ISI Reprint Series
ISI/RS-88-210
June 1988

University of Southern California

Gabriel Robins

Applications of the ISI Grapher

Reprinted from
Proceedings of the Artificial Intelligence and Advanced Computer Technology Conference,
held May 4-6, 1988 in Long Beach, California.

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY 2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	3. DISTRIBUTION / AVAILABILITY OF REPORT This document is approved for public release; distribution is unlimited.
4. PERFORMING ORGANIZATION REPORT NUMBER(S) ISI/RS-88-210	5. MONITORING ORGANIZATION REPORT NUMBER(S) ---------------
6a. NAME OF PERFORMING ORGANIZATION USC/Information Sciences Institute 6b. OFFICE SYMBOL *(If applicable)*	7a. NAME OF MONITORING ORGANIZATION ---------------
6c. ADDRESS *(City, State, and ZIP Code)* 4676 Admiralty Way Marina del Rey, CA 90292	7b. ADDRESS *(City, State, and ZIP Code)* ---------------
8a. NAME OF FUNDING / SPONSORING ORGANIZATION DARPA 8b. OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS *(City, State, and ZIP Code)* 1400 Wilson Boulevard Arlington, VA 22210	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. --------------- PROJECT NO. --------------- TASK NO. --------------- WORK UNIT ACCESSION NO. ---------------

11. TITLE *(Include Security Classification)*
Applications of the ISI Grapher [Unclassified]

12. PERSONAL AUTHOR(S) Robins, Gabriel

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Year, Month, Day)*	15. PAGE COUNT
Research Report	FROM ______ TO ______	1988, June	32

16. SUPPLEMENTARY NOTATION
Reprinted from *Proceedings of the Artificial Intelligence and Advanced Computer Technology Conference*, held May 4-6, 1988 in Long Beach, California.

17. COSATI CODES

FIELD	GROUP	SUB-GROUP
09	02	

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
artificial intelligence tools, graph algorithms, graphs, intelligent systems, ISI Grapher, layout algorithms, user interfaces

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

This report describes various end-user applications that were built using the ISI Grapher, a portable software tool for displaying graphs pictorially. This report enumerates current research projects that already utilize the ISI Grapher, and also outlines several general domains where a grapher would be of considerable benefit.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] [✓ SAME AS RPT.] ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Sheila Coyazo Victor Brown	22b. TELEPHONE *(Include Area Code)* 213-822-1511 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR
83 APR edition may be used until exhausted.
All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

ISI Reprint Series
ISI/RS-88-210
June 1988

University of Southern California

Gabriel Robins

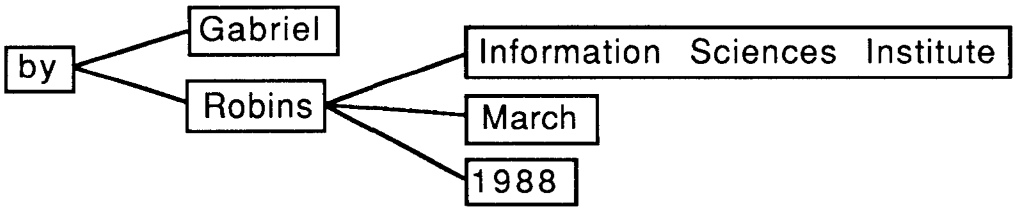
Applications of the ISI Grapher

Reprinted from
Proceedings of the Artificial Intelligence and Advanced Computer Technology Conference,
held May 4-6, 1988 in Long Beach, California.

INFORMATION SCIENCES INSTITUTE
213/822-1511
4676 Admiralty Way/Marina del Rey/California 90292-6695

This research is supported by the Defense Advanced Research Projects Agency under Contract No. MDA903 81 C 0335. Views and conclusions contained in this report are the author's and should not be interpreted as representing the official opinion or policy of DARPA, the U.S. Government, or any person or agency connected with them.

Applications of the ISI Grapher



Information Sciences Institute
4676 Admiralty Way
Marina Del Rey, Ca, 90292-6695, U.S.A.

Abstract

We describe various end-user applications that were built using the ISI Grapher, a portable software tool for displaying graphs pictorially. We enumerate numerous current research projects that already utilize the ISI Grapher, and also outline several general domains where a grapher would be of considerable benefit.

1. Introduction

[Robins, 1987] demonstrated that the ability to interactively display and manipulate arbitrary directed graphs could greatly enhance end-user productivity, both quantitatively and qualitatively, and developed a practical linear-time algorithm for laying out graphs. [Robins, 1988] described the implementation and usage of the ISI Grapher, a portable tool for displaying graphs pictorially. The salient features of the ISI Grapher are its speed, portability, extensibility, and versatility.

Over the past year we received several hundreds of requests for the ISI Grapher from companies and universities worldwide, illustrating the substantial demand for such a tool in both industry and the research community. The ISI Grapher currently runs on several different kinds of workstations (including Symbolics, TI Explorers, SUNs, HP Bobcats, Apollos, and the Apple MacIntosh II), and is also available commercially through ExperTelligence Inc.

This paper describes numerous current research projects that already utilize the ISI Grapher, and outline several additional domains where a grapher would be of considerable benefit. Throughout this paper, the term *user* denotes a person who is using the ISI Grapher (or who is using some application that is built on top of the ISI Grapher, such as the NIKL/LOOM Browser.) On the other hand, the term *application-builder* will be used to denote a person who is actually building an application using the ISI Grapher as a foundation.

Section 2 summarizes the ISI Grapher, its implementation, and its usage. Section 3 presents several existing applications already built on top of the ISI Grapher. In Section 4 we describe in further detail some methods of customizing the ISI Grapher for particular

Reprinted from the proceedings of the Artificial Intelligence and Advanced Computer Technology Conference, Long Beach, California, May, 1988.

applications. Sections 5 and 6 describe some current research efforts, both at Information Sciences Institute and elsewhere, that already use the ISI Grapher in prototype systems. Section 7 describes additional potential application areas that could greatly benefit from the usage of a grapher. Section 8 describes other existing graphers and related research. Finally, Section 9 gives instructions on how to obtain the ISI Grapher program itself.

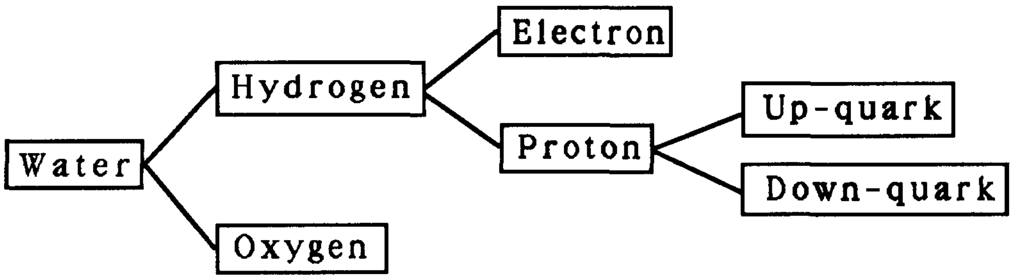
2. Overview of the ISI Grapher

The ISI Grapher is invoked at the top-level by calling the function **graph-lattice** with a list of roots/options and a "sons-function". This provides a means for the ISI Grapher to infer the complete description of the graph by recursively calling the sons-function on the roots and their descendents. Next, a reasonable graphical layout is computed for the graph by mapping its nodes onto lattice points in the plane and the resulting diagram is presented on the display. Various mouse sensitivity and functionality is automatically provided for, creating a versatile and user-friendly browsing environment.

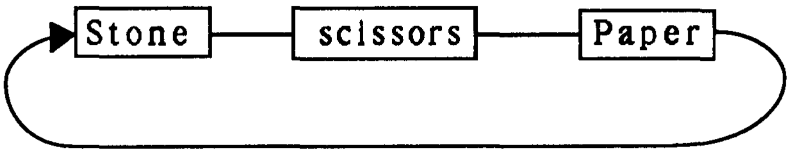
For example, suppose our graph is {(a,b),(a,c),(b,d)}, our root is {a}, and our sons-function is given by:

```
(defun components (x)
       (cond  ((eq  x  'water)  (list  'hydrogen  'oxygen))
                    ((eq  x  'hydrogen)  (list  'electron  'proton))
                    ((eq  x  'proton)  (list  'up-quark  'down-quark))
                    (t  NIL)))
```

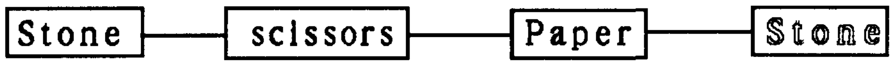
Note that the sons-function returns NIL if and only if the given node is a leaf in the graph (that is, the given node has no children). The call (**graph-lattice** 'water 'components) would produce the following:



The *function* **graph-lattice** *also accepts an optional* **layout-flag** *argument which may* be either **'tree** or **'lattice**. **'tree** means the graph will be displayed as a pure tree, regardless of its structure (in case there are cycles, they will be "broken" for displaying purposes by the introduction of "stub" nodes), while **'lattice** means that all the cross-edges in the graph will actually appear in the drawing. For example, if this flag is **'tree** the graph {(a,b),(b,c),(c,a)} which appears as follows:



will actually be displayed as follows:



where "Stone" represents the same graph node as "Stone". That is, the graph node represented by "Stone" is displayed twice (with an obvious notation that this has occurred, such as the usage of a distinctive font; this is automatically provided for by the ISI Grapher and may also be controlled by the application-builder). It is also possible to display the given graph so that the cross-edges are all displayed as they occur in the graph, with nodes properly displaced horizontally so that all edges are directed from left to right. If the graph contains any directed cycles, they are automatically broken as described above.

