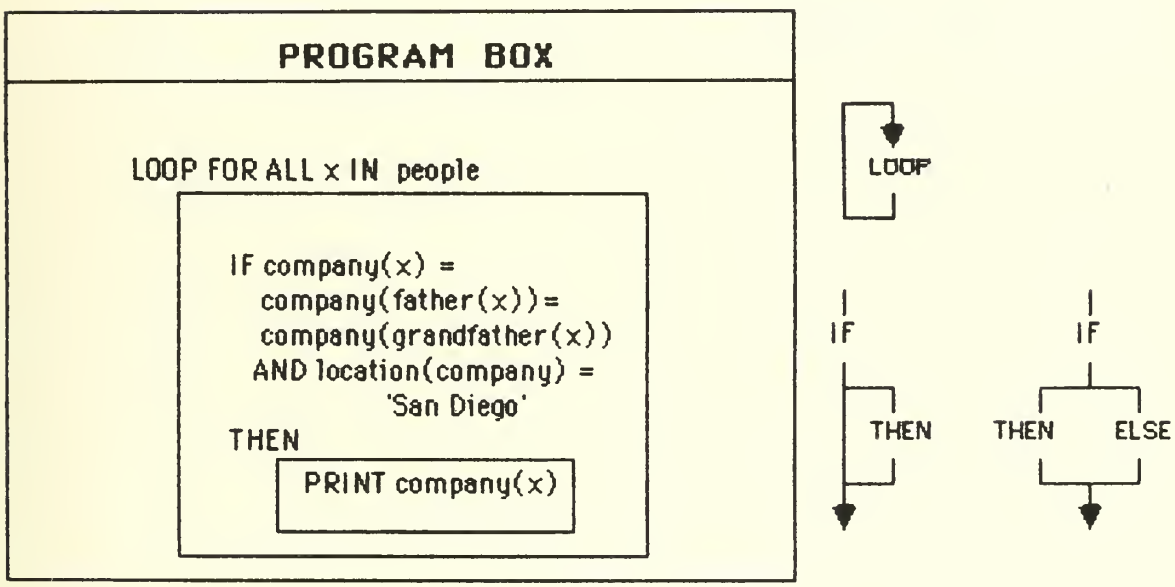


FIGURE 6.1



Program for listing all companies in San Diego where three generations of a family work

FIGURE 6.2

INITIAL DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Dudley Knox Library Code 0142 Naval Postgraduate School Monterey, CA 93943-5000	2
Office of Research Administration Code 012 Naval Postgraduate School Monterey, CA 93943-5000	1
C. Thomas Wu Code 52Wq Computer Science Department Monterey, CA 93943-5000	20
Chief of Naval Research Arlington, VA 22217	1

DUDLEY KNOX LIBRARY
3 2768 00347477 6