AN ABSTRACT OF THE THESIS OF

Raimund KH Ege for the degree of Master of Science in Computer Science presented on July 10, 1984.

Title: The Display Function in ALLEGRO

Redacted for Privacy

Abstract approved: _____
Earl F. Ecklund, Jr.

Display, which is the function for displaying the schema of the network at the conceptual level, is implemented to enhance the network database system ALLEGRO.

To display the schema is one step towards modifiable schemas in the network model. The problem of updating a schema is examined at all three levels in a database architecture: the external, the conceptual, and the internal level.

The display function produces two representations of the schema: the data description and data storage description languages (DDL and DSDL), and a graphical display of the schema graph together with interactive tools such as selecting records and zooming.

The DDL and DSDL follow the standards set by the current implementation of their compiler.

The graph of the schema is produced via a topological-sort-type algorithm which orders the nodes in the graph by their placement in the partial order. The

layout of the directed graph is optimized by reducing the number of crossings. The graph of the network schema is displayed on a graphical display terminal.

The Display Function in ALLEGRO

by

Raimund KH Ege

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Completed July 10, 1984

Commencement June 1985

APPROVED:

Redacted for Privacy

Assistant Professor of Computer Science in charge of major

Redacted for Privacy

Head of Department of Computer Science

Redacted for Privacy

Dean of Graduate School

Date thesis presented: July 10, 1984

Typed by Raimund KH Ege for himself.

TABLE OF CONTENTS

LIST OF FIGURES

The Display Function in ALLEGRO

CHAPTER I

INTRODUCTION

A. Purpose

ALLEGRO means being brisk in tempo. ALLEGRO is also the name of a network database system serving as a tool in the investigation of database system architecture. Currently it exists as a kernel which is described in [17], [16].

The short term goal is to enhance ALLEGRO by adding different functions such as user interface, compiler for Data Description and Data Storage Description Language, and displaying the schema.

The long term research goal is to develop a system which will include dynamic data description language, multiple data models, and an external relational user interface.

The implementation of the display function is one step towards the concept of modifiable schemas. This is a basic function every database administrator has to use in order

to start any modification and in updating the network schema. In the current stage the display function can produce the data description language, which can then be changed and reentered into the system by means of the compiler.

B. Scope

This thesis provides one step in the direction to make modifiable schemas possible. It introduces the display function as proposed by the Conference on Data Systems Languages (CODASYL) [4] Data Description Language Committee (DDLC) in their Journal of Development 1981.

This function provides the user with the original data description, the data storage description language and a graphical display of the network structure. The graphical part also provides means to interactively select and zoom the schema to get better visual results.

C. Current Research Objectives

It is the goal of a database management system to be able to model the user's world as adequately as possible. Two data models seem to remain the choice for a database designer: The relational model and the network model.

There has been an ongoing dispute about which of the

two models is better and has the chance to emerge into a standard for the future [13], [14].

The network model has become a "quasi-standard" due to the endorsement by CODASYL [4]. Because of its reasonable performance it has been adopted widely. The research on System R [2] and INGRES [15] has shown that a relational system can perform acceptably well in production use.

So now the dispute focuses on the quality of the two data models. The relational model seems to be simpler while the network model gives the database designer the control and performance tools he needs. Both will endure as "standards".

Stonebraker [14] argues against the CODASYL proposal for a data model standard. He says that currently changes in the network schema require that the applications have to be extensively modified at high cost because the data manipulation language is of very low level. He states that the CODASYL proposal is not likely to support multiple data models, but it should be possible to support network, hierarchical and relational views as user schemas. He demands that the semantic information of the set is to be separated from the data and stored openly.

If, for example, we want to build a distributed information system consisting of many different kinds of databases it is necessary to allow the coexistence of the two models. Therefore we need the same level of data representation and updatability on all sites of the system [18].

Changing the database schema without unloading and reloading the database is possible in most of the current relational systems [9]. If the network model wants to take part in this it has to provide means to maintain its schemas.

Recent studies, to extend the existing theory of the relational environment to the network model, try to translate network schemas into equivalent relational schemas and to apply existing relational theorems [6].

Another possibility is to build another system on top of the conceptual level in order to provide a uniform global interface to a heterogeneous collection of local databases. Katz [10] uses an entity relationship approach to derive equivalent schemas for the network and relational systems. Two of the main problems which are encountered when comparing the two models are: first, it is not possible to create temporary schema objects such as the results of selection or projection; second, joins are sometimes not necessary because they may be prestored in the database as CODASYL sets.

Multiple views on a database are the key issue if we want to implement a common interface to several database systems, or the transference of both databases and programs between systems which are defined in terms of different data models [3].

So there are two possible configurations for the system of the future [7]:

- A canonical system which is based on a canonical model capable of supporting CODASYL, relational or other models as user views through appropriate user schemas and access languages.
- A heterogeneous collection of different types of databases at the sites throughout the system where the features must work together and conversion must take place which will consume a lot of time.

As in [9] we come to the conclusion that the importance of the database management system's ability to model the user's world adequately cannot be overstated. It is possible to use the two different views of the database at the same time. The best of both worlds is at hand when we are able to provide relational query interfaces and even relational update programming interfaces to network-based database management systems. This then gives the database administrator the control and performance tool he needs while maintaining a simpler view for the end user and programmer. This is also the ultimate goal for ALLEGRO.

CHAPTER II

UPDATABILITY OF NETWORK SCHEMAS

If we now start to talk about the updatability of the schema of a network-type database, we first have to introduce our notion of a general network model.

A. A General Network Model

The way of looking at the model discussed here is analogous to some ideas presented by Kent [11] and Dayal and Bernstein [6]. The basic idea is that we look at two different structures of the network. One is the schema, the other is the extension or the occurrence of the underlying database structure for a specific application.

The ALLEGRO system is based on the so called object schema. The object schema has four record types: the SCHEMA record, the RECORD record, the FIELD record, and the SET record. It has three sets: the SCHEMA record owns the SCHEMA-RECORD set with member RECORD records, and the SCHEMA-SET set with member SET records. The RECORD record owns the RECORD-FIELD set with member FIELD records.

As we discussed above, this system deals with two kinds of objects: records and sets. Records represent en-

tities with certain attributes which are the fields. Sets are links or relationships between them.

In our generalization we represent records by nodes in a directed graph, and sets by edges between them. The data structure we get is called a "schema graph" [6]. Each node has a label which specifies the name of the record type and the attributes (fields) associated with it. Each edge in the graph has a label which is the link type.

Figure 1 shows a schema graph for the OBJECT_SCHEMA. The attributes (fields) of the records are not included in the figure. They are described in [17].

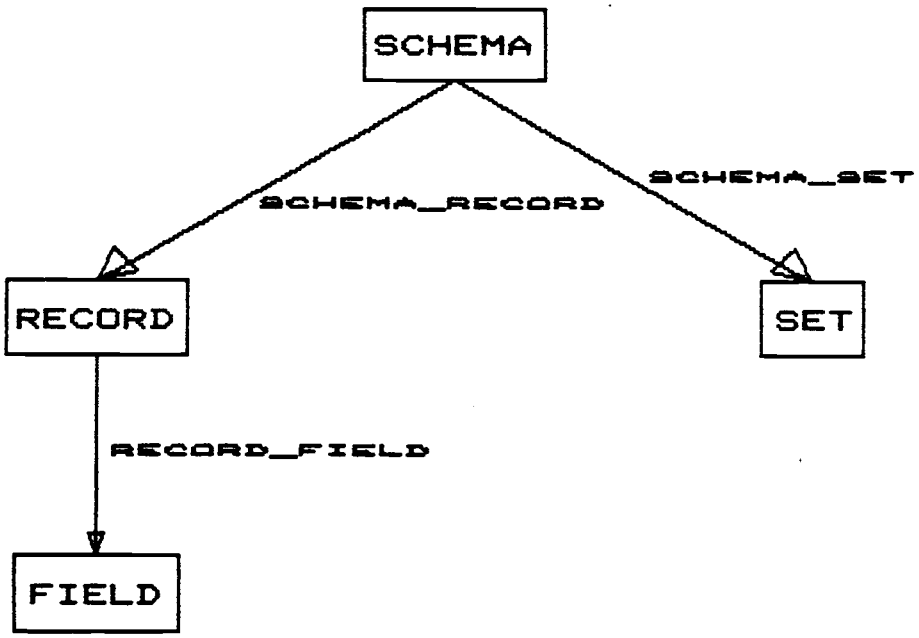

Figure 1: The Schema Graph of the OBJECT_SCHEMA

The nodes in this graph are not elementary items. They are record types which are partitioned into classes that share the same data structure and interpretation. The edges in this graph represent relationships between record types. They are directed from the owner to the members of the set.

An extension of the schema graph is given by the occurrence graph. This is also a directed graph where the nodes represent the actual occurrences of a record type and the edges represent the occurrences of the set types. Each node is labelled with its type and name, each edge is labelled with its type.

Figure 2 shows the schema graph for the sample database schema BOOK_AUTHOR which is described also in [17]. (The fields are again not included to preserve clarity.)

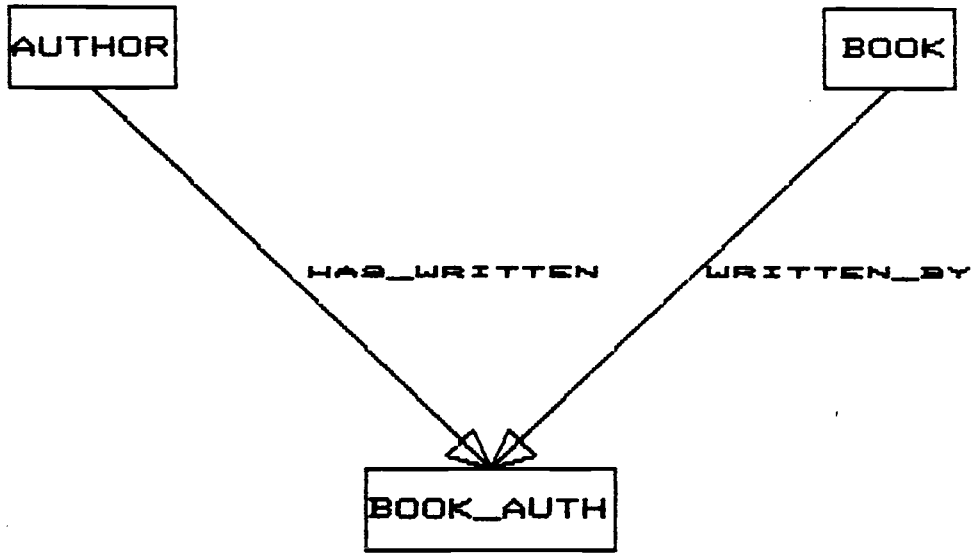

Figure 2: The Schema Graph of BOOK_AUTHOR

Combining the schema graph for the object schema of ALLEGRO with the actual schema BOOK_AUTHOR yields an occurrence graph which is shown in figure 3.