The first argument to **graph-lattice** may in fact be a command list that allows the user to precisely select the subset of the graph to be processed. Options include various set-theoretic operations on the nodes of the graph, and the ability to graph nodes below the given one, nodes above the given one, nodes not below the given one, and so on. The "search-depth" (e.g., a cutoff-depth) may be also specified, allowing the graphing of only the nodes that are not more than a given distance away from a specified node in the graph. Once a graph has been layed-out and is displayed in a window, various commands are available from the main command menu.

The time required by the ISI Grapher to layout a graph is linearly proportional to the size of the graph; that is, the asymptotic time (and space) complexity of the layout algorithm for a graph $G=(V,E)$ is $O(|V| + |E|)$, where $|V|$ is the size of the node set, and $|E|$ is the size of the edge set. Moreover, the constant of proportionality in this linear relation is relatively small, yielding both a theoretical optimum as well as practical efficiency. In benchmark runs, speeds of over 2,500 nodes per real-time minute have been achieved by the ISI Grapher when running on a Symbolics workstation. This efficiency is rather striking considering that it can be shown that under some simple esthetic assumptions, "optimal" layout becomes NP-hard, even to within a small bounded approximation [Supowit and Reingold, 1983].

The layout algorithm employed by the ISI Grapher exhibits an interesting symmetry: layout is performed independently in the X and Y directions. The X coordinates (of the nodes in the layout) are computed, and then the Y coordinates are computed **without** referring to the value of any of the X coordinates. This property implies a certain logical "orthogonality" in the treatment of the two planar dimensions, and is the source of the simplicity of the layout algorithm, the heart of which occupies less than two pages of code.

The Y coordinates of a node N are computed as follows: if N is a leaf node (that is, if N has no children in the graph), its Y coordinate is selected so that it is as close as possible to, but does not overlap any previously layed out node. If N has children, their Y coordinates are computed first, and then N's Y coordinate is set to be the arithmetic average of the Y coordinates of N's children. Note that the second rule implies depth-first recursion, which is indeed how the algorithm is implemented. The Y-direction layout is sensitive to the heights of the objects being displayed. On the other hand, the Y-direction layout is completely oblivious to the X-coordinate values.

Similarly, the X coordinates of a node N are computed as follows: if N is a root node (that is, if N has no parents in the graph), its X coordinate is set to zero. If N has parents, their X coordinates are computed, and then N's X coordinate is set to be some fixed amount larger than the maximum of the X coordinates of N's parents. Again, note that this implies depth-first recursion. The X-direction layout is sensitive to the lengths of the objects being displayed, and is completely oblivious to the Y-coordinate values.

For the sake of completeness, we specify the X and Y layout algorithms more formally. The layout algorithm for the Y coordinates is specified as follows:

```
For N in Nodes do Y[N] := 0;
Last-y := 0;
For N in Roots(G) do Layout-Y(N);

Procedure Layout-Y(N);
begin
if Y[N] = 0 then                          /* N was not yet layed-out */
        If N has any unlayed-out children then
                begin                     /* layout the children first. */
                for C in Children(N) do Layout-Y(C);
                Y[N] := average-Y(Children(N));
                end
        else    begin                     /* layout a leaf. */
                Y[N] := Last-y + Height(N);
                Last-Y := Y[N];
                end;
end;                                      /* of procedure Layout-Y */
```

The layout algorithm for the X coordinates is specified as follows:

```
For N in Nodes do X[N] := 0;
For N in Leaves(G) do Layout-X(N);

Procedure Layout-X(N);
begin
if X[N] = 0 then                          /* N was not yet layed-out. */
        If N has parents then
                begin                     /* layout the parents first. */
                for C in Parents(N) do Layout-X(C);
                X[N] := Max{X[i] + Width(i) | i in Parents(N)} + constant;
                end
end;                                      /* of procedure Layout-X */
```

The ISI Grapher maintains various data structures for each graph that it processed. In particular, each node and edge of the graph is represented as an instance of a LISP record structure. Nodes structures point to both their parents and children. The Grapher maintains several hash tables that serve to map between node names and the corresponding data structures. Application builders should remember that their code should leave these various data structures in a consistent state, and are advised to use whenever possible the provided built-in functions for such manipulations.

Once a graph has been layed-out and is displayed in a window, various commands are available from the main command menu. Many other functions are also available for the application-builder's use. When the mouse points to a node in an active Grapher window, that node becomes highlighted and various additional commands from the main command menu become available and operate with respect to that node. For example, if a node is selected (highlighted) and the command "delete-node" is issued by selecting the corresponding menu item, that node will be removed from the graph and the window will be redrawn. Figure 1 is an example of an ISI Grapher display, depicting the ExperTelligence class system. The entire graph is visible on the right, while the highlighted section is magnified and displayed on the left.

To provide for its portability, the ISI Grapher code is divided into two main modules. The first and largest module consists of pure Common LISP code; this code is responsible for all the layout, control, and data-structure manipulation algorithms. The second module is substantially smaller, and consists of numerous low-level primitive calls that are quite likely to be implementation-dependent. The intent here is that when the ISI Grapher is to be ported to another (Common LISP) environment, **only** the second module should require modification. In order to further minimize porting efforts, the calls from code in the first module to functions in the second module were designed to be as generic as possible.

In summary, if a new environment has a window-system that supports a reasonable set of window and graphics primitives (such as open-window, draw-line, print-string, etc.), then porting the ISI Grapher to this new environment or machine should require a minimal coding effort, probably all of which would be confined to the second section of the ISI Grapher code. The ISI Grapher has also been ported to X, a standard portable window system environment implemented by a group at MIT; this means that porting the ISI Grapher to a system that supports **both** Common LISP and X Windows should indeed be trivial.

3. Simple Applications of the ISI Grapher

We now describe the various applications that are built on top of the ISI Grapher:

<u>The List Grapher</u> - This application displays the natural correspondence between lists and trees, by taking the CAR of the list to be the root and the CDR to be the list of children, recursively. This provides an easy means of quickly obtaining large or complex graphs. For example, the following call would produce the graph in Figure 2.

```
(graph-list
  '(device
       (computing-device
          (analog-computer slide-rule wind-tunnel)
          (digital-computer super-computer micro-computer))
       (electrical-device super-computer micro-computer radio)
       mouse-trap)))
```

<u>The Flavor Grapher</u> - This application displays the interdependencies between flavors in flavor-based LISP environments (such as on Symbolics or TI workstations). Nodes represent flavor names, and edges mean "depends on." This type of a diagram could be quite useful in LISP software development. For example, the call:

```
(graph-flavor 'tv:window 'lattice)
```

would graph (as a lattice) all the flavors that depend on the tv:window flavor.

