Lessons from a "Living In a Database" Graphical Query Interface

Dennis Fogg[1]

Abstract

The Living In a Database system (LID) is a user-friendly interface to an entity-relationship database. Its underlying ideas are similar to Cattell's PDB [Cattell 80], but its presentation is significantly different. LID uses a bit-mapped graphics terminal with mouse pointer to create an attractive interaction environment. Experience from the implementation suggests that dynamic graphic displays -- those which have graphic symbols that change as the data they present change -- are an important feature in user interfaces but are difficult to implement with current technology. The implementation also uncovers an important inadequacy in the PDB/LID idea: the inability to operate on sets of data instances in the same way as individual data instances. An extension to LID is suggested to alleviate the problem.

1. Introduction

The need to provide user-friendly interfaces has been studied by several investigators. Queries in QBE [Zloof 75] are formed by providing example answers in relational model templates. SDMS [Herot 80] uses graphics to allow users to browse above a sea of data and zoom in or out to see varying levels of detail. SDMS is handicapped, however, by its inability to create groups of related data items and perform aggregate operations on them. Timber [Stonebraker 82] is a user-friendly interface that introduces browsing and powerful update commands in a relational database. Timber's weakness is the absence of a user-friendly relational join operator to connect relations.

The interface described in this paper uses the ideas of PDB [Cattell 80], a semantic network database browser. In that system each node in the semantic network is represented by a *frame* which contains the node's name, associated text, and a list of adjacent nodes. A user queries PDB by examining the contents of nodes and by moving to adjacent nodes to gather related information. PDB is implemented on a regular character oriented screen so the user's interaction with the interface is rather primitive: each adjacent node is preceded by a single character and users move to related frames by typing the associated character. [Cattell 80] suggests more user-friendly mechanisms can be used for selection depending on the hardware available.

The Living In a Database (LID) interface uses the ideas of PDB in the entity-relationship data model. The LID style of interaction lets users live inside the database and answer queries by simply examining their surroundings. Specifically, users live inside a single entity-relationship tuple and see the database from the perspective of that tuple. Simple queries are answered by finding the appropriate tuple to live in. More complicated queries are expressed by browsing in the maze of data created by relationship set tuples. LID's appearance to users differs significantly from PDB because LID takes advantage of a bit-mapped display terminal and mouse pointer. The next section illustrates the flavor of the interaction through an example and comments on issues in dynamic graphics. Section 3 points out an important deficiency in LID's and PDB's ideas and suggests a solution. Throughout this paper the reader is assumed to have basic knowledge of the entity-relationship (ER) model [Chen 76].

2. LID's Style of Interaction

LID's original display was very similar to Cattell's. It used a regular character oriented terminal to list a tuple's attributes and values along with the entities it was attached to. User commands were numbers or letters typed from the keyboard. Demonstrations were given with this simple display, and the overall reaction was mild. When the implementation was modified to run with a bit-mapped display terminal, user

[1]Author's current address is: Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Mass. 02139

reaction jumped to very enthusiastic. We believe this phenomenon is important. That is, different segments of the potential user audience consider different criteria to evaluate user interfaces. For naive audiences (the user group that user-friendly interfaces are aimed at) appearance will be a major factor in selecting interfaces. The following example illustrates how LID presents PDB's ideas on a bit-mapped display. Since LID uses a different data model than PDB, new terms will be defined in the example.

2.1. A LID Session

The sample database shown in Figure 1 will be used to answer the query "Who is in Dennis Fogg's group?" Our strategy is to follow a two step plan: (1) find the Fogg tuple in the database, (2) browse around looking for his fellow group members. The session begins by showing a menu for the 3 button mouse (see the bottom of Figure 2). The leftmost menu element is labeled "entity menu" and when the user depresses the corresponding mouse button a pop-up menu appears containing a list of possible entities to live in (Figure 2). We want to find the Fogg tuple so we select the "employee" entity. Figure 3 appears. This is LID's primary display. It shows the database from the perspective of one particular tuple called the *current tuple*. Values for the entity fields are listed in the *current tuple box* at the center of the screen. The diamond and smaller box connected to the current tuple resemble the relationship set and entity symbols in the global ER diagram of Figure 1, but there is a subtle and important difference. Their presence indicates the current tuple is connected to at least one tuple in the dept entity through the dept–sec relationship set. That is, Penny Holl is a secretary for some department. This is a key idea in LID: entities and relationship sets are listed only if there is an instance of the current tuple in the relationship set. Other relationship sets may be connected to the employee entity in the global schema, but their tuples are unreachable from this tuple so they are hidden. It is convenient to call dept–sec a *path* and dept a *path entity*. This is slightly different from Cattell's system in that PDB would list all instances of the dept tuples around the current tuple box whereas LID aggregates them into one structure.[2]