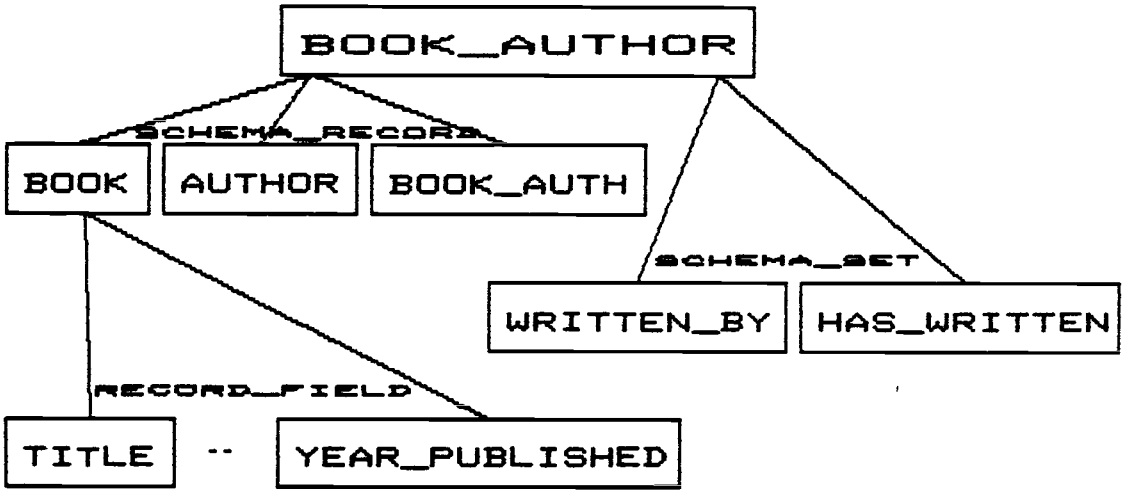

Figure 3: The Occurrence Graph for the BOOK_AUTHOR Schema

Many edges in the graph bear the same label. The set consists of all records connected by edges bearing the same label. The edges are directed from the owners to the members of the sets. A set occurrence consists of one owner record, and all the members to which it is connected by edges labelled with the same name.

All edges with the same label must emanate from nodes of the same type. A node may not be the destination of two edges with the same label.

B. Updating the Schema

To apply updates to the schema we now look at two representations of the schema at the conceptual level:

1. The schema graph.
2. An occurrence graph for the OBJECT_SCHEMA.

For example, to modify the BOOK_AUTHOR schema, we have to look at two representations.

1. The schema graph for the BOOK_AUTHOR schema (see Figure 2).
2. The occurrence graph for the occurrences of the BOOK_AUTHOR schema information in the OBJECT_SCHEMA schema (see Figure 3).

The user will be concerned with the schema graph. Most likely there are two broad classes of change that can occur: either growth (additions to the schema), or restructuring of the conceptual level. So the database administrator will add fields to a record, add records, add sets, delete fields from a record, delete records, or delete sets. Sets can be added if the corresponding records ex-

ist. Fields can be added to or removed from existing records.

The first thing to do to enable a DBA to interactively modify a schema is to display this schema graph on a graphical device to show the structure of the system. Then we have to provide tools to modify the structure, such as features which allow one to place records in the graph, to specify their attributes, to connect them with sets, to remove sets, and to remove records. Adding and deleting record fields could be done on the same screen, but should not appear in the graphical display because they do not influence the structure of the network.

The changes made in the schema graph must be applied to the occurrence graph. Records which are added will create a new instance of the RECORD record of the OBJECT_SCHEMA schema. New sets will create an instance of the SET record. The fields of the SET record are derived from the structure of the schema graph for the modified schema. The SET record contains information about owner, member, etc. which is described in [17].

The new FIELD records are added, as specified by the database administrator, for the appropriate records. The attributes for the RECORD and FIELD records are computed from the information about the structure, and the specifications of the database administrator.

Then we obtain a new occurrence graph for the occurrence of the schema information in the object schema,

that is we have modified the schema. But this is not a stand-alone operation. We now have to consider the influence of the modification on the user interface, the external level, and the impact on the internal level, i.e. the physical storage.

Therefore, now that we have changed the conceptual level, let us look at the problems that will arise at the external level and at the internal level.

C. Influence on the External Level

The external level will be defined in terms of subschemas for the given schema as far as a network user interface is concerned. For a relational user interface, view definitions have to map the external view onto the schema at the conceptual level. The two main modification needs, such as growth or restructuring of the schema, can be handled by just changing the mappings from the external to the conceptual level.

In the current state of the system ALLEGRO there is no subschema mechanism. As an user interface the CODASYL Data Manipulation Language (DML) is used. In this case, after modification of the schema, the programs basically have to be rewritten. Even for parts of the schema which are not changed we probably have to recompile the programs. So a subschema mechanism is a desirable enhancement for ALLEGRO.

D. Influence on the Internal Level

In contrast to the external level, modifying the schema has significantly greater impact on the physical storage in this network-type database. The structure of the network is not only present in the data definition but also in the data itself. The actual records, physically stored, contain pointers which connect them to other set members and the owner. So if we change the structure, i.e. remove and add a set type, we have to relink all the set members and owners. This involves a lot of run time and should be done by unloading and reloading the database.

The actual computation which has to be done depends on the implementation of the network database system. One possibility would be to implement a facility which could relink the records incrementally whenever they are used. We then would have to keep versions of all records in order to distinguish between records which are new, that is which are already updated to conform with the new schema, and old records which follow the old schema definition. We also have to keep track of the different versions of the schemas. Old records which have not yet been converted to the new structure have to accessed via the old schema. New records have to be accessed via the new schema.

Another problem is that much of the set connection is

time dependent, that is it depends on the sequence of events prior to the CONNECT (STORE). A significant change in the schema may add semantic information to a new relationship that cannot be inferred from the structure alone.

To solve this problem is beyond the scope of this thesis.

CHAPTER III

DISPLAYING THE SCHEMA IN ALLEGRO

As we stated earlier one step towards modifying a schema is to create some tool which retrieves the current schema of the system. To do this we have to retrieve the structure from the system and construct forms of it which the user can understand.

One possibility, obviously, is to disassemble or decompile the internal representation into the original Data Description and Data Storage Description Languages.

Another possibility is to use graphics and plot the schema graph of the conceptual level which we described in chapter two.

To help the user, as well as improving the usability of this graphics approach, we also provide means of selecting records and zooming the entire graph. So it is possible to handle large structures by concentrating on specific portions of the network.

A. Storage Organization in ALLEGRO

The ALLEGRO network database system involves two files. The first file contains the object schema; the second file contains the application database.

During normal operation of the system, the schema is first loaded into main memory before any navigation or access can be performed. The records of the object schema are stored in the same way as the records of the application. They follow the schema definition of OBJECT_SCHEMA which we described in section II.A.

The process of loading the OBJECT_SCHEMA definition is part of the OPENDB command of ALLEGRO. In loading the schema definitions, the system uses the same FIND and GET commands it uses to retrieve the records in the application database. It treats the schema definitions as instances in a database called the schema database. This whole process is combined into the "loadschema" function of ALLEGRO and is described in [17].

After loading, the schema definition for the application database is stored in main memory as an occurrence of the object schema.

B. Data Description and Data Storage Description Language

In their 1981 Journal of Development [4], the CODASYL Data Description Language Committee (DDLC) proposes to divide the description of the database into two parts.

The "conceptual view" of the network is defined by the SCHEMA in terms of the Data Description Language. It consists essentially of definitions of the various types of records in the database, the data-items they contain, and the sets into which they are grouped.

The "internal view", the storage structure of the database is described by the STORAGE SCHEMA written in the Data Storage Description Language.

The syntax of the Data Description Language used in ALLEGRO has been modified slightly to be consistent with the style of the programing language C. An example of the DDL for the BOOK_AUTHOR schema is given in Figure 4.

The Data Storage Description Language deals with the actual storage of the database. It contains storage record types which represent the record types defined at the schema level. It specifies how the record types are mapped onto storage records, how they are placed in the storage area and how they are accessed. The set types declare pointer chains or indices between storage records.

An example of the DSDL is given for the sample

BOOK_AUTHOR schema in Figure 5. Compilers for the Data Description and the Data Storage Description Language are currently under development.

```
SCHEMA NAME IS BOOK_AUTHOR;

   RECORD NAME IS BOOK;
          KEY IMMATERIAL title DUPLICATES NOT ALLOWED;
      title: CHARACTER;
      ISBN: CHARACTER;
      number_of_pages: INTEGER;
      year_published: SHORT INTEGER;

   RECORD NAME IS AUTHOR;
          KEY IMMATERIAL name DUPLICATES NOT ALLOWED;
      name: CHARACTER;
      affiliation: CHARACTER;

   RECORD NAME IS BOOK_AUTH;
          DUPLICATES ALLOWED;
      author_status: CHARACTER;

   SET NAME IS WRITTEN_BY;
      OWNER IS BOOK;
         ORDER FOR INSERTION IS IMMATERIAL;
      MEMBER IS BOOK_AUTH;
         INSERTION IS AUTOMATIC RETENTION IS FIXED
   SET NAME IS HAS_WRITTEN;
      OWNER IS AUTHOR;
         ORDER FOR INSERTION IS IMMATERIAL;
      MEMBER IS BOOK_AUTH;
         INSERTION IS MANUAL RETENTION IS FIXED;
```

Figure 4: Data Description Language for Schema BOOK_AUTHOR

```
STORAGE SCHEMA IS BOOK_AUTHOR FOR BOOK_AUTHOR;

   STORAGE RECORD NAME IS BOOK
           PLACEMENT IS CALC
           USING PREDEFINED HASH FUNCTION;
      title: CHARACTER;
         SIZE IS 80 CHARACTERS;
      ISBN: CHARACTER;
         SIZE IS 16 CHARACTERS;
      number_of_pages: INTEGER;
         SIZE IS 4 CHARACTERS;
      year_published: SHORT INTEGER;
         SIZE IS 2 CHARACTERS;

   STORAGE RECORD NAME IS AUTHOR
           PLACEMENT IS CALC
           USING PREDEFINED HASH FUNCTION;
      name: CHARACTER;
         SIZE IS 24 CHARACTERS;
      affiliation: CHARACTER;
         SIZE IS 40 CHARACTERS;

   STORAGE RECORD NAME IS BOOK_AUTH
           PLACEMENT IS CLUSTERED VIA SET WRITTEN_BY;
      author_status: CHARACTER;
         SIZE IS 1 CHARACTERS;

   SET NAME IS WRITTEN_BY;
      OWNER STORAGE RECORD IS BOOK;
         POINTER FOR PRIOR NEXT MEMBER;
      MEMBER RECORD IS BOOK_AUTH;
         STORAGE RECORD IS BOOK_AUTH;
         POINTER FOR OWNER PRIOR NEXT TENANT;

   SET NAME IS HAS_WRITTEN;
      OWNER STORAGE RECORD IS AUTHOR;
         POINTER FOR PRIOR NEXT MEMBER;
      MEMBER RECORD IS BOOK_AUTH;
         STORAGE RECORD IS BOOK_AUTH;
         POINTER FOR OWNER PRIOR NEXT TENANT;
```

Figure 5: Data Storage Description Language for Schema BOOK_AUTHOR

C. Graphics

To display the structure of the network, we first have to convert the internal schema representation into a storage structure which does not include the implementation details of the system but only the relations between the records.