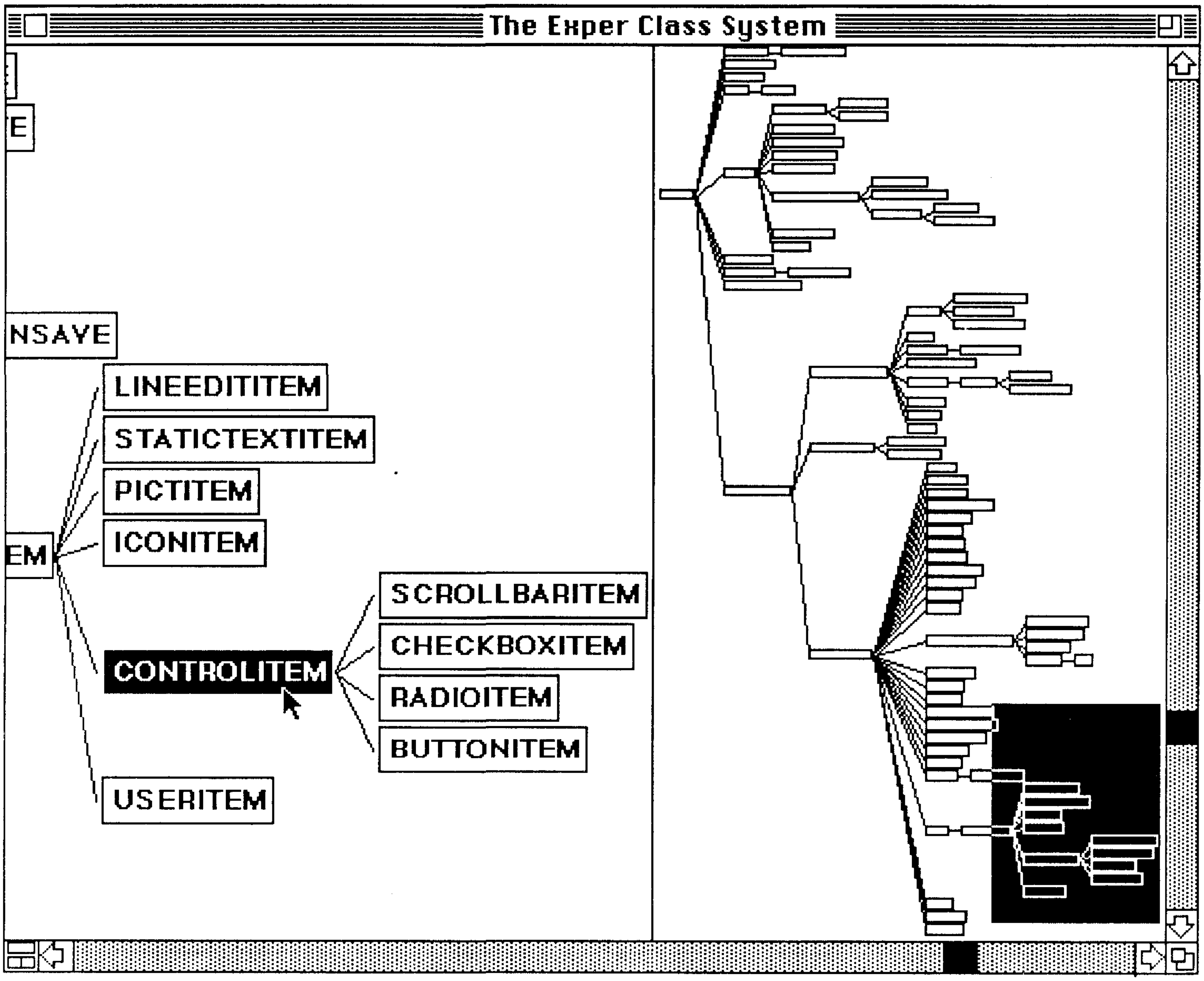


Figure 1: The ExperTelligence class hierarchy.

<u>The Package Grapher</u> - This application produces a graph of the package interdependencies between a package and all packages that use it, where nodes represent packages and edges represent package inheritance. An example of a call is:

```
    (graph-package "global")
```

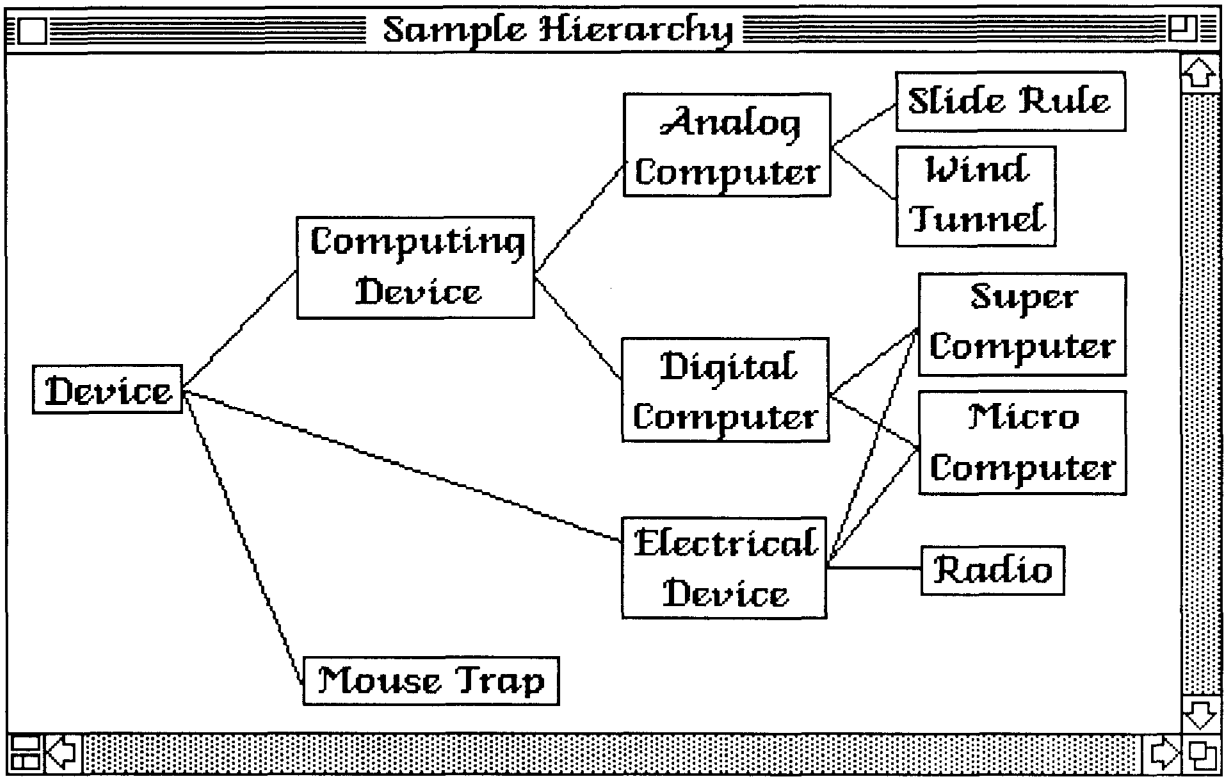
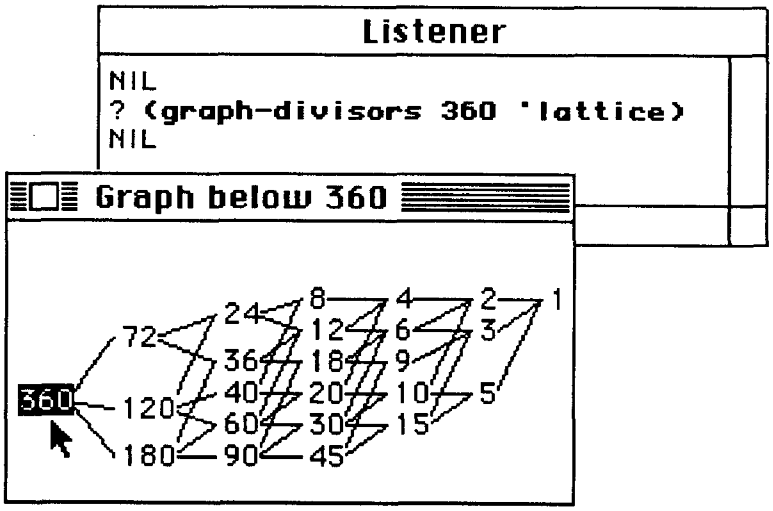


Figure 2: An example of the List Grapher.

<u>The Divisors Grapher</u> - This application displays the divisibility graph of a given integer; that is, all the divisors of an integer are represented by nodes, where an edge between two nodes means "is divisible by." This is also a quick method to produce large graphs. For example, the following call would produce the graph in Figure 3:

```
(graph-divisors 360 'lattice)
```



```
NIL
? (graph-divisors 360 'lattice)
NIL
```

Figure 3: The integer divisors of 360.

The NIKL Browser - This application is a browsing tool for NIKL networks. NIKL is a state-of-the-art, classification-based knowledge representation language, developed jointly by ISI and BBN. In NIKL, concepts are ordered by logical subsumption and it is often desirable to see a picture of a NIKL taxonomic network. For example, Figure 4 depicts a typical medium-sized NIKL taxonomy.