The Holl tuple appeared when we asked to live in the employee entity, but we wanted to find the Fogg tuple. How do we get there? When we selected the employee entity, a set of tuples was placed into the *current tuple list*. The Holl tuple happened to be first in the list so it was displayed. The numbers in the right corner of the current tuple box indicate that there are 13 tuples in the current tuple list and the Holl tuple is the first. To find the Fogg tuple, we will use the search function which finds a tuple within the current list. Alternatively, we could sequence through the list using the next or previous commands or access it directly by specifying the position number. The search command is invoked by moving the mouse to the current tuple box and depressing the button labeled "select". When the area is active (i.e. text is reversed videoed and an explanation of the command appears below the mouse menu as in Figure 4) releasing the button executes the command. Command selection differs from most other interactive systems in that the selection is based on both the mouse button and the mouse position on a changing display. The search command produces the screen in Figure 5 where search strings are entered to find the Dennis Fogg tuple. The search produces Figure 6. Notice that Fogg's paths are different from Holl's paths. Fogg has a group--member relationship set instance to the group entity but none in dept–sec to dept. That is, Dennis Fogg is a group member.

To find Dennis Fogg's group members, we simply browse through the database. From the choices in the Figure 6 the only reasonable entity to enter is group. To travel into a related entity, we activate the path follow command by moving the cursor to the group–member path and depressing the select mouse button. The result (Figure 7) is a new current tuple list containing all the group tuples connected to the Fogg tuple. It is important to recognize that although there may be many tuples in the group entity only those connected to the Fogg tuple through group–member are in the current list. Notice the background box behind the current tuple box reminds users that the current tuple list was derived from the employee entity through group–member.

To get a list of group members in Dennis Fogg's group, we simply follow the group–member path back into employee to get employees connected to Dennis Fogg's group. The 3 tuple list is shown in Figure 8. Notice that traveling back over the group–member path did not cancel the effect of the first path follow. The second path follow simply retrieves all tuples connected to the current tuple. What if we did want to cancel the last path follow? For example, let's find Dennis Fogg's department head. Instead of reentering the employee entity and searching for Fogg, we can undo the last path follow and return to Fogg's group. The unfollow command is selected by depressing the select mouse button and pointing to the "GROUP --> group–member" list description in the background of the current tuple box. The result is Figure 9. To find the department head, we choose the dept–group path to enter the dept entity (Figure 10) and then the dept–head path to the employee entity for the answer (Figure 11).

2.2. A Comment on Tailoring Graphics to Changing Data

Figures 2 - 11 are examples of graphic displays that change as the data they display changes. Transformations include simple changes like varying dimensions of boxes to more complicated alterations like determining the screen positions of radiating path diamonds and path entity boxes.[3] LID revealed that the programming technology to implement these dynamic graphic symbols is quite limited. For example, in LID we originally intended to include a "radar" command. The need

[2] An experiment was tried in which all instances of related tuples were displayed as in PDB. The character of the particular data set used had high fanout in some 1:n relationships. Thus, the current tuple required many sets of diamonds and boxes around it. Managing the large number of tuple instances with limited screen space became too troublesome to handle. The scheme illustrated in Figure 3 has better scaling properties in this respect