1. Internal Representation

The relations between the record types are the sets which consist of owners and members. Such a relationship can be modelled by a directed graph. The nodes of the directed graph represent the record types; the directed edges represent the set types. An arc is an ordered pair of nodes which are connected by an edge, where the tail models the owner and the head models the member of the corresponding set.

Several data structures can be used to represent directed graphs. The appropriate choice of the data structure depends on the available storage and the operations that will be applied to the edges and nodes of the directed graph. Some basic representations are described by Aho, Hopcroft and Ullman [1].

We use the so-called "adjacency list" representation. It has the advantage of requiring less storage, which is proportional to the sum of the number of record types plus the number of set types. The adjacency list for a record type is a list, in some order, of all record types which belong to sets of which it is the owner. We can represent the directed graph of our network structure by an array of all record types where each contains a pointer to the sets it is the owner of.

Figure 6 shows the adjacency list representation of the schema graph for the schema BOOK_AUTHOR of Figure 2. The actual implementation of the adjacency list contains more information such as how many edges go out or come into this node, pointers to the actual records in the basic original record array of the schema representation in ALLEGRO, and pointers to the elements in the graph structure which is described in section III.C.2.

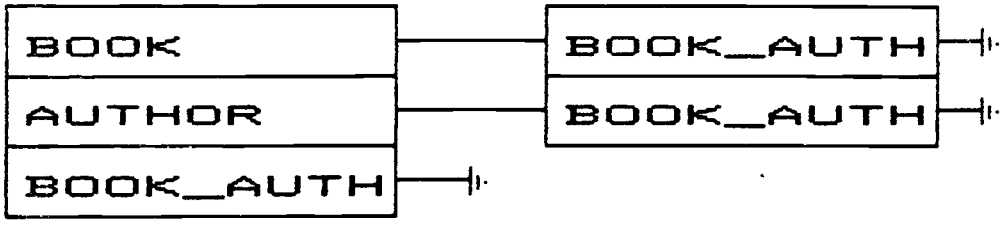

Figure 6: Adjacency List for Schema BOOK_AUTHOR

The adjacency list representation is built dynamically on the basis of the array for the record types. The adjacent records are then computed by scanning the set array which also has been loaded into main memory by the "loadschema" function of the ALLEGRO system as mentioned in section III.A.

2. Building the Graph

The adjacency list representation of the directed graph of the schema structure is just a vehicle in creating the ability to apply algorithms to produce a reasonable layout of the graph.

The basic idea we use in the algorithm is grouping the nodes according to the number of edges which go into them, i.e. the number of sets they belong to as member records. The algorithm follows the rudiments of "topological sorting" which is described by Knuth in [12].

In this algorithm the nodes are ranked according to their indegree. The indegree is defined as the number of incoming edges in the directed graph. Starting with the node which has the lowest indegree (probably it is equal to zero) the nodes are grouped into classes of the same rank. After one class has been established all the outgoing edges of these nodes are removed from the graph thus decreasing the indegrees of the other nodes. Then we look again at the nodes with the lowest indegree.

This sorting yields an ordered sequence of nodes which belong to the same rank. The nodes of the same rank are inserted in the internal representation of the graph all at the same level. The nodes of rank zero, which are the first ones if there is no cycle, are inserted at the top

level, all other nodes according to their rank in the partial order of the directed graph. Nodes which come later in the sorting sequence are inserted at lower levels. Nodes of the same rank belong to the same level in the internal graph.

This algorithm only works if there are no cycles in the directed graph. But in the schema for a network database system cycles are valid. So if the algorithm detects a cycle, that is there is no other node with indegree zero, it removes that edge from the graph which closes the loop. To find the best edge to remove it looks at the node with the lowest indegree and removes its incoming edges. These edges are stored separately and inserted in the graph at the time when the graph is plotted.

The internal representation also contains the coordinates for the screen and is referenced by the adjacency list for all the records. The graph in its internal representation is shown in Figure 7 for the schema BOOK_AUTHOR. How the nodes are placed within the appropriate level is discussed in the next subsection of this section.

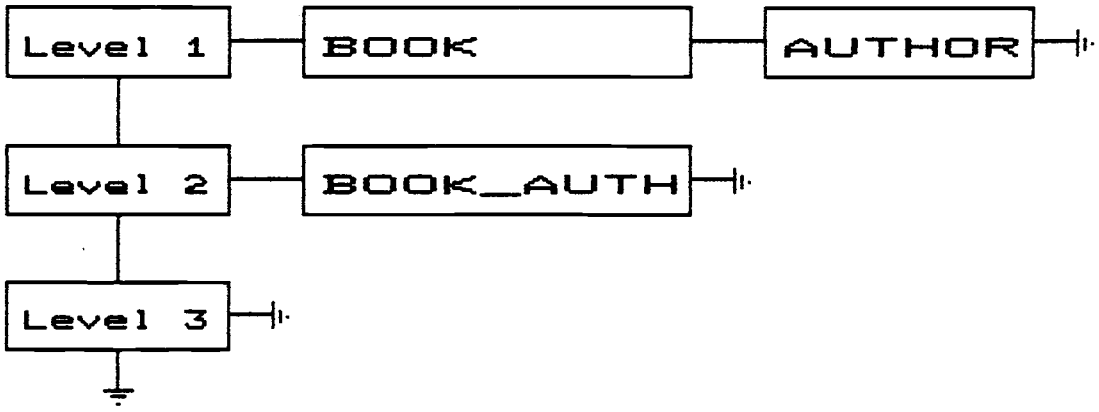

Figure 7: Graph in Internal Representation for Schema BOOK_AUTHOR

3. Optimizing the Layout

When entering a node at a certain level in the graph we have to consider the layout of the graph. We want to avoid crossings of edges which represent the sets. We also want to balance the graph, i.e. center the nodes with regard to their incoming and outgoing edges.

To avoid crossings for the edges has priority, therefore the system calculates the number of crossings which

are introduced when the node is inserted at any position in the level. It is then inserted at the position where the least crossings occur. If there is more than one possible position then the system tries to balance the nodes according to the number of outgoing and incoming edges.

After the nodes have been inserted, the system tries to put dependent members directly under their owners in case the owner has just a single (dependent) member. To accomplish this, dummy elements are included in the graph which are later ignored by the plot routines.

An example for a graph in which this occurs is shown in Figure 1 where the FIELD node was placed under the RECORD node rather than centered at its level.

4. Plotting the Graph

After all optimizing is done and the layout is manifested by the internal representation of the graph, the system computes the coordinates of all nodes by knowing the depth and width of the graph. The nodes are placed in equal distance to each other.

Then the plot procedure draws the graph based on the adjacency list structure for all record types. Boxes are drawn for all the record types and connected by lines for the sets. The lines are labelled by the set names that are retrieved from the set array (which still resides in main

memory).

Figure 1 and 2 show graphs for the two schemas OBJECT_SCHEMA and BOOK_AUTHOR.

The actual plotting is done on the GIGI graphics terminal which is accessed via the procedures from a terminal driver given in Appendix B. The exact description of the features of the GIGI graphics terminal can be found in [8]. The plot procedures follow the standards for the plotfile interface of the UNIX operating system. Therefore it is possible to plot the graphs on other terminals other than the GIGI by supplying the appropriate terminal driver and loading it with the program for the display function.

D. Interaction

To enhance the graphical display the system provides two functions to help the user. One is the possibility of selecting records. The other is the possibility of zooming the graph. The functions can be combined to get better visual results.

1. Selecting Records

To get better graphs, especially when the network is very large, this system provides the tool to select records. If the user gives the name of a specific record the system reduces the graph to all records which either are members in sets where the selected record is the owner, or are owners of sets where the selected record is a member.

Figure 8 shows the portion of the OBJECT_SCHEMA schema graph (see Figure 1) obtained when RECORD record has been selected.

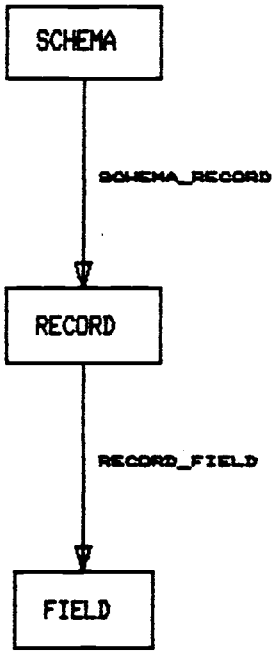

Figure 8: Schema Graph of Selected record RECORD in Schema OBJECT_SCHEMA

2. Zooming

For additional user friendliness zooming is possible. The system will prompt the user to give a zooming factor and to select the center for zooming.

The zooming factor can be any positive real number. Zero indicates that no zooming should take place. Numbers less than one will yield a reduction of the graph. Numbers greater than one will yield an enlargement of the graph. The text will also be enlarged, therefore a reasonable limit for the zooming factor is eight, because the largest text size available is sixteen and the text size for zooming factor one is two. For the same reason it is better to specify the zooming factors in steps of 0.5 because the text and the graph will stay in the same size relation.

The center for zooming can be selected when the terminal is in graphics locator mode. In this mode the locator cursor appears on the terminal. It is a large cross-hair which can be moved with the cursor moving keys (arrow keys) of the keyboard. Figure 9 shows the schema graph of Figure 1 plus the prompt to select a center on the GIGI graphics terminal. Figure 10 shows the same structure after zooming with factor 7 at the center SCHEMA.