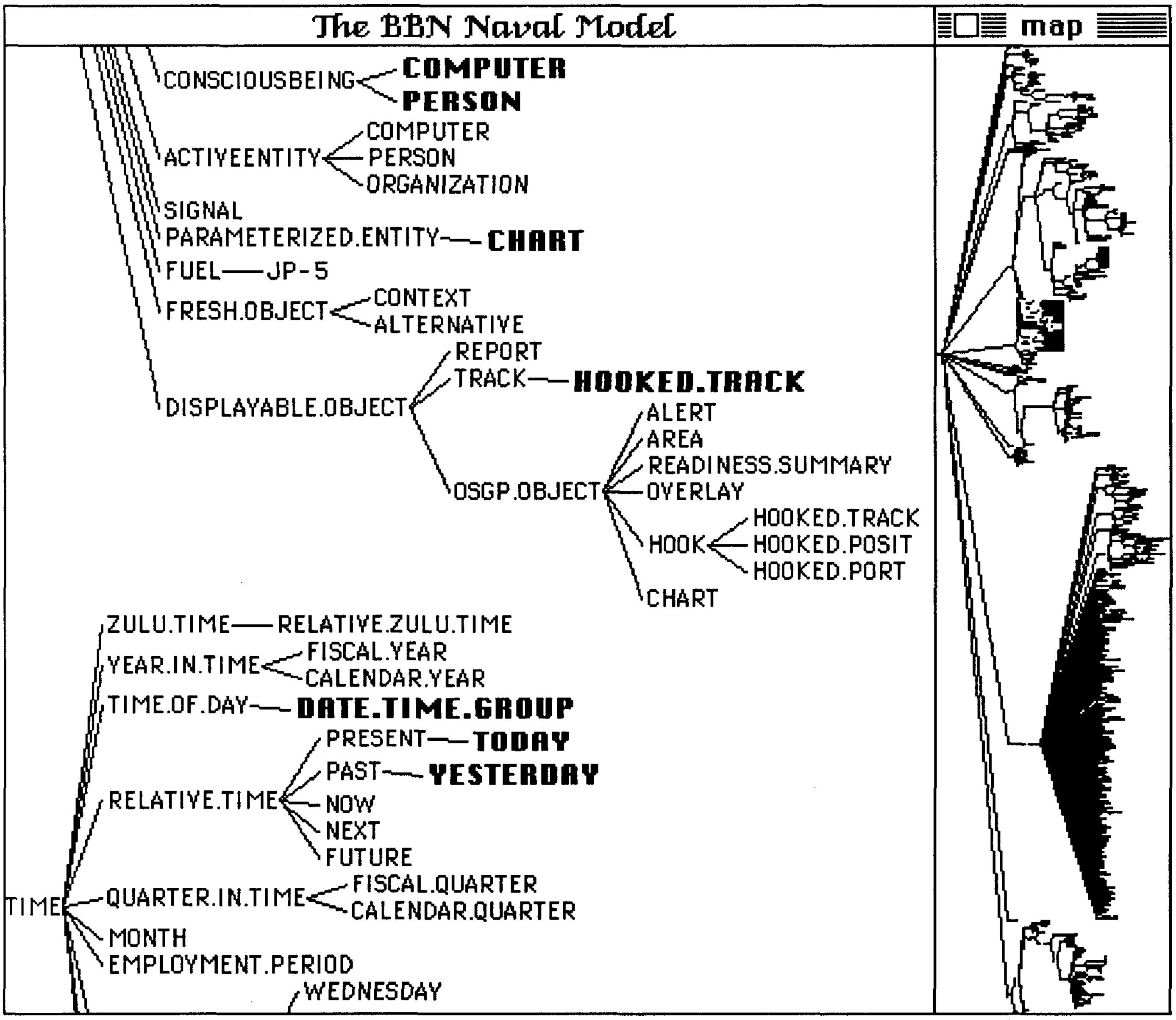


Figure 4: A typical medium-sized NIKL taxonomy.

3.1. The List Grapher Revisited

To illustrate how an application may be concisely and easily built on top of the ISI Grapher, we provide the complete code for the List Grapher application:

```
(defvar sons-table)

(defun sons (nodes)
   (if (atom nodes) (setq nodes (list nodes)))
   (mapcan #'(lambda (i) (gethash i sons-table)) nodes))

(defun search-sons (lattice)
   (if (or (null lattice) (not (listp lattice)))
      (return-from search-sons nil))
   (puthash (car lattice)
        (mapcar #'(lambda (x) (if (listp x) (car x) x))
           (cdr lattice))
        sons-table)
   (mapcar #'search-sons lattice))

(defun graph-list (the-list (layout-style 'tree))
   (setq sons-table (make-hash-table)
   (search-sons the-list)
   (graph-lattice (car the-list) 'sons layout-style))
```

This code consists of some simple preprocessing during which the nodes and their children are stored in a hash table for fast future reference; this relationship is computed according to the recursive rule that the CAR of a list is the root while the CDR is the list of children. Once this relation has been computed, all that needs to be done is to call the ISI Grapher function **graph-lattice**, and the corresponding graph will be computed, layed-out, and displayed. Other ISI Grapher applications are also as concise and easily specifiable as this application.

4. Application-building

It is possible for the application-builder to define new functionality and customize the menus of the ISI Grapher. In addition, several basic Grapher operations may be controlled via the specification of alternate functions for performing these tasks. These operations include the drawing of nodes and edges, the selections of fonts, the determination of print-names, pretty-printing, and highlighting operations. Standard definitions are provided for these operations if the application-builder chooses not to override them.

For example, the default method of highlighting a graph node when the cursor points to it on the screen is to invert a solid rectangle of bits over the node. Suppose that the user is not satisfied with this mode of highlighting and would like to have thin boxes drawn around highlighted nodes instead. He may write a highlighting function that does exactly that, and tell the Grapher to use that function whenever a node needs to be highlighted.

As another example, suppose the user is not satisfied with the way nodes are displayed on the screen; ordinarily nodes are displayed on the screen by printing their ASCII print-names at their corresponding screen location. If the user would prefer that some specialized icon be displayed instead, he may then specify his icon-displaying function as the normal node-painting function; from then on, whenever a node needs to be displayed on the screen, that function will be called upon, thus achieving the desired effect.

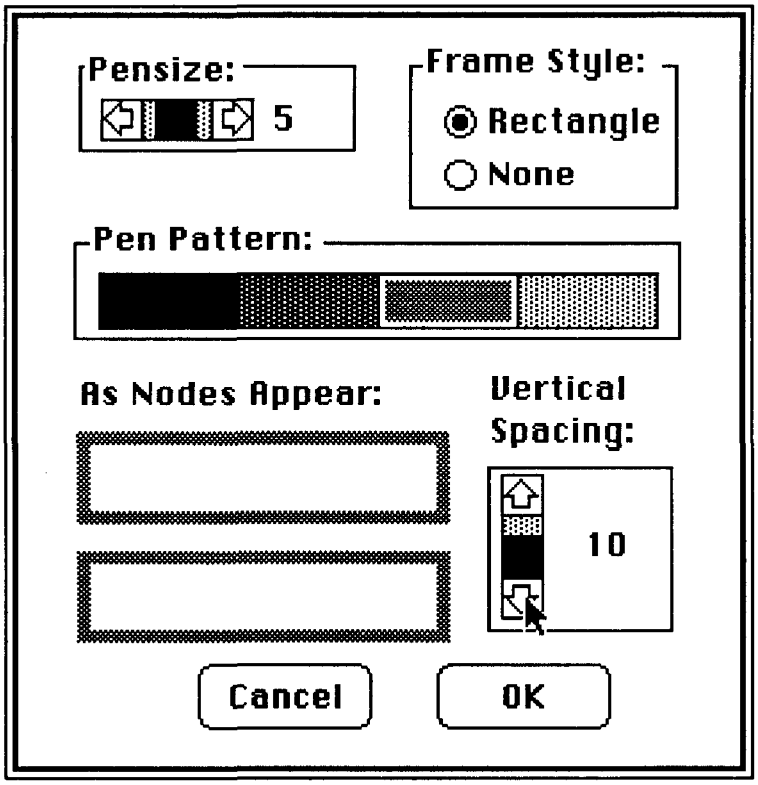
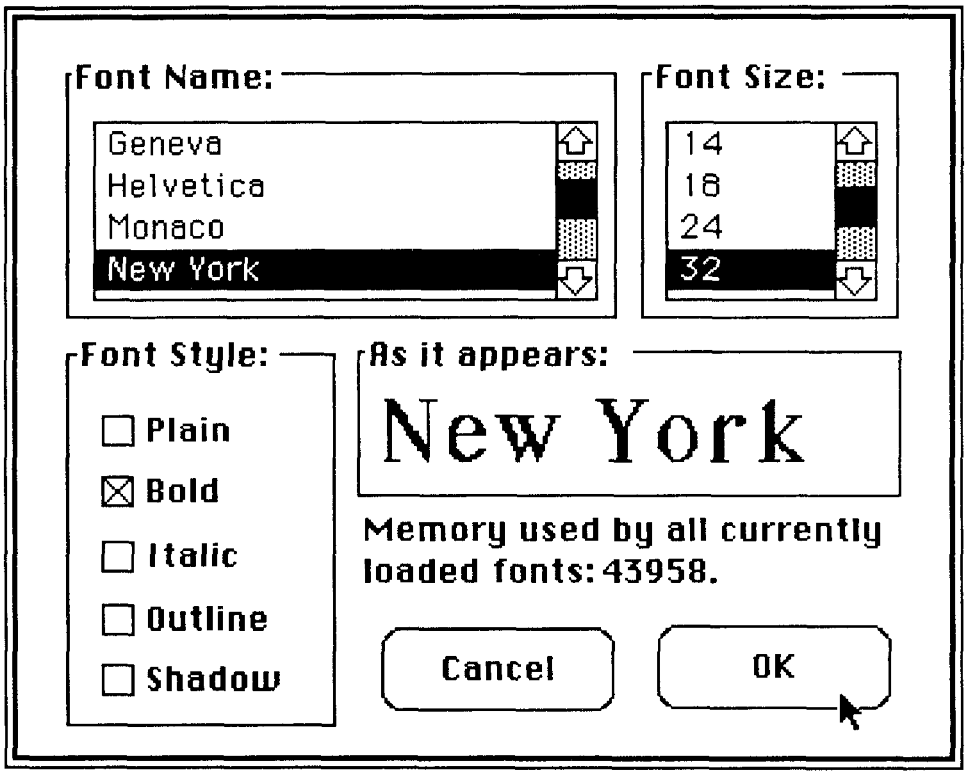

Pensize: 5
Frame Style: [✓ Rectangle] None
Pen Pattern:
As Nodes Appear:
Vertical Spacing: 10

Figure 5: Changing node display characteristics.