arose when a user was in the Dennis Fogg tuple and wanted to find Fogg's dept head. How does he know that he should go to the group entity? What other entities are near (but not adjacent to) the current tuple? Radar could be used to change the standard display in Figure 6 to the diagram with solid lines in Figure 12. A second application of radar adds the dashed lines in Figure 12. The diamonds and rectangles are still paths and path entities, not ER model structures. That is, a path appears only if there is a "data connection" between the path entity and the current tuple. From the graphics implementation view, the difficulty is dynamically producing a user-digestible diagram where every path and path entity appear only once, their placement is readable, and the display does not waste valuable screen space. Graphic icons can not be assigned fixed locations as in traditional graphics programs because the diagram must change as the data it displays changes. Brute force tree expansions algorithms were rejected because they wasted screen space. Simple algorithms that draw a single instance of each path and path entity failed to produce palatable diagrams. The unexpected difficulties of the algorithmic techniques contrast the unnoticed simplicity of the human solution and suggest artificial intelligence methods may be more appropriate. In particular, the knowledge-based systems technology may be a good compromise for encoding human diagram creation knowledge into the algorithmic computer. Hopefully, a forthcoming issue of [IEEE 85] will discuss some of these solutions.

3. List-oriented LID

Both LID and PDB suffer from the weakness that sets of data items can not be operated on like single data items. Unfortunately, this limitation imposes significant restrictions on LID's utility. Consider the query "list all group−members in the Advanced Systems Department" for the data schema in Figure 1. In LID a user would have to do a depth first tree traversal: travel from the Advanced Systems dept tuple through the dept−group path to produce a list of groups, examine the members of the first group by following the group−member path, unfollow back to the list of groups, examine the second group's members, unfollow again, etc. An important conceptual modification called list-oriented LID solves this problem.

List-oriented LID is a generalization of tuple-oriented LID in which users live in a list of tuples instead of in a single tuple. List-oriented paths are present in the display when at least one tuple in the current tuple list is connected by the relationship set to a tuple in the path entity. The path follow command produces a new tuple list from the path entity tuples. For example, if the current list is the single tuple "Advanced Systems Department" then following the dept−group path would produce a current list containing all the groups in the dept. Now, if the group−members path is followed then the new current list contains all the group members in all the groups in the dept. Thus, user can take advantage of the set level operations of list-oriented LID to avoid the tree search required with tuple-oriented LID.

List-oriented LID requires a few additional commands. A "make the current tuple a separate list" command should be convenient so list-oriented LID can do everything tuple-oriented LID can do. Searches should form lists in list-oriented LID instead of position within a list as in tuple-oriented LID. Set operations and ways to store and retrieve lists are needed to combine lists. The novelty of list-oriented LID is that it spans a spectrum of database perspectives. With one uniform mechanism, the relationships of a single tuple can be investigated as in tuple-oriented LID, the relationships of entire entities can be examined as in the global ER schema, or the relationships of any entity subset can be browsed. Entities, entity subsets, and individual tuples can be merged into the single concept of a tuple list for query expression.

4. Summary

LID is a query interface based on the ideas of Cattell's PDB and implemented on a graphics terminal. Experience reveals that LID's pleasing display and interaction format are an important factor in evaluation of the system by naive users. Unfortunately, the implementation also suggests that producing dynamic graphic symbols, an important factor in aesthetic displays, is difficult with current technology. As a separate issue experience from using LID uncovered a significant inadequacy in the PDB/LID ideas. List-oriented LID is proposed to alleviate the problem.

5. Acknowledgements

Past association with Murari Kumpati and Mark Johnson have been very beneficial to this work. Thanks also go to Mike Stonebraker for his comments on an earlier draft of this paper.

References

[Cattell 80] Cattell, R. G. G.
An Entity-based Database User Interface.
Proc. of 1980 ACM-SIGMOD Conference on Management of Data, May, 1980.

[Chen 76] Chen, P. P.
The Entity-Relationship Model -- Toward a Unified View of Data.
ACM Transaction on Database Systems 1(1), March, 1976.

[Herot 80] Herot, C. F.
Spatial Management of Data.
ACM Transactions on Database Systems 5(4), December, 1980.

[3] Our algorithm for diamond and box placement defined a fixed number of screen locations for path entity boxes. At runtime path diamonds are placed along the connecting line between the current tuple box and the path entity box such that they do not overlap with other objects in the display.

[IEEE 85]
IEEE Computer Graphics and Applications.
Issue on expert systems in computer graphics due in early 1985.

[Stonebraker 82] Stonebraker, M. and Kalash, J.
TIMBER: A Sophisticated Relational Browser.
Technical Report, University of California, Electronics Research Laboratory, May, 1982.
Memo UCB/ERL M81/94.

[Zloof 75] Zloof, M. M.
Query by Example.
1975 National Computer Conference, May, 1975.