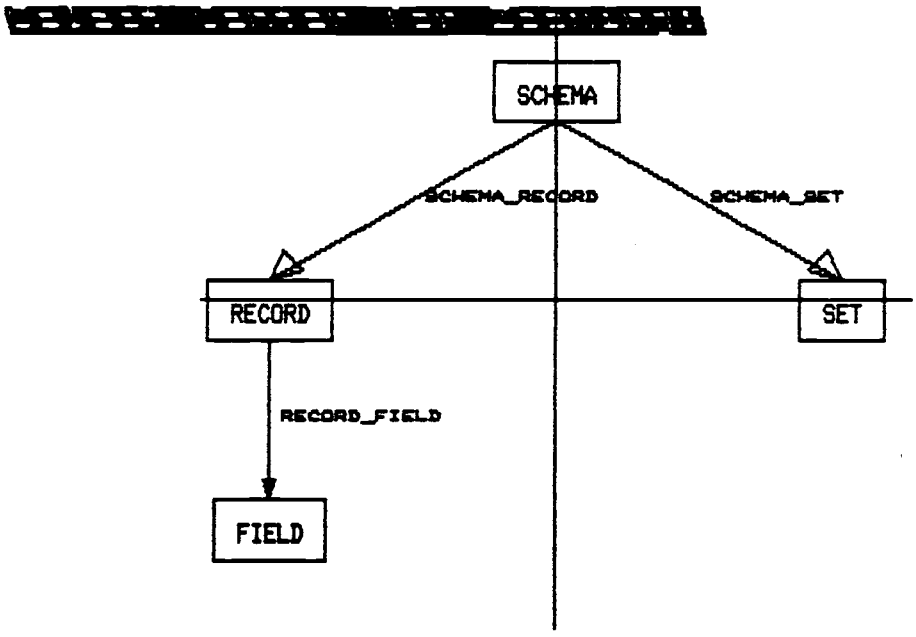

Figure 9: Schema Graph of OBJECT_SCHEMA with Zooming Prompt

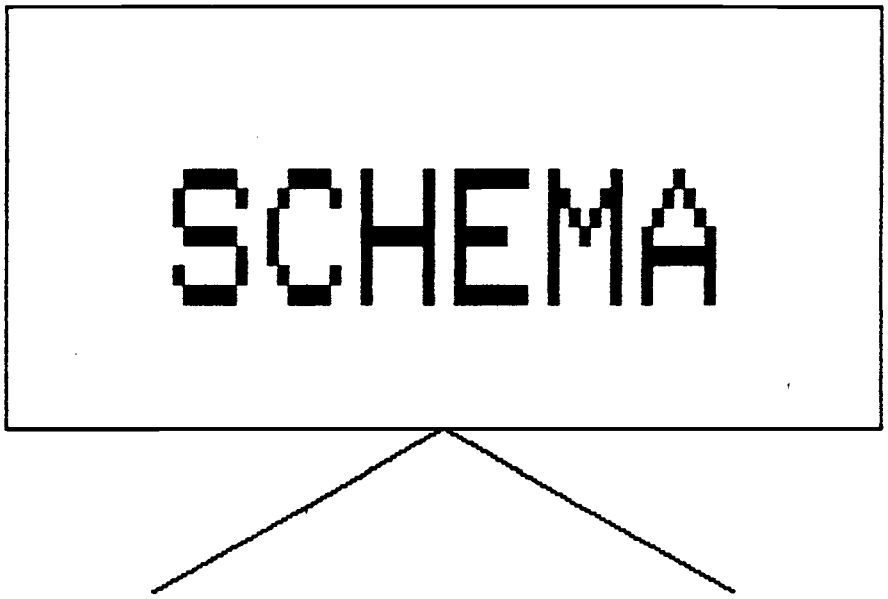

Figure 10: Schema Graph of OBJECT_SCHEMA after Zooming

The two functions of selecting records and zooming the graph can be combined. This will result in a graph which is restricted to certain records as described in section III.D.1, has the selected record in the center, and is zoomed by the specified zooming factor.

E. Invoking the Display Function

The commands which will invoke the display function of ALLEGRO are: "DISPLAY", "display", "select", "zoom", and "selzoom". "DISPLAY" invokes the function with all its capabilities as described in this thesis. The commands "display", "select", "zoom", and "selzoom" provide subparts of the function for a convenient user interface. There are two modes of operation. One can type the command with all arguments or one can omit them. The system will then prompt the user for the missing arguments.

Synopsis:

```
DISPLAY   <schema-name>
          <select-record>
          <function-type>
          <zooming-factor>
          <foreground-color>
          <background-color>

display   <schema-name> <function-type>

select    <schema-name> <select-record>

zoom      <schema-name> <zooming-factor>

selzoom   <schema-name> <select-record> <zooming-factor>
```

Description of the arguments:

<schema-name> : name of an existing schema in the file "objectschema".

<select-record> : name of a record in the schema of argument one, which is to be selected.

<function-type> : type of function to be performed:
Valid options are:
1 = print DDL
2 = print DSDL
3 = plot graph on terminal vt100
4 = plot graph on lineprinter
5 = plot graph in plotfile format.

<zooming-factor> : real positive number by which the graph is to be zoomed.

<foreground-color>
<background-color> : integer number for the foreground and background colour for the GIGI terminal.
valid numbers are:
0 = black
1 = blue
2 = red
3 = magenta
4 = green
5 = cyan
6 = yellow
7 = white
Default: foreground: 0 (black)
background: 1 (blue).

CHAPTER IV

CONCLUSION

In the introduction to this thesis we stated that the short term goal in enhancing the system ALLEGRO is to add different functions. These functions were:

User interface,
compiler for DDL and DSDL,
and displaying the schema for the network.

The latter function is now implemented with several tools to improve the handling of the functions.

The next step towards the future could be the implementation of a schema editor in a way which we described in section II.B. This schema editor could produce the DDL and DSDL. The database administrator then could reenter these languages via their compiler into the system. This would not create modifiablity of the schema of a running network database system, but this function could at least help to create new schemas.

In section II.C, where we talked about the influence of updating the schema on the external level, we came to the conclusion that without a subschema mechanism it is not possible to support modifiable schemas. So this should be the next enhancement for ALLEGRO.

The long term research goal remains the same, that is to develop a system which will include a dynamic Data Description Language facility, will support multiple data models, and will provide an external relational user interface.

In section I.C we discussed the dispute between the network and the relational model for database systems. Which one will survive ? Both of them exist as quasi-standards in many applications throughout the world. It may be that none of them will be the system of the future.

The system of the next generation must be capable of features such as:

- dynamic schema evolution without unloading and reloading the whole database system.
- ability to handle complex data and large objects, such as structured data, text, voice, images, etc.
- ability to manage objects and time explicitly. The concept of time can be part of the data model.
- supporting different views of data.
- handling and insuring integrity constraints.
- data security.

Many of these issues are not included in the state of the art database systems of either data model. It may be that another model based on a semantic model approach will be the system of the future.

BIBLIOGRAPHY

[1] Aho, V., Hopcroft, J., Ullman, J.: Data Structures and Algorithms , Addison-Wesley Publishing Company, Reading, Massachusetts 1983.

[2] Astrahan, M. et al.: System R - A Relational Approach to Database Management , ACM Transactions on Database Systems, Volume 1, No 2, 1976.

[3] Borkin, S.: Data Models , The MIT Press, Cambridge, Massachusetts 1980.

[4] CODASYL: Data Definition Language Committee Journal of Development , Secretariat of the Canadian Government EDP Standards Commitee, Ottawa, Canada 1981.

[5] Date, C.: An Introduction to Database Systems , Addison-Wesley Publishing Company, Reading, Massachusetts 1981.

[6] Dayal, U., Bernstein, P.: On the Updatability of Network Views - Extending Relational View Theory to the Network Model , in: Information Systems, Vol 7, No 1, pp. 29-46, Pergamon Press Limited, 1982

[7] Deen, S.: Distributed Databases: Some Problems , Proceedings International Conference on Databases, British Computer Society Workshop Series, Aberdeen 1980.

[8] Digital Equipment Corporation GIGI/ReGIS Handbook , Order Number AA-K336A-TK, Maynard, Massachusetts 1981.

[9] Gradwell, D. (ed): Database - The 2nd Generation , State of the Art Report, Series 10, Number 7, Pergamon Infotech Limited, Maidenhead, Berkshire, England 1982.

[10] Katz, R.: Compilation of Relational Queries into CODASYL DML , in: Scheuermann, P. (ed): Improving Database Usability and Responiveness , Academic Press, New York 1982.

[11] Kent, W.: Data and Reality , North Holland Publishing Company, New York 1978.

[12] Knuth, D.: The Art of Computer Programming, Volume 1, Fundamental Algorithms , Addison-Wesley Publishing Company, Reading, Massachusetts 1973.

[13] Rustin, R. (ed): Data Models: Data-Structure-Set versus Relational , Workshop on Data Description Access and Control, ACM 1975.

[14] Stonebraker, M.: The Argument against CODASYL , in: Draffan, Poole (ed) : Distributed Databases. Cambridge University Press, New York 1980.

[15] Stonebraker, M., Wong, E., Kreps, P., Held, G.: The Design and Implementation of INGRES , ACM Transactions on Database Systems, Volume 1, No 3, 1976.

[16] Sullivan, D.: Enhancements for ALLEGRO , Project Report, Oregon State University 1984.

[17] Uy, Myra Lane: A Network Data Base Management System , M.S. Thesis, Oregon State University 1983.

[18] Wong, E., Katz, R.: Logical Design and Schema Conversion for Relational and DBTG Databases , in: Chen, P. (ed): Entity-Relationship-Approach to Systems Analysis and Design. North Holland Publishing Company, New York 1980.