When the application-builder supplies his own function for performing a particular task category, that function must externally mimic the semantics for that task. In particular, the application-builder's function must have the same number (and type) of arguments (and returned value) as the default function for that category. For example, if the application-builder defines a new font-function, it must accept two arguments, a node object and a window, and return a font. How the returned font is selected (or whether it indeed depends on the input at all), is a decision left entirely to the application-builder. The application-builder must exercise some care therefore, in designing his replacement functions; for example, in most applications, the unhighlight-function should "undo" what the highlighting-function does, etc. The Grapher cannot determine whether the application-builder has provided a consistent (or even a useful) set of functions.

Much of this functionality can be invoked from the command menus; for example, Figure 5 shows how the manner in which nodes are displayed may be controlled via menus/dialogues, and similarly for fonts, as shown in Figure 6. After both the display and font styles have been changed by the user, the display may acquire the style of Figure 7. Figure 8 depicts a NIKL taxonomy drawn as a lattice, where numerous cross-edges are visible; when graphs become dense (in the sense that the ratio of edges to vertices is high) experience has shown that it is preferable to display them as trees.


Font Name:
Geneva
Helvetica
Monaco
New York

Font Size:
14
18
24
32

Font Style:
☐ Plain
☒ Bold
☐ Italic
☐ Outline
☐ Shadow

As it appears:
New York

Memory used by all currently loaded fonts: 43958.

Figure 6: Changing display fonts and styles.

5. Specific Projects Which Use the ISI Grapher

This section describes various current research projects at Information Sciences Institute that have used the ISI Grapher in the implementation of prototype systems.

5.1. Integrated Interfaces

A wide variety of user interface modes and media are available for modern computer systems. The Integrated Interfaces project, headed by Dr. Norman Sondheimer, is engaged in research that allows users to choose among functionally equivalent methods on input while the system distributes display responsibilities to a variety of different output methods. This project uses natural language understanding and generation, graphic input and output, command language interaction, menus, icons, forms, multiple windows, keyboard, mouse, and eventually speech. They are developing a generic user interface technology with clear separation between the interface system, device support software, and application software systems.

This approach combines artificial intelligence technology and human factors, and employs a knowledge-based software architecture for the system. The knowledge base describes the capabilities of the input and output devices, and the input and output requirements of the application software systems. AI reasoning is used to analyze input in the context of a multi-media and multi-mode dialog. AI planning is used to assign output demands to the most appropriate mix of different media and modes. The knowledge about the structure of the rules used in planning and hence the form and behavior of the interface is based on a series of human factors experiments.

The ISI Grapher is used in this project as a fundamental display mode. Various relations among the components of the domain may be displayed pictorially and manipulated using the !SI Grapher.

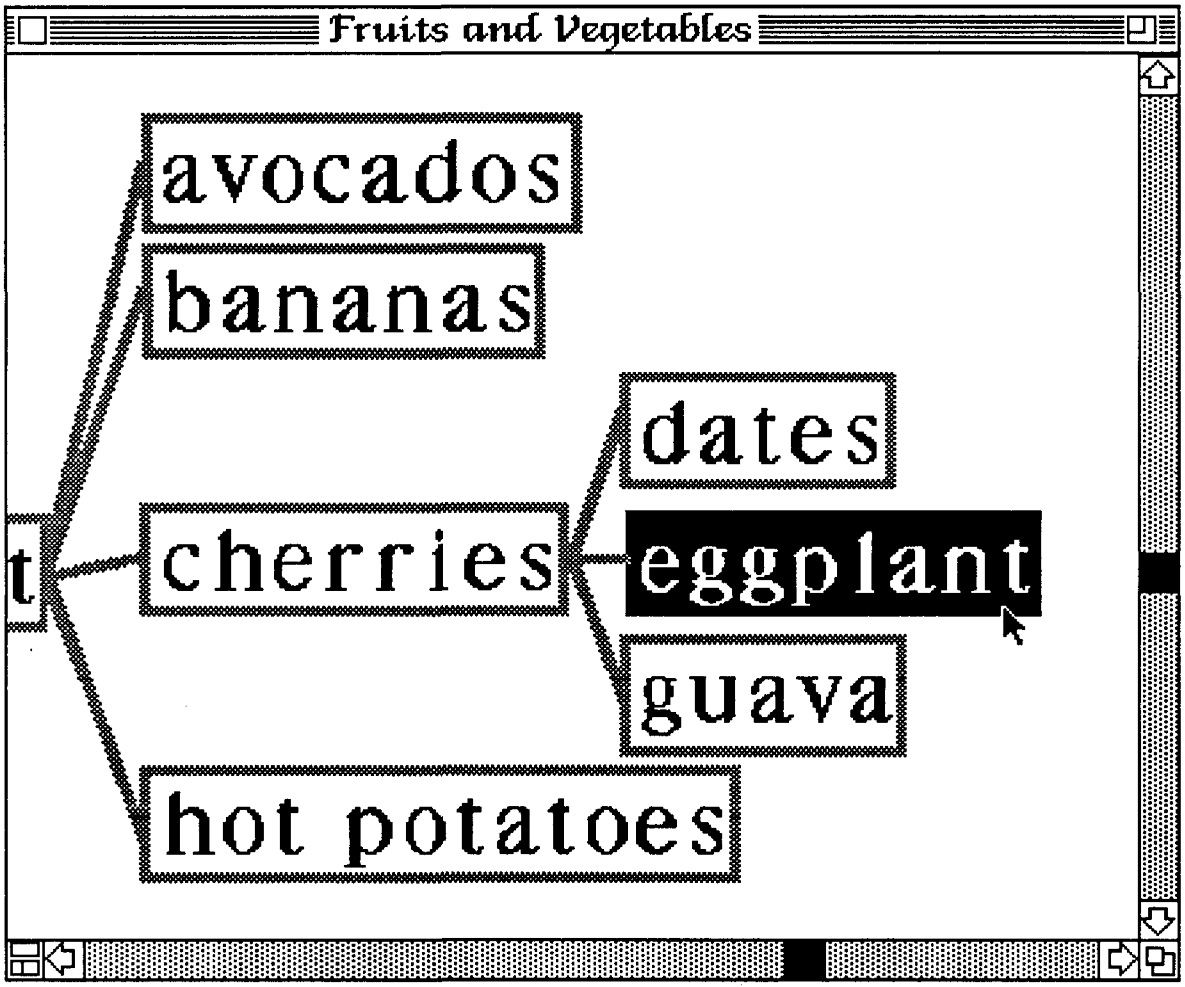


Figure 7: An example of using a bigger font and thicker boxes/lines.

5.2. Natural Language and Text Generation

This project, headed by Dr. Bill Mann, explores new technology for expressing computer-internal information in sentences and paragraphs of English text. Many systems require information output that is flexible, understandable, and precise; canned text is inflexible and often misleading. The objective of this research is to develop autonomous English generation technology with a focus on in-context multi-sentence generation. This technology has fundamental applications in building English-out, English-in human-computer interfaces to computer software systems such as database and expert systems.

The text generation system is called PENMAN, and is based on the design described in [Mann]. The major components of PENMAN include a sentence generator and several text planning modules. The sentence generator uses a large systemic-functional grammar of English (the grammar is called NIGEL [Mann, and Matthiessen]). The text planning modules are based on Rhetorical Structure Theory (RST) [Mann, and Thompson]. The inputs to PENMAN are expressed in a meaning representation language (Penman-MRL) which is a special variety of the first order predicate calculus. The terms of Penman-MRL are interpreted in the context of a taxonomic representation of knowledge about the subject matter [Sondheimer, and Nebel]. This research in text generation involves active interaction between AI specialists and linguists. In addition to the text generation system itself, the project has produced significant results in the study of text structure and an experimental parsing program based on the same systemic-functional grammar [Kasper].