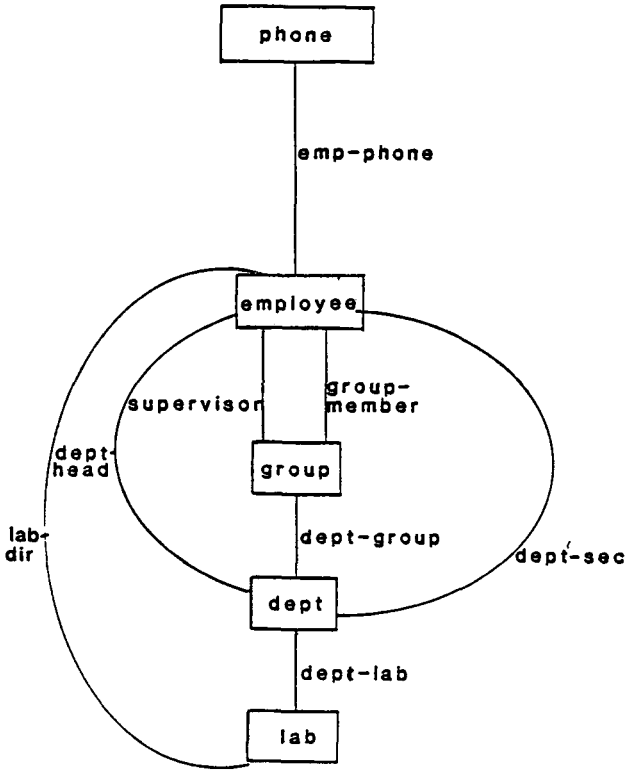

Figure 1.

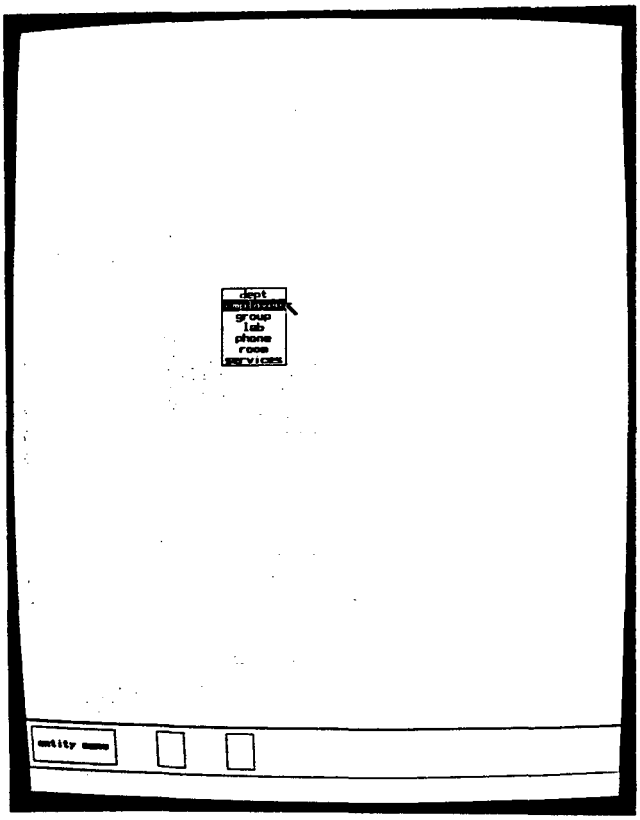

Figure 2.