APPENDIX

A. The Source Code

```
/*************************************************************/
/*                                                           */
/*       display                                             */
/*          mainprogram of ray's thesis                      */
/*                                                           */
/*************************************************************/

#include <stdio.h>
#include "global"
#include "local"

long status;
int i,j,pswitch;
char c;
char answer [] = { '1', '2' };

main(argc, argv)
int argc;
char *argv[];
{
   strncpy (a_schema.name,
            "                                ",32);

   if (argc < 2)
   {
       prints("What schema do you want to look at? ");
       scanf ("%s",a_schema.name);
   }
   else
       strncpy(a_schema.name, argv[1], strlen(argv[1]));

   for ( i = 0; i < 32; i++)
       if (a_schema.name[i] == ' ')
           a_schema.name[i] = ' ';


/* set up the objectschema */

   schema = &o_schema;
   record = o_record;
   set    = o_set;
   field  = o_field;

   fd = open("objectschema",0);
   readblk (fd, buf, 0);
   copyseg (buf, &system, sizeof(system));
```

```
ldhashinfo ();

loadschema (a_schema.name);
close (fd);

strncpy (select_record,
         "                                ",32);

if (argc < 3)
{
    prints("Do you want to select a record ? ");
    scanf ("%s",select_record);
}
else
    strncpy(select_record, argv[2], strlen(argv[2]));

select_mode = FALSE;
for ( i = 0; i < 32; i++) {
    if (select_record[i] == ' ')
        select_record[i] = ' ';
}
select_record[length(select_record)+1] = ' ';

for ( i = 0; i < a_schema.n_records; i++) {
    a_record[i].name[length(a_record[i].name)+1] = ' ';
    if ( !strcmp(select_record, a_record[i].name)) {
        select_number = i;
        select_mode = TRUE;
    }
}
if (argc < 5) {
    prints ("Do want to ZOOM (and how) ? ");
    scanf  ("%f",&zoom_mode);
}
else
    sscanf (argv[4], "%f", &zoom_mode);

if (argc < 6) foreground = 0;
else          sscanf (argv[5], "%d", &foreground);
if (argc < 7) background = 1;
else          sscanf (argv[6], "%d", &background);

if (argc > 3) *answer = *argv[3];
else
{
   prints ("You have the following options:0);
   prints ("- data definition language    = 10);
   prints ("- storage definition language = 20);
   prints ("- graph of networkstructure on device:0);
   prints ("     > printer  = 30);
   prints ("     > vt100    = 40);
   prints ("     > plotfile = 50);
```

```
      scanf ("%s",answer);
   }

   if ((*answer > '0') && (*answer < '6'))
      switch (*answer)
      {
         case '1' : ddl(); break;
         case '2' : dsdl(); break;
         default :
         {
            linepos = 0;
            crossings = 0;

            switch (*answer)
            {
               case '3' : horspace = 132;
                          verspace = 66;
                          lines = (short *)
               malloc (sizeof(*lines)*verspace*horspace);
                          break;
               case '4' : horspace = 78;
                          verspace = 24;
                          lines = (short *)
               malloc (sizeof(*lines)*verspace*horspace);
                          break;
               default:   horspace = 767;
                          verspace = 479;
                          luft = 15;
            }
            intergraph ();
            optgraph ();

            switch (*answer)
            {
               case '3' : fd = creat("picture",0755);
                          printgraph(); break;
               case '4' : fd = 1; printgraph(); break;
               case '5' : interaction(); break;
            }
         }
      }
} /* main */
```

```
/*********************************************************/
/*                                                       */
/*       ddl                                             */
/*          data definition language                     */
/*                                                       */
/*********************************************************/

#include <stdio.h>
#include "global"

long status;
int i,j,pswitch;
char c;
ddl()
{
   a_schema.name[length(a_schema.name)] = ' ';
   for (i = 0; i < a_schema.n_records; i++)
      a_record[i].name[length(a_record[i].name)] = ' ';
   for (i = 0; i < a_schema.n_fields; i++)
      a_field[i].name[length(a_field[i].name)] = ' ';
   for (i = 0; i < a_schema.n_set_records; i++)
      a_set[i].name[length(a_set[i].name)] = ' ';

   printf("SCHEMA NAME IS %s;0, a_schema.name);

   for ( i = 0; i < a_schema.n_records ; i++)
   {
      printf("0);
      printf("   RECORD NAME IS %s;0, a_record[i].name);

      pswitch = TRUE;
      for ( j = a_record[i].first_field;
            j < (a_record[i].n_fields +
                 a_record[i].first_field);
            j++)
      {
         if (a_field[j].pos_in_key != 0)
         {
            if (pswitch == TRUE)
            {
               printf("          KEY ");
               pswitch = FALSE;
            }

            switch (a_field[j].sort_order)
            {
               case 'a' : printf("ASCENDING");break;
               case 'd' : printf("DESCENDING");break;
               case 'i' : printf("IMMATERIAL");break;
               default  : printf("SYSTEM DEFAULT");break;
            }
            printf(" %s ", a_field[j].name);
         }
```

```
    }
    if (pswitch == TRUE)
       printf("          ");

    printf ("DUPLICATES ");
    switch (a_record[i].dup)
    {
       case 'a' : printf ("ALLOWED;0); break;
       case 'n' : printf ("NOT ALLOWED;0); break;
       default  : printf ("SYSTEM DEFAULT;0); break;
    }
    for ( j = a_record[i].first_field;
          j < (a_record[i].n_fields +
               a_record[i].first_field);
          j++)
    {
       if (strcmp(a_field[j].name, "token")) {
          printf("      %s: ", a_field[j].name);
          switch (a_field[j].data_type)
          {
             case 'c' : printf("CHARACTER"); break;
             case 'i' : printf("INTEGER"); break;
             case 'b' : printf("BOOLEAN"); break;
             case 'x' : printf("STRING"); break;
             case 'l' : printf("LONG INTEGER"); break;
             case 'f' : printf("FLOAT"); break;
             case 's' : printf("SHORT INTEGER"); break;
             default  : printf("SYSTEM DEFAULT"); break;
          }
          if (a_field[j].occurrence != 1)
             printf(" %d",a_field[j].occurrence);
          printf (";0);
       }
    }
}

for ( i = 0; i < a_schema.n_set_records ; i = i + 2)
{
    printf("0);
    printf("   SET NAME IS %s;0, a_set[i].name);
    printf("      OWNER IS %s;0,
           a_record[a_set[i].owner_token].name);
    printf("         ORDER FOR INSERTION IS ");
    switch (a_set[i].ins_order)
    {
       case 'i' : printf ("IMMATERIAL"); break;
       case 'f' : printf ("FIRST"); break;
       case 'l' : printf ("LAST"); break;
       case 'a' : printf ("NEXT"); break;
       case 'b' : printf ("PRIOR"); break;
       case 's' : printf ("SORTED BY DEFINED KEYS");
                  break;
       case 't' : printf ("SORTED WITHIN RECORD TYPES");
```

```
                    break;
            default  : printf ("SYSTEM DEFAULT");
        }
        printf (";0);
        printf ("      MEMBER IS %s;0,
                a_record[a_set[i+1].rec_token].name);
        printf ("          INSERTION IS ");
        switch (a_set[i+1].insertion_mode)
        {
            case 'a' : printf ("AUTOMATIC"); break;
            case 'm' : printf ("MANUAL"); break;
            case 'n' : printf ("NEXT"); break;
            default  : printf ("SYSTEM DEFAULT"); break;
        }
        printf (" RETENTION IS ");
        switch (a_set [i+1].retention_mode)
        {
            case 'f' : printf ("FIXED"); break;
            case 'o' : printf ("OPTIONAL"); break;
            case 'm' : printf ("MANDATORY"); break;
            default  : printf ("SYSTEM DEFAULT"); break;
        }
    }
    printf(";0);

    printf("0);

} /* data definition language */
```

```
/*********************************************************/
/*                                                       */
/*       dsdl                                            */
/*          Data storage definition language             */
/*                                                       */
/*********************************************************/

#include <stdio.h>
#include "global"

long status;
int i,j,pswitch;
char c;
dsdl()
{
   a_schema.name[length(a_schema.name)] = ' ';
   for (i = 0; i < a_schema.n_records; i++)
      a_record[i].name[length(a_record[i].name)] = ' ';
   for (i = 0; i < a_schema.n_fields; i++)
      a_field[i].name[length(a_field[i].name)] = ' ';
   for (i = 0; i < a_schema.n_set_records; i++)
      a_set[i].name[length(a_set[i].name)] = ' ';

   printf("STORAGE SCHEMA IS %s", a_schema.name);
   printf(" FOR %s;0,a_schema.name);

   for ( i = 0; i < a_schema.n_records ; i++)
   {
      printf ("0);
      printf("   STORAGE RECORD NAME IS %s0,
             a_record[i].name);
      printf("            PLACEMENT IS ");
      switch (a_record[i].loc_mode)
      {
         case 'c' :
         printf ("CALC USING PREDEFINED HASH FUNCTION");
         break;
         case 'v' : printf ("CLUSTERED VIA SET ");
         printf ("%s",a_set[a_record[i].set_token].name);
         break;
         default  : printf ("SYSTEM DEFAULT");
      }
      printf (";0);
      for ( j = a_record[i].first_field;
          j < (a_record[i].n_fields +
               a_record[i].first_field);
          j++)
      {
         if (strcmp(a_field[j].name, "token")) {
            printf("      %s: ", a_field[j].name);
            switch (a_field[j].data_type)
            {
               case 'c' : printf("CHARACTER"); break;
```

```
                case 'i' : printf("INTEGER"); break;
                case 'b' : printf("BOOLEAN"); break;
                case 'x' : printf("STRING"); break;
                case 'l' : printf("LONG INTEGER"); break;
                case 'f' : printf("FLOAT"); break;
                case 's' : printf("SHORT INTEGER"); break;
                default  : printf("SYSTEM DEFAULT"); break;
            }
            printf(";0);
            printf("          SIZE IS %d CHARACTERS;0,
                    a_field[j].length);
        }
    }
}

for ( i = 0; i < a_schema.n_set_records ; i = i + 2)
{
    printf("0);
    printf("   SET NAME IS %s;0, a_set[i].name);
    printf("      OWNER STORAGE RECORD IS %s;0,
           a_record[a_set[i].owner_token].name);
    printf("         POINTER FOR ");

    if (a_set[i].ptr_mode & 32)
    {
        printf ("PRIOR ");
    }
    if (a_set[i].ptr_mode & 64)
    {
        printf ("NEXT ");
    }

    printf ("MEMBER;0);

    printf ("      MEMBER RECORD IS %s;0,
            a_record[a_set[i+1].rec_token].name);
    printf ("         STORAGE RECORD IS %s;0,
            a_record[a_set[i+1].rec_token].name);
    printf ("         POINTER FOR ");

    if (a_set[i+1].ptr_mode & 128)
    {
        printf ("OWNER ");
    }
    if (a_set[i+1].ptr_mode & 32)
    {
        printf ("PRIOR ");
    }
    if (a_set[i+1].ptr_mode & 64)
    {
        printf ("NEXT ");
    }
    printf ("TENANT;0);
```

```
    }
} /* data storage definition language */
```

```
/*********************************************************/
/*                                                       */
/*     intergraph                                        */
/*        Manipulation of the internal datastructure     */
/*                                                       */
/*********************************************************/

#include <stdio.h>
#include "global"
#include "local"

int size;
int last_owner;
int i,j;
int low_indegree, low_index, low_cross;
struct element *vep, *nvep;
struct graphelement *vg;
struct firstelement *vf;
struct cycle *vcyc;
short newlevel;

intergraph()
{
   size = sizeof(*network);
   network = (struct ns * )
      malloc(a_schema.n_records * size);
   n_records = 0;

   for ( i = 0; i < a_schema.n_records; i++) {
      network[i].indegree = 0;
      network[i].outdegree = 0;
      network[i].ep = NULL;
      network[i].unused = TRUE;
      if (strcmp(select_record, a_record[i].name) &&
          select_mode)
          network[i].turn = FALSE;
      else {
          n_records++;
          network[i].turn = TRUE;
      }
   }

/* find all records which are directly
   connected to the selected record */

   if (select_mode)
      for ( i = 0; i < a_schema.n_set_records; i++) {
          if (a_set[i].n_mem_types > 0) {
             last_owner = a_set[i].owner_token;
          }
          if (a_set[i].n_mem_types == 0) {
             if (!strcmp(select_record,
                  a_record[last_owner].name)) {
```

```
                    if (!network[a_set[i].rec_token].turn)
                       n_records++;
                    network[a_set[i].rec_token].turn = TRUE;
                }
                if (!strcmp(select_record,
                     a_record[a_set[i].rec_token].name)) {
                    if (!network[a_set[i].rec_token].turn)
                       n_records++;
                    network[last_owner].turn = TRUE;
                }
            }
        }

/* convert from set-array to adjacency list structure  */

    for ( i = 0; i < a_schema.n_set_records; i++) {
        if (a_set[i].n_mem_types > 0) {
            last_owner = a_set[i].owner_token;
            network[a_set[i].owner_token].outdegree++;
        };
        if (a_set[i].n_mem_types == 0) {
            vep = network[last_owner].ep;
            nvep = (struct element *) malloc(sizeof(*vep));
            nvep->number = a_set[i].rec_token;
            nvep->set_token = i;
            nvep->ep = vep;
            network[last_owner].ep = nvep;
            if (network[nvep->number].turn &&
                network[last_owner].turn) {
                network[a_set[i].rec_token].indegree++;
                network[a_set[i].rec_token].res_indegree++;
            }
        }
    }

/* look for source and produce graph    */

    anchor = (struct firstelement *)
              malloc(sizeof(*anchor));
    anchor->depth = 0;
    anchor->width = 0;
    anchor->son = NULL;
    anchor->brother = NULL;
    level = anchor;