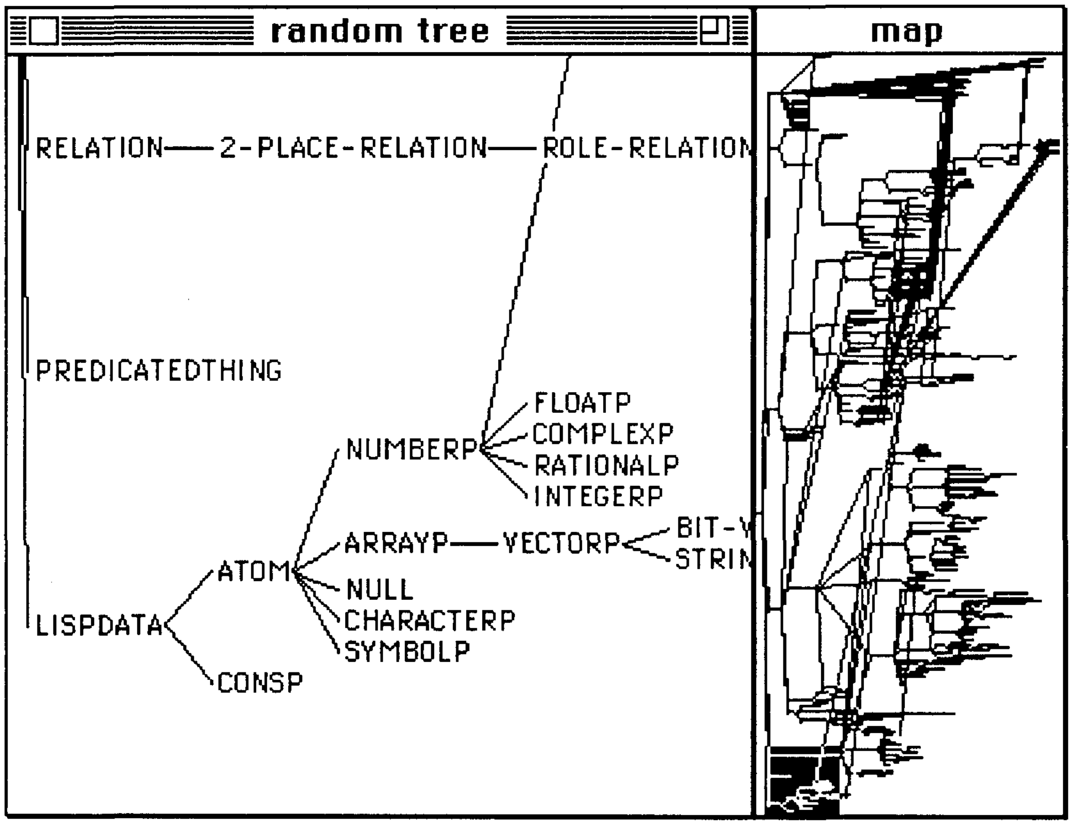


Figure 8: An example of a NIKL taxonomy drawn as a lattice.

The ISI grapher has proved to be a very useful tool for displaying several types of structures that are produced in the PENMAN system. It is used to browse through large data structures that define grammars and lexicons, and to display several kinds of structures that are produced as results of generating or parsing text. Four specific applications of the grapher will be briefly described in this section: the grammatical system network browser, a word class hierarchy editor, the display of sentence structures, and the display of text structures.

Grammatical System Network Browser: A systemic-functional grammar is organized into a network of interdependent choices of grammatical features. It is often helpful for a maintainer or user of the grammar to browse through this network, first looking at its overall organization, and then displaying information about a particular portion of the network. By providing an interface to application-defined functions on nodes of the graph, the ISI grapher is well suited to this application. This browsing facility is quite similar to the NIKL/LOOM browser, another application built on top of the ISI grapher, described elsewhere in this paper.

Word Class Hierarchy Editor: The ISI grapher is also used as a browser of a hierarchy of word classes that are used in defining lexical items to be used by the PENMAN system [Cumming & Albano 86]. In addition to browsing through the existing word classes, the word class hierarchy editor also provides functions for adding and modifying word classes.

Display of Sentence Structures: Systemic grammars describe the functional roles of sentence constituents in addition to their structure. It is common for a single constituent to fill several functional roles simultaneously. Thus, it is very helpful to have a tool that displays these functional structures explicitly as a graph, so that shared structures can be observed. This type of graph is displayed as the result of parsing a sentence, and may also be displayed after generating a sentence to observe how the sentence was constructed. A sample graph of the functional structure of a sentence is shown in Figure 9.

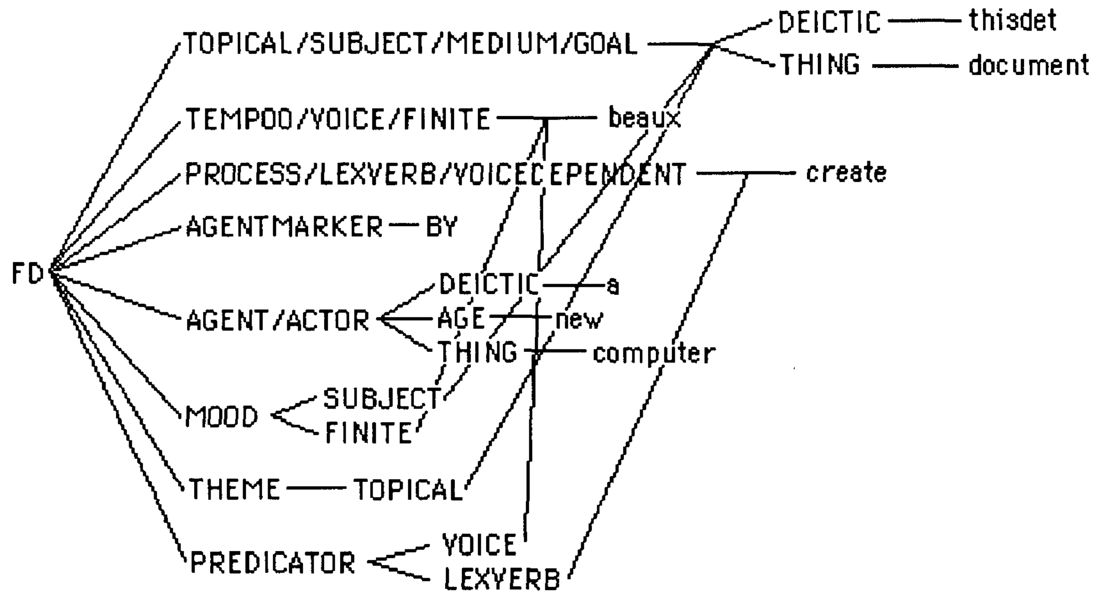


Figure 9: Sentence structure of "This document was created by a new computer."

Display of Text Structures: The overall organization of a text is planned by a module of PENMAN that builds structures spanning many sentences [Hovy]. The ISI grapher is used here to display these text structures and show the relations that exist between different sentences of a text.

In all of these applications the automatic layout and ease of scrolling over large graphs saves considerable time to developers and users of our natural language processing programs.

5.3. A Single Interface to Multiple-Systems (SIMS)

This project, headed by Dr. Norman Sondheimer, is pursuing a line of research that allows independently developed computing systems to be seen as a single, consistent whole. Just as distributed databases allow both the end-user and the application developer to see a set of independent data bases as a single system, the goal here is to produce a component that allows end-users and programmers to see a set of independent systems as a single system.

SIMS supports both a uniform view of data and a uniform view of computer services. This enables a system to satisfy a request, divide up requests that require more than one system to perform them, and consider alternative services if a system fails to satisfy a request. The key is Model-Mediated Interaction, an approach utilizing a knowledge-base which models the data structures and capabilities of the component systems. When application programmers wish to add functionality to the underlying system, they can see what functionality already exists and where their new system fits. A planner is being built to use these rules.

When an end-user presents a request, our system puts it into a form that the planner can evaluate to produce a series of commands to the component systems that satisfies the request. In addition, since several translations may exist, the system monitors the execution of the requests and may attempt recovery from failure. The same mechanism can be used to support program invocation of services.

The approach described here is dynamic in that SIMS determines which server will satisfy the request when the task is defined. SIMS performs logical integration, where the servers remain separate, but a common model makes them appear as if they were providing a single virtual service. Logical and dynamic integration allow the system of which SIMS is a part to be more easily designed and maintained. The ISI Grapher is being used in SIMS as an alternate method of display and interaction.

5.4. Classification-based Knowledge Representation

This project, headed by Dr. Robert MacGregor, is developing a state-of-the-art knowledge representation system, as well as a set of tools that would aid the process of constructing large knowledge bases. In the past, nearly every project with a need for a knowledge base has generated its own knowledge representation system. This has resulted in duplicate efforts, diversions caused by the construction and debugging of the knowledge representation system, and idiosyncratic knowledge bases that cannot be shared or reused. A major goal of this project is to provide a widely available and reusable knowledge representation system, and to validate it through application in a wide range of projects, including intelligent interfaces, expert systems, and natural language.

A usable knowledge representation system can have as dramatic an impact on the knowledge processing community as data bases have had on the information processing community. This effort is based on our previous work in defining and implementing the KL-ONE and NIKL knowledge representation systems [Robins, 1986; Kaczmarek, Bates, and Robins, 1986]. Our experience with building knowledge-based systems has led us to concentrate on the following features: expressiveness, well-defined semantics, and domain-independent reasoning and acquisition facilities.

Our new knowledge representation system is called Loom, and it features richer semantics and more powerful forms of inference than its predecessors; it is especially designed to work in tandem with other systems [Mac Gregor and Bates, 1987]. Loom provides the foundation for further research in knowledge representation; we will be developing tools and methodologies for constructing programs and knowledge bases which can be easily modified and extended, and which can be shared by multiple applications.