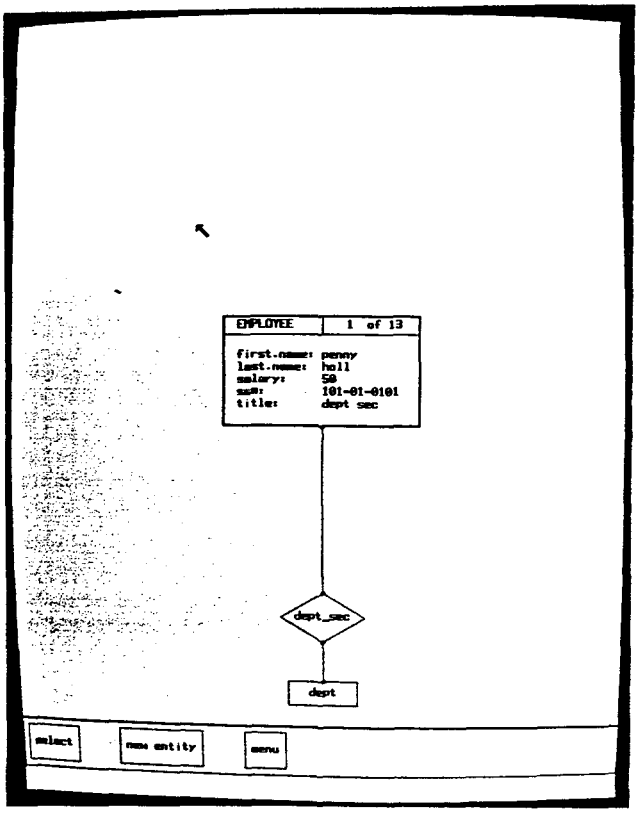

Figure 3.

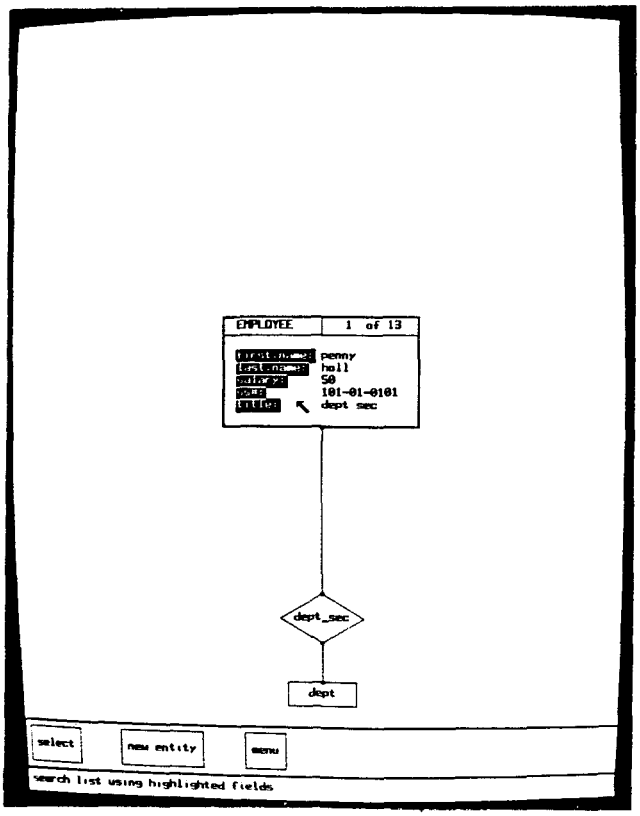

Figure 4.

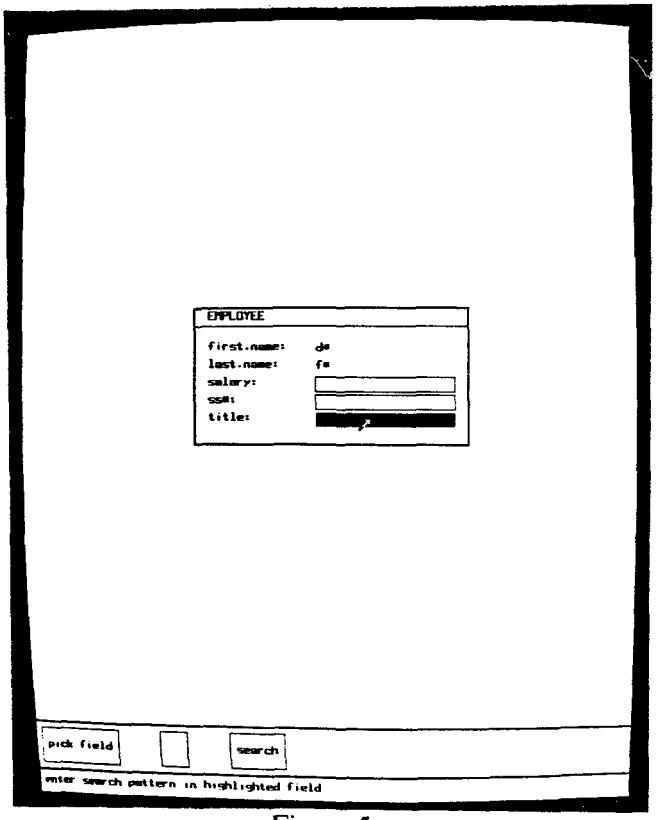

Figure 5.