    cycles = NULL;
    n_found = 0;

    do {
        newlevel = FALSE;
        for ( i = 0; i < a_schema.n_records; i++) {

            if (network[i].indegree == 0 &&
               network[i].unused && network[i].turn) {
```

```
            network[i].unused = FALSE;
            newlevel = TRUE;
            vg = (struct graphelement *)
                  malloc(sizeof(*vg));
            vg->number = i;
            network[i].gp = vg;
            ingraph (vg);
            level->width = level->width + 1;
            n_found++;
         }
      }

      if (newlevel) {
         vg = level->brother;
         while ( vg != NULL ) {
            vep = network[vg->number].ep;
            while ( vep != NULL) {
               if (network[vep->number].turn)
                   network[vep->number].indegree--;
               vep = vep->ep;
            }
            vg = vg->brother;
         }
         vf = (struct firstelement *)
               malloc(sizeof(*anchor));
         level->son = vf;

         if (level == anchor) anchor->depth++;
         else {
            anchor->depth++;
            level->depth++;
         }
         vf->depth = 0;
         vf->width = 0;
         vf->brother = NULL;
         vf->son = NULL;
         level = vf;
      }
      else if (n_records > n_found) {

/*    there is a cycle in the graph   */

         newlevel = TRUE;

/*    find element with lowest indegree   */

         low_indegree = 999;

         for ( i = 0; i < a_schema.n_records; i++)
            if ( network[i].turn && network[i].unused &&
                 (network[i].indegree < low_indegree)) {
               low_indegree = network[i].indegree;
               low_index = i;
```

```
            }

/*    remove his incoming edges, thus reducing    */
/*    his indegree to zero                        */

         for ( i = 0; i < a_schema.n_records; i++)
            if (network[i].turn && network[i].unused) {
               vep = network[i].ep;
               while (vep != NULL) {
                  if (vep->number == low_index) {
                     network[low_index].indegree--;
                     vcyc = (struct cycle *)
                            malloc(sizeof(*cycles));
                     vcyc->set_token = vep->set_token;
                     vcyc->next = cycles;
                     cycles = vcyc;
                  }
                  vep = vep->ep;
               }
            }
      }
   }
   while (newlevel);

/*   try to reduce number of crossings */

   vf = anchor;
   for ( i = 0; ((i < anchor->depth) &&
         ((low_cross = numcross()) > 0)); i++) {

      do {
         vg = vf->brother;
         if ((vg != NULL) && (vg->brother != NULL)) {
            for ( j = 0; j < (vf->width - 2); j++)
               vg = vg->brother;
            vg->brother->brother = vf->brother;
            vf->brother = vg->brother;
            vg->brother = NULL;
            vg = vf->brother;
            for (j = 0; vg != NULL; vg = vg->brother)
               vg->coordinates[1] = j++;
         }
      }
      while (numcross() > low_cross);

      if (i == 0 ) anchor = vf;
      vf = vf->son;
   }
} /* intergraph */
```

```
/*********************************************************/
/*                                                       */
/*      ingraph                                          */
/*         procedure to insert a node into               */
/*         the internal graph                            */
/*                                                       */
/*********************************************************/

#include "global"
#include "local"

ingraph (node)
struct graphelement *node;
{
    struct graphelement *vg;
    int lcross, min, mip, max = 0;

    node->coordinates[1] = 0;
    node->coordinates[0] = anchor->depth;

    while (node->coordinates[1] <= level->width) {
        lcross = numcross () - crossings;
        if (node->coordinates[1] == 0) {
           min = lcross;
           mip = 0;
        }
        else if (lcross < min) {
           min = lcross;
           mip = node->coordinates[1];
        }
        if (lcross > max) max = lcross;
        node->coordinates[1] = node->coordinates[1] + 1;
    }
    node->coordinates[1] = mip;
    crossings = crossings + min;

/*  if there are no crossings at all then try to center */

    if (max == 0) {
        vg = level->brother;
        mip = 0;
        while ((vg != NULL) &&
               ((network[node->number].indegree +
                 network[node->number].outdegree) <
                (network[vg->number].indegree +
                 network[vg->number].outdegree))) {
            vg = vg->brother;
            mip++;
        }
    }

/*  position according to mip  */
```

```
    if (mip == 0) {
       node->brother = level->brother;
       level->brother = node;
    }
    else {
       vg = level->brother;
       for (lcross = 1;
            lcross < mip; lcross++) vg = vg->brother;
       node->brother = vg->brother;
       vg->brother = node;
    }

/*  recalculate the y-coordinates  */

    vg = level->brother;
    lcross = 0;
    while (vg != NULL) {
        vg->coordinates[l] = lcross;
        lcross++;
        vg = vg->brother;
    }
}    /* ingraph */
```

```
/*************************************************************/
/*                                                           */
/*    numcross                                               */
/*       procedure to calculate the number of crossings      */
/*                                                           */
/*************************************************************/

#include "global"
#include "local"

int numcross ()
{
    int first, sec, cross=0;
    struct element *fd, *sd;

    for (first = 0; first < a_schema.n_records; first++) {
       if ( !(network[first].unused) &&
             network[first].turn) {
          fd = network[first].ep;
          while (fd != NULL) {
             if ( !(network[fd->number].unused) &&
                   network[fd->number].turn) {
                for (sec = (first+ 1);
                     sec < a_schema.n_records;
                     sec++) {
                   if ( !(network[sec].unused) &&
                        network[sec].turn) {
                      sd = network[sec].ep;

                      while (sd != NULL) {
                         sd->number =
                         a_set[sd->set_token].rec_token;
                         if (!(network[sd->number].unused)
                            && network[sd->number].turn) {
                            if (linecross(
                              network[first].
                                 gp->coordinates,
                              network[fd->number].
                                 gp->coordinates,
                              network[sec].
                                 gp->coordinates,
                              network[sd->number].
                                 gp->coordinates))
                              cross++;
                         }
                         sd = sd->ep;
                      }
                   }
                }
             }
             fd = fd->ep;
          }
       }
```

```
    }
    return(cross);
}    /* numcross */
```

```
/*********************************************************/
/*                                                       */
/*      optgraph                                         */
/*         optimize graph layout                         */
/*         after internal coordinates                    */
/*                                                       */
/*********************************************************/

#include "global"
#include "local"


optgraph ()
{
    int c1[2], c2[2], c3[2], oi, oj;
    double point;
    struct firstelement *ogp;
    struct graphelement *ovg;
    struct element *vep;

/*  compute width of each level   */

    width = (int *) malloc(sizeof(oi)*(anchor->depth+1));
    ogp = anchor;
    max_width = 0;
    for (oi = 0; ogp != NULL; oi++) {
        width[oi] = ogp->width;
        if (width[oi] > max_width)
            max_width = width[oi];
        ogp = ogp->son;
    }

/*  try to move all directly depending sets */

    for (oi = 0; oi < a_schema.n_records; oi++) {
        if ( network[oi].turn &&
            (network[oi].outdegree == 1) &&
             network[network[oi].ep->number].turn &&
            (network[network[oi].ep->number].
             res_indegree == 1) ) {
            c1[0] = network[oi].gp->coordinates[0];
            c1[1] = network[oi].gp->coordinates[1];
            c2[0] = network[network[oi].ep->number].
                    gp->coordinates[0];
            c2[1] = network[network[oi].ep->number].
                    gp->coordinates[1];
            if (c1[1] != c2[1]) {
                if ((c1[1] > c2[1]) &&
                    (width[c1[0]] > width[c1[0]]))
                    insert_dummy (c2,c1[1]-c2[0]);
                if ((c1[1] < c2[1]) &&
                    (width[c1[0]] < width[c1[0]]))
                    insert_dummy (c1,c2[1]-c1[0]);
```

```
            }
            else if (width[c1[0]] > width[c2[0]])
               insert_dummy (c2,width[c2[0]] -
                             width[c1[0]]);
            else if (width[c1[0]] > width[c2[0]])
               insert_dummy (c1,width[c1[0]] -
                             width[c2[0]]);
         }
      }

/*  insert dummy-elements where set-lines cross levels */

      for (oi = 0; oi < a_schema.n_records; oi++) {
         if ( network[oi].turn ) {
            c1[0] = network[oi].gp->coordinates[0];
            c1[1] = network[oi].gp->coordinates[1];
            vep = network[oi].ep;
            while (vep != NULL) {
               if ( network[vep->number].turn ) {
                  c2[0] = network[vep->number].
                          gp->coordinates[0];
                  c2[1] = network[vep->number].
                          gp->coordinates[1];
                  if (c2[0] > (c1[0] + 1)) {
                     for (oj = c1[0] + 1;
                          oj < c2[0]; oj++) {
                        c3[0] = oj;

/*    c3[1] needs to be adjusted between the 2 points  */

                        point = (double)(c1[1]*(c2[0]-
                                 c1[0]-oi) +
                                 c2[1]*oj/(c2[0]-c1[0]));
                        c3[1] = point;

/*    but on which side should the dummy be inserted ? */
/*        neg.: to the right                           */
/*        pos.: to the left                            */
/*    lets move it to the middle                       */

                        if ((width[c3[0]] != 1)  ||
                            ((width[c1[0]] == 1) &&
                             (width[c2[0]] == 1))) {
                           if (point <
                              ((double)width[c3[0]]/2.0))
                              insert_dummy (c3,  1);
                           else
                              insert_dummy (c3, -1);
                        }
                     }
                  }
               }
               vep = vep->ep;
```

```
            }
        }
    }

/*  compute coordinates of all graphelements  */

    level = anchor;
    for (oi = 0; oi < anchor->depth; oi++) {
        ovg = level->brother;
        for (oj = 0; oj < level->width; oj++) {
            ovg->coordinates [0] =
               (oi + 0.5) * verspace / anchor->depth;
            ovg->coordinates [1] =
           (ovg->coordinates [1] + 0.5) *
            horspace / level->width;
            ovg = ovg->brother;
        }
        level = level->son;
    }
}     /* optgraph */
```

```
/*********************************************************/
/*                                                       */
/*     interaction                                       */
/*        Interact with the user                         */
/*        to find coordinates for new center             */
/*                                                       */
/*********************************************************/

#include <stdio.h>
#include "global"
#include "local"


interaction ()
{
    openpl(background);
    shade(foreground,0);
    zoom_factor = 1;

    if (zoom_mode) {

/*  if record has been selected then center around it */

       if (select_mode) {
          x_zoom =
          network[select_number].gp->coordinates[1];
          y_zoom =
          network[select_number].gp->coordinates[0] + 7;
       }
       else {
          plotgraph ();
          shade(2,0);
          move (10,15);
          label ("Select Center for Zooming:",2,3,9,1);
          shade(foreground,0);
          move (horspace/2, verspace/2);
          x_zoom = 3;
          y_zoom = 3;
          report (&x_zoom, &y_zoom);
       }
       zoom_factor = zoom_mode;
       zoomgraph ();
    }
    plotgraph ();
    closepl();
}   /* interaction */
```

```
/*************************************************************/
/*                                                           */
/*      plotgraph                                            */
/*         plot the graph                                    */
/*                                                           */
/*************************************************************/

#include <stdio.h>
#include "global"
#include "local"

struct element *vep;
int x, y, l;

plotgraph ()
{
    clearpl();
    luft = (float) luft * zoom_factor;
    let_h = 14.0 * zoom_factor;
    let_w =  9.0 * zoom_factor;
    sets = NULL;

    for (linepos = 0;
         linepos < a_schema.n_records;
         linepos++) {
        if (network[linepos].turn) {
           y = network[linepos].gp->coordinates[0];
           l = length(a_record[linepos].name);
           x = network[linepos].gp->coordinates[1] -
               l*let_w/2;
           box (x-luft, y-luft,
                let_w*l + luft*2, let_h + luft*2);
           move(x, y);
           a_record[linepos].name[l] = ' 00';

           label(a_record[linepos].name,
              let_w/9, let_h/7, 0, 0);

           vep = network[linepos].ep;
           while (vep != NULL) {
              if (network[vep->number].turn) {
                 drawset (linepos, vep->number,
                                   vep->set_token);