One research goal is to develop a language that will allow users to do most of their programming entirely within the environment of a knowledge representation system. A second research direction takes a knowledge-based approach to knowledge base construction wherein the acquisition process itself is modelled. This work will lead to the formulation of expert systems which can assist users in the model-building process.

In the NIKL/LOOM knowledge representation system, concepts are ordered by logical subsumption. NIKL/LOOM taxonomies often become quite large, containing thousands of nodes and edges. It is therefore very desirable to see a picture of a NIKL taxonomic network, rather than to inspect the equivalent formal syntactical specification, and it is indeed in the context of the NIKL project that the ISI Grapher was initially born. Users have reported enormous time savings in using the ISI Grapher to construct and debug NIKL taxonomies. The application that graphs NIKL taxonomies is called the NIKL Browser and is described in more detail earlier in this document.

5.5. FAST Workstation Project

The FAST Project, headed by Dr. Robert Gurfield and Dr. Robert Neches, seeks to demonstrate a model of electronic commerce [Neches, 1987]. It is divided into two sub-projects: the FAST Broker and the FAST Workstation. The FAST Broker project focuses on utilization of rapid electronic networks to speed communications between DoD and DARPA buyers of electronic parts and the vendors of such parts. The FAST Workstation project focuses on the development of user and software interfaces to enable human participants in the process to easily integrate and engage in transactions with the system.

One of the immediate goals of the FAST Workstation effort is to provide a package of flexible, generic software tools to aid in the procurement process. BACKBORD (Browsing Aid Complementing Knowledge Bases Or Data Bases) [Neches, DeBellis, and Yen, 1988] is an instance of such a tool. It is an intelligent interface for databases and knowledge bases, modeled after a psychological theory of human information retrieval called *retrieval by reformulation.* On top of this shell are built specific applications such as database and knowledge base browsers, an interface for the creation and attachment of notes to objects in a knowledge base, and an interface for the creation of mail messages from a parts buyer to the FAST Broker.

The ISI Grapher is used in BACKBORD to graphically display sections of the knowledge base hierarchy, and to choose values from the graphical display. For example, when creating a note, the user can view the hierarchy of note types and choose a value from the hierarchy as the type of note to create.

5.6. Diagnostic Expert Systems

This project, headed by Dr. Len Friedman, is developing a domain-independent expert system for diagnosis. It already performs state-of-the-art trouble-shooting of faults in domains as diverse as spacecraft science instruments, avionic equipment, and aircraft turbine engines. Research is being undertaken to increase its power and adaptiveness in several areas: learning, assumptions, multiple model consistency, and explanation.

In particular, research is being undertaken to emulate human performance in learning of diagnostic groups, learning when and what to measure, learning significance of cues and "danger signals", and learning to specify diagnostic mode calls. A major goal of this research is to improve the efficiency with which searches are made to reach diagnoses. The plan is to modify the knowledge bases between diagnostic sessions, using the NIKL/LOOM classifier as a pattern matcher and decision maker.

Deep diagnosis depends on the use of functional models at increasing levels of detail. This project has undertaken to specify automatic and general ways for the system to verify whether measurements made at a lower level are consistent with all hypotheses made at higher levels within a specific model. If there is inconsistency, the system searches for ways to select the set of models that are maximally consistent with the available evidence.

Maintaining such model consistency implies that we can move up and down the hierarchy of models by some means such as a web of concepts and relations linking the models. We can do this with the NIKL/LOOM knowledge representation scheme. The ability to find higher-level models from which the current model derives means that we can find justifications or explanations at a higher semantic level. Similarly, we can descend to lower levels for a more detailed explanation when needed. This enables the system to provide explanations that are more than simple execution traces.

The ISI Grapher is used in this project as a browser for NIKL/LOOM, the underlying knowledge representation language used by this expert system. Moreover, the ISI Grapher can also be used here to dynamically display search trees and decision hierarchies, perhaps interacting with the user to aid the search. Finally, various relationships between the components (e.g., transistors, chips, wires, turbine blades, fuel lines, etc.) of the domain being analyzed may be visually depicted.

5.7. The Soar Project

The Soar project attempts to build a system capable of general intelligent behavior [Laird, 1987]. It will be capable of working on a broad range of tasks, from highly routine to extremely difficult open-ended problems. Currently Soar is capable of employing the full range of problem-solving methods and representations required for these tasks, and is capable of learning about various aspects of the tasks and its performance on them. The research approach is to focus on understanding what mechanisms are necessary for intelligent behavior and how they work together to form a general cognitive architecture. Soar [Laird, 1986] consists of five components: (1) a long-term recognition memory (productions), (2) a short-term working memory, (3) a decision procedure, (4) a subgoal generator, and (5) an experience-based learning mechanism (chunking).

Since a search (for a solution) through an abstract space (of possibilities) is logically equivalent to traversing a directed graph, the progress of the search at any intermediate point in the problem-solving process may be depicted by a suitably-trimmed graph. In situations when the search is to be user-directed, the possible choices for the next-alternative-to-try may be pictorially presented to the user by means of a graph. Both of these are natural cases where a grapher may be useful.

Soar has been applied to a wide range of tasks (from simple puzzles to complex knowledge-intensive tasks), problem solving methods, and learning capabilities [Steier, et al, 1987]. Current work is focused on the development of an I/O capability, a range of knowledge-intensive systems (medical diagnosis, algorithm design, etc.), various aspects of learning from the outside (advice taking, task acquisition, and the acquisition of declarative knowledge), recovery from incorrect knowledge, problem solving methods (abstraction, progressive deepening, etc.), production-system implementation techniques (both software algorithms and parallel hardware), and cognitive modeling. The Soar project is being conducted in collaboration with researchers at a number of other sites, particularly Carnegie-Mellon University, Stanford University, and the University of Michigan; the principal SOAR researcher at ISI is Paul Rosenbloom.

5.8. Knowledge-Based Specification Assistant

Another research effort, performed in the context of the KBSA project (Knowledge Based Specification Assistant), focuses on converting declaratively stated system behaviors (e.g., system specifications written in GIST,) into a procedural form. This makes the specifications more understandable because the global constraints become localized and a cognitively different yet behaviorally equivalent view of the constraints becomes available. Furthermore, this conversion renders the specifications amenable to visual presentation and manipulation. The resulting form is presented as flow graphs where nodes represent actions (events), pre- and post conditions, and case branches, while edges represent enablements and disablements, a natural application for the ISI Grapher.

6. Outside Projects Using the ISI Grapher

In this section we describe various current research projects outside ISI that have used the ISI Grapher in their implementation of various prototype systems.

6.1. CMU's Theo Project

Theo is a software framework to support development of self-modifying problem solving systems. The Theo project, headed by Dr. Tom Mitchell of CMU, provides a frame-based representation language, representation of slots in terms of frames, automatic inference of slot values upon demand, maintenance of explanations for all inferred slot values, a learning mechanism that forms efficient inference methods from these explanations, and automatic learning of control information to infer slot values.

The ISI Grapher has been modified at CMU to run under the X Window System. This modified form of the ISI Grapher will be used as a user interface into the Theo system. The ISI Grapher would graphically present the various hierarchies that Theo maintains. This would be particularly useful in tracing the inference of a particular slot value, or in pointing out errors in knowledge base development. Both cases are currently tracked down by searching through file references. Access to editors that allow modification to any particular frame currently in the system, as well as changing hierarchy links and creation of new frames are also being considered. The ISI Grapher helps to streamline the user interface in performing all of these tasks.

6.2. Legal Precedent Browser

A researcher working on a dissertation within the legal profession used the ISI Grapher to display graphs depicting precedence among court cases. In this application, nodes represented trials and judicial decisions, while edges represented instances where the judge based the decision on a legal precedent established in a past court trial. This scheme proved quite useful in analyzing in an interactive fashion the interdependence among a large set of legal decisions.

6.3. The Wisdom Project

TA Triumph Adler AG, a major German manufacturer of business machines and computers, is currently undertaking Germany's largest joint research project in the area of office automation, the WISDOM project (Knowledge Based System for Office Communication: Document Processing, Organization, Man-Computer Communication). This research effort, carried out jointly with German universities (Universitaet Stuttgart, Technische Universitaet Muenchen) and research laboratories (Fraunhofer Gesellschaft IAO, Gesellschaft fuer Mathematik und Datenverarbeitung) is developing and applying knowledge-based techniques in filing and retrieval of documents, cooperative office procedures, and user interfaces.

A major objective of this research effort is to develop a knowledge representation paradigm, especially for certain defined sub-domains in the context of an office (for example, modeling organizations, semantic structure of combined text, graphics and natural language, documents, etc.) The theoretical aspects of the work entail issues of knowledge representation, knowledge acquisition, and the architecture of expert systems.

Implementation activities concentrate on the LUIGI knowledge representation system and various related tools. These tools include the LUIGI Knowledge Editor (LUKE) for the acquisition and maintenance of knowledge bases. LUKE consists of various instantiable sub-editors for each knowledge kind (e.g., instances, affairs, relations between objects). These sub-editors coexist in a multiple-process and multiple-window environment and communicate between each other over a blackboard. This architecture allows structured design of sub-editors and provides incremental extensibility in LUKE. The ISI Grapher is used within this framework for displaying the contents of the knowledge base graphically, and also allowing their manipulation via mouse and menu interaction.