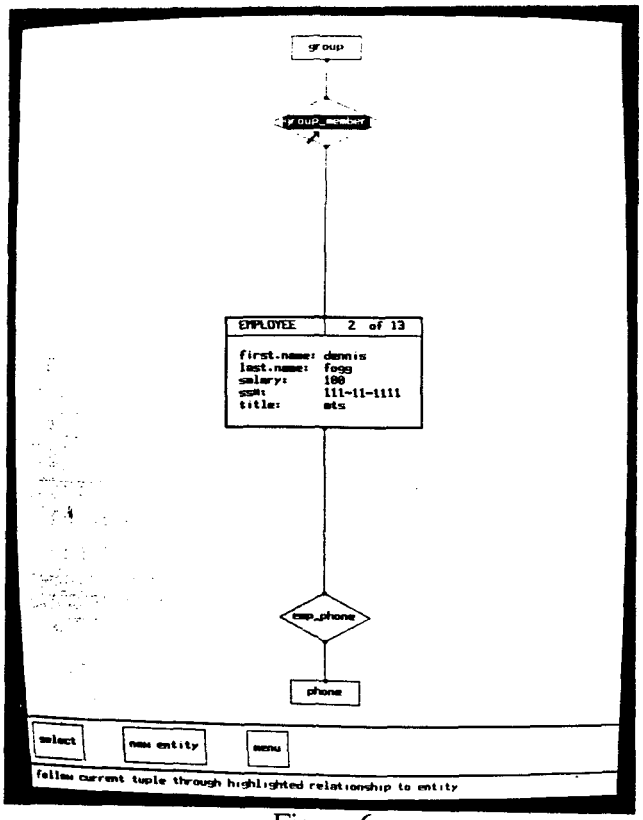

Figure 6.

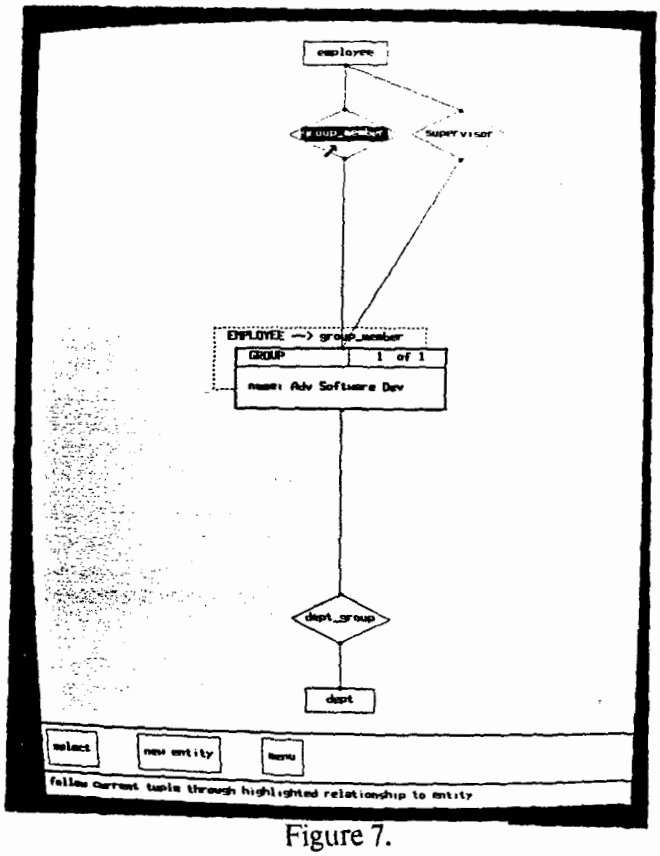

Figure 7.