              }
              vep = vep->ep;
           }
        }
    }
} /* plotgraph */
```

```
/*********************************************************/
/*                                                       */
/*     drawset                                           */
/*        draw a line from point 1 to point 2 and        */
/*        insert the setname                             */
/*                                                       */
/*********************************************************/

#include <stdio.h>
#include "global"
#include "local"

int ox, oy, mx, my, sx, sy, nl, vs, hs;
struct set_cor *v_set;

drawset (owner, member, set)
int owner, member, set;
{
/*  find out relative position of owner and member */

    ox = network[owner].gp->coordinates[1];
    oy = network[owner].gp->coordinates[0];
    mx = network[member].gp->coordinates[1];
    my = network[member].gp->coordinates[0];
    nl = length(a_set[set].name);
    a_set[set].name[nl] = ' 00';
    vs = verspace/anchor->depth/2*zoom_factor;
    hs = nl*let_w + 5*luft;
    if ((mx+ox) < horspace) hs *= -1;
    sy = 0;

/*  include set in list of all sets   */

    v_set = (struct set_cor *) malloc (sizeof(*sets));
    v_set->n_set = sets;
    v_set->s_owner = owner;
    v_set->s_member = member;
    sets = v_set;

    v_set = v_set->n_set;

    while (v_set != NULL) {
       if ((v_set->s_owner == owner) &&
           (v_set->s_member == member))
          sy += luft;
       v_set = v_set->n_set;
    }

/*  member is in lower level as owner */

    if (my > oy) {
       sx = (ox + mx + luft) / 2;
       sy += (oy + my) / 2;
```

```
            arrow (ox, oy+let_h+luft, mx,my-luft,zoom_factor);
        }

    /*  member is on same level as owner */

        else if (my == oy) {
            line (ox, oy+let_h+luft, ox, oy+vs);
            cont (mx, my+vs);
            arrow (mx, my+vs, mx, my+let_h+luft, zoom_factor);
            sx = (ox + mx) / 2 - nl/2;
            sy += oy+vs;
        }

    /*  member is on higher level than owner */

        else {
            line (ox, oy+let_h+luft, ox, oy+vs+luft);
            cont (mx+hs, oy+vs+let_h);
            cont (mx+hs, my-vs+let_h);
            cont (mx, my-vs+let_h);
            arrow (mx, my-vs+let_h, mx, my-luft, zoom_factor);
            sx = (ox + mx + luft) / 2 + hs;
            sy += (oy + my) /2;
        }

    /*  label the arc  */

        move (sx, sy);
        label (a_set[set].name, let_w/9, let_h/14, 0, 0);

    } /* drawset */
```

```
/*************************************************************/
/*                                                           */
/*      zoomgraph                                            */
/*         procedure to zoom the graph                       */
/*                                                           */
/*************************************************************/

#include <stdio.h>
#include "global"
#include "local"

struct graphelement *ge;

zoomgraph ()
{

/*  center all coordinates */

    level = anchor;
    while (level->son != NULL) {
        ge = level->brother;
        while (ge != NULL) {
            ge->coordinates[1] -= x_zoom;
            ge->coordinates[0] -= y_zoom;
            ge = ge->brother;
        }
        level = level->son;
    }

/*  zoom all coordinates */

    level = anchor;
    while (level->son != NULL) {
        ge = level->brother;
        while (ge != NULL) {
            ge->coordinates[0] =
              (float)ge->coordinates[0] * zoom_factor;
            ge->coordinates[1] =
              (float)ge->coordinates[1] * zoom_factor;
            ge = ge->brother;
        }
        level = level->son;
    }

/*  move all coordinates to new center */

    level = anchor;
    while (level->son != NULL) {
        ge = level->brother;
        while (ge != NULL) {
            ge->coordinates[1] += horspace / 2;
            ge->coordinates[0] += verspace / 2;
            ge = ge->brother;
```

```
        }
        level = level->son;
    }
}   /* zoomgraph */
```

```
/*************************************************************/
/*                                                           */
/*      printgraph                                           */
/*         graph-picture on vt100 or for printer             */
/*                                                           */
/*************************************************************/

#include "global"
#include "local"

int i,j;

struct graphelement *vg;
char name[32];

printgraph ()
{
    linecalc();
    level = anchor;
    pagepos = 0;

    while (level->son != NULL) {

        while (pagepos <
            (level->brother->coordinates[0] - 2)) {
            drawline ('0');
        }
        linepos = 0;
        vg = level->brother;
        while (vg != NULL) {
            if (vg->number == -1)
               strncpy(name, "dummy ", 6);
            else
               strncpy(name,a_record[vg->number].name,32);

            while (linepos < (vg->coordinates[1] - 2)) {
                drawline (' ');
            }
            for (i = 0; i < length(name) + 4; i++) {
                drawline ('*');
            }
            vg = vg->brother;
        }

        drawline ('0');
        vg = level->brother;

        while (vg != NULL) {
            if (vg->number == -1)
               strncpy(name, "dummy ", 6);
            else
               strncpy(name,a_record[vg->number].name,32);
```

```
            while (linepos < (vg->coordinates[1] - 2) ) {
                drawline (' ');
            }
            drawline ('*');
            drawline (' ');
            for (i = 0; i < length(name); i++) {
                drawline (name[i]);
            }
            drawline (' ');
            drawline ('*');
            vg = vg->brother;
        }

        drawline ('0);

        vg = level->brother;

        while (vg != NULL) {
            if (vg->number == -1)
               strncpy(name, "dummy ", 6);
            else
               strncpy(name,a_record[vg->number].name,32);

            while (linepos < (vg->coordinates[1] - 2) ) {
                drawline (' ');
            }
            for (i = 0; i < (length(name) + 4); i++) {
                drawline ('*');
            }
            vg = vg->brother;
        }
        level = level->son;
    }
    drawline('0);
    drawline('0);
}                                   /*  printgraph */
```

```
/*************************************************************/
/*                                                           */
/*      linecalc                                             */
/*         prepare lines in lines-array for printing         */
/*                                                           */
/*************************************************************/

#include <stdio.h>
#include "global"
#include "local"

int i,j;
int xl,x2,yl,y2;
struct element *vep;
double ydiff,xdiff;

linecalc ()
{
    for (i = 0; i < verspace; i++) {
        for (j = 0; j < horspace; j++) {
            *(lines + (i * horspace + j)) = FALSE;
        }
    }
    for (i = 0; i < a_schema.n_records; i++) {
        if (network[i].turn) {
            xl = network[i].gp->coordinates[0] ;
            yl = network[i].gp->coordinates[1] +
                 length(a_record[i].name) / 2;

            vep = network[i].ep;
            while (vep != NULL) {
             if (network[vep->number].turn) {
              x2 =
            network[vep->number].gp->coordinates[0] - 2;
              y2 =
            network[vep->number].gp->coordinates[1] +
            length(a_record[vep->number].name) / 2;

              for (j = xl; j <= x2; j++) {
               xdiff = x2 - xl;
               ydiff = y2 - yl;
               ydiff = ydiff / xdiff;
               linepos = yl + ydiff * (j-xl);
               if (*(lines + (j*horspace + linepos)))
                  *(lines + (j*horspace + linepos)) = 4;
               else {
                if (ydiff > 0.5)
                   *(lines + (j*horspace + linepos)) = 1;
                else if (ydiff < -0.5)
                 *(lines + (j*horspace + linepos)) = 2;
                else
                 *(lines + (j*horspace + linepos)) = 3;
               }
```

```
                }
            }
                vep = vep->ep;
            }
        }
    }
} /* linecalc */
```

```
/*********************************************************/
/*                                                       */
/*       local utility-procedures for ray's thesis       */
/*                                                       */
/*********************************************************/

#include <stdio.h>
#include "global"
#include "local"

int di,dj;
struct firstelement *ifp;
struct graphelement *igp, *ligp, *cgp, *vgp;


drawline (what)
char what;
{
    if (what == '0) {
        dj = horspace - linepos;
        what = ' ';
    }
    else {
        dj = 1;
    }

    for ( di = 0; di < dj; di++) {
     if (*(lines + (pagepos * horspace + linepos)) == 1)
            buf [di] = '\';
     else if (*(lines +(pagepos*horspace+linepos)) == 2)
            buf [di] = '/';
     else if (*(lines +(pagepos*horspace+linepos)) == 3)
            buf [di] = '!';
     else if (*(lines +(pagepos*horspace+linepos)) == 4)
            buf [di] = '+';
     else
            buf [di] = what;
        linepos++;
    }

    if ( linepos >= (horspace - 1) ) {
        buf [dj] = '0;
        dj++;
        pagepos++;
        linepos = 0;
    }
    write (fd, buf, dj);
} /* drawline */


/*********************************************************/
/*                                                       */
/*     length                                            */
```

```
/*                                                              */
/****************************************************************/

int length (name)
char name [];
{
   for ( di = 0;
       ((name[di] != ' ') &&
        (name[di] != ' 00') && ( di < 32));
       di++) ;
   return (di);

}              /*    length     */


/****************************************************************/
/*                                                              */
/*     prints                                                   */
/*                                                              */
/****************************************************************/

prints (st)
char st[];
{
   strncpy (buf, st, strlen(st));
   write (2, buf, strlen(st));
}    /*   prints    */


/****************************************************************/
/*                                                              */
/*     linecross                                                */
/*                                                              */
/****************************************************************/

int linecross (c1,c2,c3,c4)
int c1[2],c2[2],c3[2],c4[2];
{
    double a1,a2,b1,b2,x;

    if ((c1 == c3) || (c2 == c4)) return (0);
    if (c2[0]==c1[0]) return (0);
    if (c4[0]==c3[0]) return (0);

    a1 = (double)(c2[1]-c1[1])/(double)(c2[0]-c1[0]);
    a2 = (double)(c4[1]-c3[1])/(double)(c4[0]-c3[0]);

    b1 = (double)c1[1] - a1*(double)c1[0];
    b2 = (double)c3[1] - a2*(double)c3[0];

    if (a1==a2) return (0);

    x  = (b2-b1)/(a1-a2);
```

```
    if (((x>cl[0]) && (x<c2[0])) ||
        ((x<cl[0]) && (x>c2[0])))
        return (x);
    else
        return (0);

}  /* linecross */


/*************************************************************/
/*                                                           */
/*     discoor                                               */
/*                                                           */
/*************************************************************/

discoor (cl, c2, c3, c4)
int cl[2], c2[2], c3[2], c4[2];
{
    printf ("%d, %d, %d, %d, %d, %d, %d, %d0,
             cl[0], cl[1], c2[0], c2[1],
             c3[0], c3[1], c4[0], c4[1]);
}   /* discoor */


/*************************************************************/
/*                                                           */
/*     insert_dummy                                          */
/*                                                           */
/*************************************************************/

insert_dummy (c, t)
int c[], t;
{
    int in;

/*  point to x-coordinate */

    ifp = anchor;
    while ((ifp->brother->coordinates[0]) < c[0])
        ifp = ifp->son;

/*  point to y-coordinate */

    igp = ifp->brother;
    for (in = 0; ((in < c[1]) && (igp != NULL)); in++) {
        ligp = igp;
         igp = igp->brother;
    }

/*  depending on t insert- or append dummmy-elements */

    if (t > 0) {
```

```
        cgp = igp;
        for (in = 1; in <= t; in++) {
            vgp = (struct graphelement *)
                  malloc(sizeof(*igp));
            vgp->coordinates[0] = igp->coordinates[0];
            vgp->coordinates[1] = igp->coordinates[1];
            igp->coordinates[1]++;
            width[igp->coordinates[0]]++;
            vgp->number = igp->number;
            network[vgp->number].gp = vgp;
            igp->number = -1;
            vgp->brother = cgp;
            cgp = vgp;
            ifp->width++;
        }
        if (cgp->coordinates[1] == 0)
            ifp->brother = cgp;
        else
            ligp->brother = cgp;
    }
    else if (igp != NULL) {
        for (in = 1; in <= (-t); in++) {
            vgp = (struct graphelement *)
                  malloc(sizeof(*igp));
            vgp->coordinates[0] = igp->coordinates[0];
            vgp->coordinates[1] = igp->coordinates[1] + 1;
            width[igp->coordinates[0]]++;
            vgp->number = -1;
            vgp->brother = igp->brother;
            igp->brother = vgp;
            igp = igp->brother;
            ifp->width++;
         }
    }
    else {
        for (in = in; in <= c[1]; in++) {
            vgp = (struct graphelement *)
                  malloc(sizeof(*ligp));
            vgp->coordinates[0] = ligp->coordinates[0];
            vgp->coordinates[1] = ligp->coordinates[1] + 1;
            width[ligp->coordinates[0]]++;
            vgp->number = -1;
            vgp->brother = NULL;
            ligp->brother = vgp;
            ifp->width++;
        }
    }
}   /* insert_dummy */
```