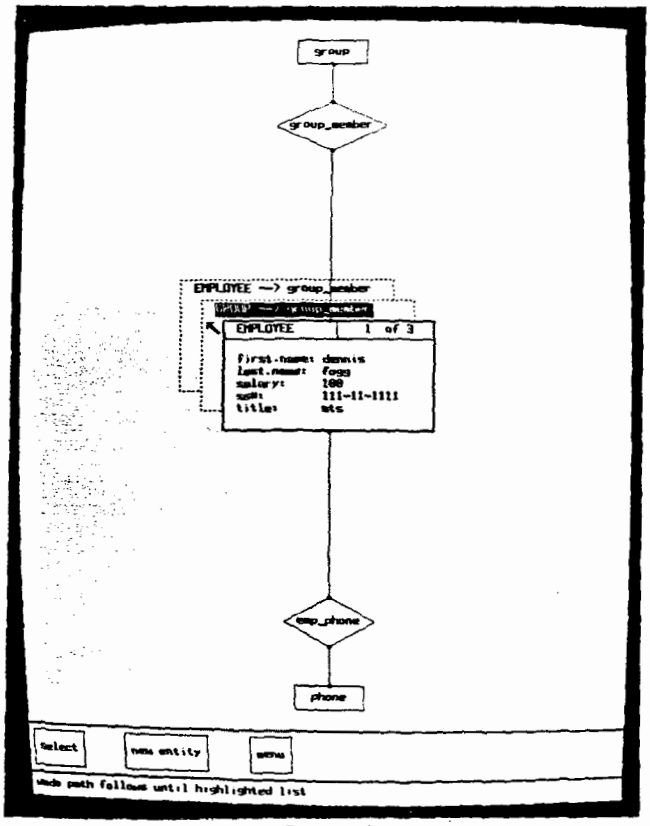

Figure 8.

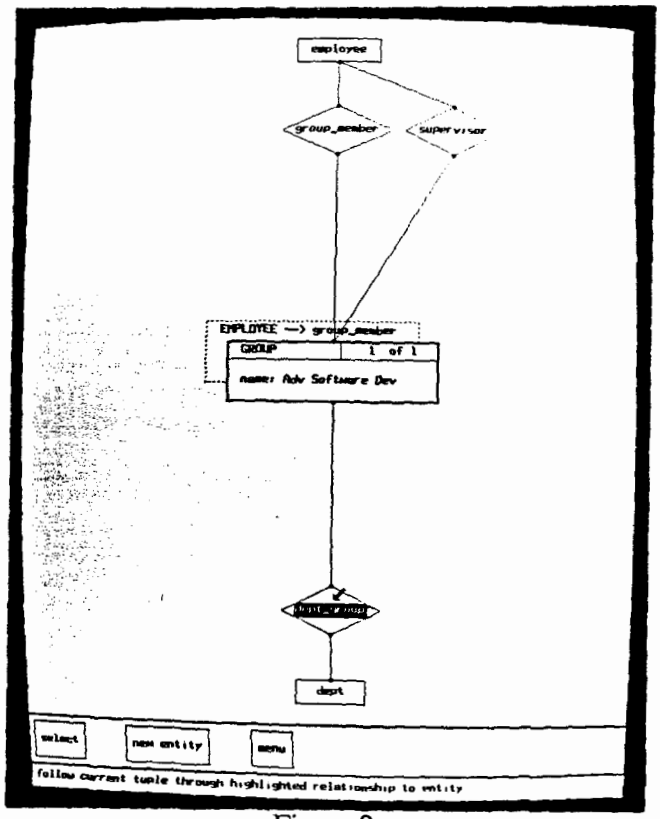

Figure 9.

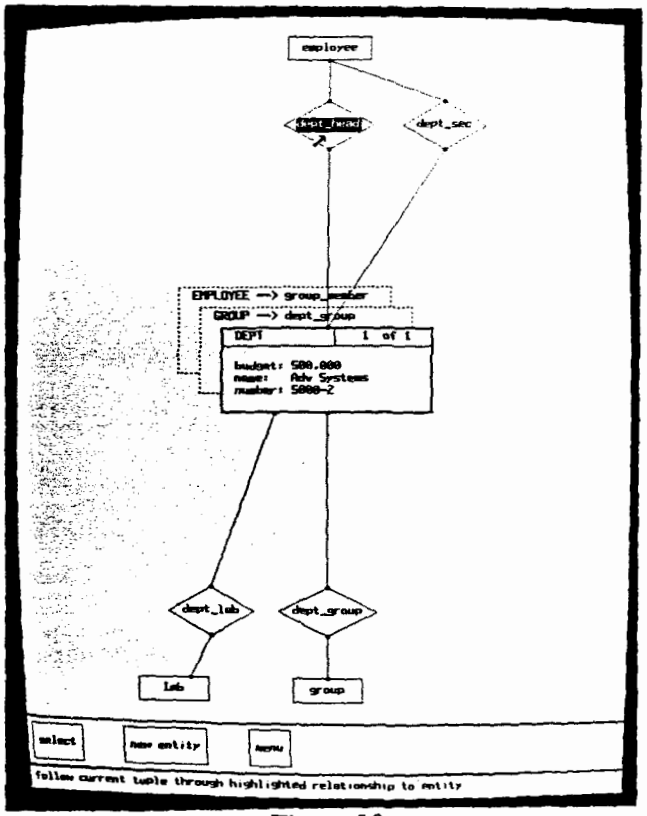

Figure 10.

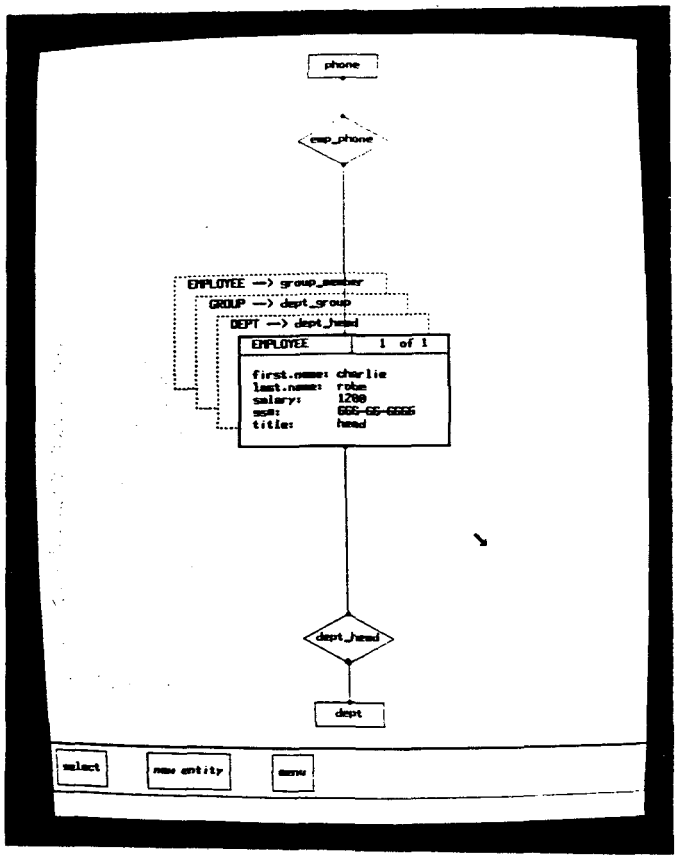

Figure 11.

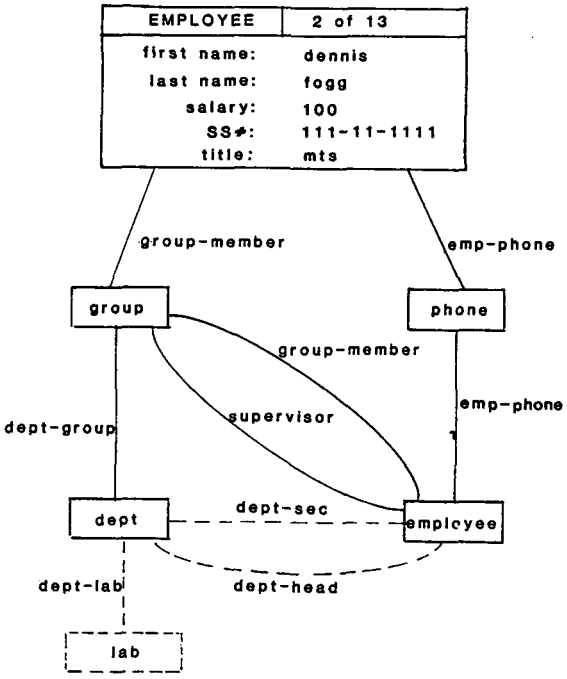

Figure 12.