B. The Driver for the GIGI Graphics Terminal

```
/*********************************************************/
/*                                                       */
/*       Interphase to gigi graphics terminal            */
/*                                                       */
/*       The procedures are:                             */
/*                                                       */
/*          openpl :  enter grapic mode                  */
/*                    set background color               */
/*          clearpl:  clear screen, set reference        */
/*          move   :  move cursor without plotting       */
/*          cont   :  move cursor with plotting          */
/*          line   :  plot line                          */
/*          box    :  draw box                           */
/*          arrow  :  plot arrow                         */
/*          arc    :  plot circle                        */
/*          label  :  print text                         */
/*          shade  :  control shading                    */
/*                    set foreground color               */
/*          report :  enter graphic input mode           */
/*          closepl:  leave graphic mode                 */
/*                                                       */
/*********************************************************/

#include <stdio.h>
#include <math.h>
#include "local"

int diff, xcl, ycl;
double alpha, beta, inter;


openpl (i)
short i;
{
    printf (" 33PpS(I%d)0,i);
    xcl = 0;
    ycl = 0;
}   /* openpl */

clearpl ()
{
    printf ("S(E)");
    xcl = 0;
    ycl = 0;
    printf ("P[0,0]0);
}   /* clearpl */
```

```
move (x,y)
short x,y;
{
    if (x < 0) {
       diff = x - xcl;
       if (diff < 0 ) printf ("P[-%u,", -diff);
       else           printf ("P[+%u,", diff);
       xcl += diff;
    }
    else {
       printf ("P[%u,", x);
       xcl = x;
    }
    if (y < 0) {
       diff = y - ycl;
       if (diff < 0 ) printf ("-%u]", -diff);
       else           printf ("+%u]", diff);
       ycl += diff;
    }
    else {
       printf ("%u]", y);
       ycl = y;
    }
    printf ("0);
}   /* move */

cont (x,y)
short x,y;
{
    if (x < 0) {
       diff = x - xcl;
       if (diff < 0 ) printf ("V[-%u,", -diff);
       else           printf ("V[+%u,", diff);
       xcl += diff;
    }
    else {
       printf ("V[%u,", x);
       xcl = x;
    }
    if (y < 0) {
       diff = y - ycl;
       if (diff < 0 ) printf ("-%u]", -diff);
       else           printf ("+%u]", diff);
       ycl += diff;
    }
    else {
       printf ("%u]", y);
       ycl = y;
    }
    printf ("0);
}   /* cont */
```

```
line (x1,y1,x2,y2)
short x1,y1,x2,y2;
{
    if (x1 < 0) {
       diff = x1 - xcl;
       if (diff < 0 ) printf ("P[-%u,", -diff);
       else           printf ("P[+%u,", diff);
       xcl += diff;
    }
    else {
       printf ("P[%u,", x1);
       xcl = x1;
    }
    if (y1 < 0) {
       diff = y1 - ycl;
       if (diff < 0 ) printf ("-%u]", -diff);
       else           printf ("+%u]", diff);
       ycl += diff;
    }
    else {
       printf ("%u]", y1);
       ycl = y1;
    }
    if (x2 < 0) {
       diff = x2 - xcl;
       if (diff < 0 ) printf ("V[-%u,", -diff);
       else           printf ("V[+%u,", diff);
       xcl += diff;
    }
    else {
       printf ("V[%u,", x2);
       xcl = x2;
    }
    if (y2 < 0) {
       diff = y2 - ycl;
       if (diff < 0 ) printf ("-%u]", -diff);
       else           printf ("+%u]", diff);
       ycl += diff;
    }
    else {
       printf ("%u]", y2);
       ycl = y2;
    }
    printf ("0);
}   /* line */

box (x, y, dx, dy)
short x, y, dx, dy;
{
    move (x,y);
    cont (x+dx, y);
    cont (x+dx, y+dy);
    cont (x, y+dy);
```

```
    cont (x, y);
}   /* box */

arrow (xl, yl, x2, y2, f)
short xl, x2, yl, y2;
double f;
{
    int arx, ary;
    line (xl, yl, x2, y2);
    arx = 14.0 * f;
    ary = 14.0 / 2 * f;
    rotate (&arx, &ary, angle(xl-x2, yl-y2));
    cont (x2+ arx, y2+ ary);
    arx =  14.0 * f;
    ary = -14.0 / 2 * f;
    rotate (&arx, &ary, angle(xl-x2, yl-y2));
    cont (x2+ arx, y2+ ary);
    cont (x2, y2);
}   /* arrow */

arc (x,y,xl,yl,x2,y2)
short x,y,xl,yl,x2,y2;
{
    short all,al2;
    all = angle(xl-x,yl-y);
    al2 = angle(x2-x,y2-y);
    if (all > al2) all = all - al2;
    else           all = 360 - al2 + all;
    printf ("P[%d,%d]C(A%d)[%d,%d]0,x,y,all,xl,yl);
}   /* arc */

label (st,s,h,i,r)
char st[];
short s,h,i,r;
{
    if ((h > 0) && (s > 0)) {
       if (s > 16) s = 16;
       if (h > 16) h = 16;
       printf ("T(S%dH%dI%dW(N%d))'%s'0,s,h,i,r,st);
       xcl += s * 9 * strlen(st);
    }
}   /* label */

shade (c, s)
int c,s;
{
    printf("W(I%d,S%d,V)0,c,s);
}   /* shade */

report (x, y)
int *x, *y;
{
    char c;
```

```
    fflush (stdin);
    printf ("R(P(I[%d,%d]))0, *x, *y);
    while ((c = getchar()) != '[');
}   scanf ("%d;%d]
```

```
closepl ()
{
    printf (" 33\0);
}   /* closepl */
```

```
/*********************************************************/
/*                                                       */
/*   utility-procedures for graphics-interphase          */
/*                                                       */
/*********************************************************/

#include "math.h"

int angle (x,y)
short x,y;
{
    double dx,dy;

    if((x>=0)&&(y==0)) return(0);
    if((x==0)&&(y>0))  return(90);
    if((x<0)&&(y==0))  return(180);
    if((x==0)&&(y<0))  return(270);

    dx = x;
    dy = y;
    dx = dy / dx;
    if (dx < 0) dx = -dx;
    dx = atan(dx);
    dx = dx * 180 / 3.14159;

    if((x<0)&&(y>0))   dx = 180-dx;
    if((x<0)&&(y<0))   dx = dx+180;
    if((x>0)&&(y<0))   dx = 360-dx;

    return (dx);
}

rotate (x, y, angle)
int *x, *y, angle;
{
    double radiant;
    int inx, iny;
    inx = *x;
    iny = *y;
    radiant = (double) angle / 180.0 * 3.14159;
    (*x) =   cos(radiant) * inx - sin(radiant) * iny;
    (*y) =   sin(radiant) * inx + cos(radiant) * iny;
}   /* rotate */
```

C. The Local Data Definitions

```
/*********************************************************/
/*                                                       */
/*       local data definitions for ray's thesis         */
/*                                                       */
/*********************************************************/


struct ns {
   int indegree, outdegree, res_indegree;
   struct element *ep;
   struct graphelement *gp;
   short unused;
   short turn;
};
struct ns *network;

struct element {
   int number;
   int set_token;
   struct element *ep;
};

struct graphelement {
   int number;
   int coordinates [2];
   struct graphelement *brother;
} ;

struct firstelement {
   int depth, width;
   struct graphelement *brother;
   struct firstelement *son;
} *anchor, *level;

struct cycle {
   int set_token;
   struct cycle *next;
} *cycles;

int verspace, horspace;
int pagepos, linepos;
int crossings;

short *lines;
int luft, let_h, let_w;
short foreground, background;
```

```
int *width;
int max_width;
char select_record[32];
int select_number;
short select_mode;
int n_records, n_found;

float zoom_mode, zoom_factor;
int x_zoom, y_zoom;

struct set_cor {
   int s_owner, s_member;
   struct set_cor *n_set;
} *sets;
```