6.4. JPL's Telerobot Flight Servicer

The Telerobot system at the Jet Propulsion Laboratory (JPL) incorporates both teleoperated and autonomous aspects of robotic control in a satellite servicing domain and will facilitate, through basic research, the design of a Telerobot Flight Servicer (TFS) for the 1990's. The current autonomous design strategy for the Telerobot includes a high-level planner that develops a task sequence for the repair or replacement of parts based on the structural dependencies between satellite components. In order to create a potentially successful plan, the task planner must be able to obtain information on the availability of manipulator trajectories which satisfy the requirement for collision-free movement in a cluttered workspace. Automated servicing plans may fail due to collisions with workspace objects or the violation of kinematic or dynamic constraints on manipulator motion.

Audrey is an interactive simulation and spatial planning environment currently under development at the JPL Sequence Automation Research Group. It will allow the Task Planner to verify the integrity of a plan by querying a software system, which contains information on object spatial locations and models of the PUMA 560 manipulator and kinematics. Audrey allows a user to directly manipulate graphics representations of objects in the satellite world and investigate potential problem areas in an automated servicing plan. Audrey also simulates motions and reports on spatial information at the command of the Task Planner. At the present stage of development, Audrey includes complete forward and inverse kinematic solutions for the manipulators, near real-time collision detection, and accurate geometric models for objects in the workspace.

The Telerobot Project requires methods that allow a user to manipulate complex nets and indicate choices by clicking on net nodes and edges (links between nodes). Such a facility would serve as an user interface to large graphs representing spatial information.

6.5. Nokia's DMG System

Nokia Inc. is developing a system for modelling and generating diagnostic networks, which is named DMG [Lounamma, Nurminen, and Tyrvainen]. The generation is based on a structural model of a device and about available diagnostic tests. Additional information regarding test costs, fault probabilities, and fault types can be used to minimize the average testing effort. This system can be used in a CAD environment to ensure that testability issues are considered in the product development phase.

DMG contains a graph editor, which is built as an application on top of the ISI Grapher. It is used to display and modify diagnostic networks. Such networks consist of interrelated flavors, and often require editing and debugging; the ISI Grapher greatly facilitates these tasks. The main command menu of the ISI Grapher was customized for this application by adding to it several editing commands that act upon the nodes/flavors of the network. Operators that require two node arguments were implemented by clicking-and-saving the arguments into some variable (using a menu command) and then executing the operator (using another menu command). The Nokia group also ported the ISI Grapher onto Apollo workstations; they reported that the port was relatively smooth and took a couple of weeks, including the changes and modifications that they needed to implement in order to fully adapt the ISI Grapher for use within the DMG system.

6.6. Alcoa's Document Management System

Alcoa Inc. is developing EGADS (Electronic Guidance and Documentation System), a system that manages documentation for a rolling mill [Van Sickel, Sierzega, Herring, and Frund]. Hardware and software supplied by various vendors is integrated with specialized user interfaces to allow a diverse group of users to access the documentation based on immediate need. Combining video-disk technology with the Hypertext paradigm, the final database is expected to span some 30,000 pages of text, pictures, and diagrams.

The hierarchical structure of the text/documentation induces a natural directed graph on the database. Users often find it helpful to navigate through the information in a hierarchical fashion; seeing a "picture" of the hierarchy graph is these situations is extremely helpful to the user. When the information is originally entered into the system, these structure trees are built and filled with information incrementally. Users may browse through the documentation by navigating through these families of trees, by keywords, topic lists, or some other ordering. Originally the EGADS system used an ad hoc grapher, but when Alcoa learned of the ISI Grapher, they decided to substitute it into their system, primarily due to its portability and speed.

6.7. IAIMS: Integrated Academic Information Management System

As part of a project to create an IAIMS (Integrated Academic Information Management System) funded in part by the National Library of Medicine, the Baylor College of Medicine is developing a technologic framework for task coordination and information sharing in biomedical work groups. This framework is called the Virtual Notebook to suggest a technologically extended analog of the ordinary laboratory notebook. The principal features of the Virtual Notebook are: a procedure for facilitating task assignment and coordination in the group, a mechanism for sharing ideas among members of the group, automatic procedures for importing relevant information into the group from external sources such as libraries, and an integration of these functions through the use of a few common representational concepts, notably the Hypertext paradigm.

IAIMS will use the ISI Grapher as a graphical tree browser for viewing and navigating the collection of information nodes in the Hypertext component of our Virtual Notebook System, as well as an interface tool to represent various structured vocabularies available in the Virtual Notebook. The most notable of these is the MeSH (Medical Subject Heading) vocabulary of about 18,000 terms maintained by the National Library of Medicine. Within the tree representation of a vocabulary, the user will be able to explore its hierarchy and select terms for use within queries to various data bases available to the system.

6.8. University of Nuernberg' FORK System

The goals of the FORK system are the implementation of a primarily object-oriented knowledge representation system and its application to the design and fault diagnosis of technical systems [Beckstein, Goerz, and Tielemann]. Whereas the kernel of the FORK knowledge representation system is completely object-oriented, the system as a whole integrates a variety of different programming styles. Via an extension for rule-oriented programming, the expressive power of the FORK system is greater than that of LOOPS. As an application of the rule-oriented component, a constraint language has been implemented, which plays an important role in our approach to the design and fault diagnosis of technical systems. The ISI Grapher is used in the FORK system to display, browse though, and manipulate flavor and object hierarchies.

7. Additional General Application Areas

This section describes additional potential application areas that could greatly benefit from the usage of a grapher.

7.1. Idea Outliners

Tables of contents, indexes, and outlines (such as the ones manipulated by idea-processors) are all naturally occurring hierarchies that would serve as excellent targets for building applications using a grapher. In fact, ExperTelligence Inc. has already produced a Grapher application that converts a textual outline into a graph; tabs are used in the text to specify the "level" of a piece of text in the hierarchy. This allows users to get a better feel for the structure of their documents and papers.

7.2. Databases

Many databases are hierarchical in nature, and would thus admit the use of a grapher quite naturally. The ISI Grapher could be used to provide an alternate front-end to a relational database, depicting pictorially the relations between the database objects; certain relations would become more apparent when displayed pictorially. This may prove to be of considerable benefit in many environments.

7.3. Hypercard

Apple's recently introduced Hypercard information management system is fundamentally hierarchical. In Hypercard, users navigate through a set of "stacks," each containing "cards" (pieces of information) and pointers to other stacks/cards, inducing a directed graph. Hypercard users often lose track of his global location in the hierarchy of stacks/cards. The ISI Grapher could easily be tailored to discern and display a given Hypercard hierarchy, providing the user with a global picture that will assist in navigation through a complex set of stacks. Of course a grapher could be built entirely within Hypercard using the Hypertalk language, but such a task would involve considerable re-implementation effort.

7.4. Bibliographical Citations

The relation of bibliographical citation gives rise to a natural partial order. Indeed the set of references to all the technical papers ever written thus induces a directed graph upon these references, where nodes represent publications and edges represent the relation "is referring to" or "cites." It is often desirable to inspect or navigate through this graph; indeed this is an integral (and time-consuming) part of normal research. A grapher would be an obvious asset in this situation.

7.5. Algorithm Animation

Recently Bob Sedgewick devised a system to animate algorithms [Brown and Sedgewick]. Users of this system program algorithms while using certain conventions to instruct the system in how to dynamically depict the execution graphically. Once execution commences, the user sees a "picture" of his algorithm in action, an often enlightening experience. This would also be an ideal area to apply a grapher, since graphs (and in particular trees) are some of the most common of data structures manipulated by programs. For example, in a balanced-tree data structure scheme, the changing tree may be dynamically displayed during run-time, yielding a pictorial depiction of the current structure of the tree.

7.6. Academic Genealogy

In a recent issue of the ACM SIGACT (Association of Computing Machinery's Special Interest Group on Automata and Computational Theory), David Johnson published an extensive list of the most renowned computer scientists and their academic advisors [Johnson]. The advisor/advisee relation naturally gives rise to a directed (and hopefully acyclic) graph; it would be amusing to display and browse through this information using the ISI Grapher. Naturally, real family trees could also be presented using a grapher.

7.7. Object-Oriented Environments

In object-oriented programming environments, various system entities are modelled as objects that posses certain behaviors and to which various "messages" can be sent. Such a set of objects is typically arranged in a hierarchy that denotes a particular partial order among the objects with respect to inheritance of certain properties and methods. When a new object is created it must be added into a specific place in the object hierarchy, and when such a hierarchy is built and debugged, it becomes important (but difficult) to keep track of the relationships among the entities in the hierarchy. From this point of view, the situation here parallels that of the knowledge representation scenario and could therefore greatly benefit from pictorially depicting these relationships, alleviating various editing operations on the partial order described here.

7.8. Program Structure, Function Call Hierarchies, and Debuggers

In a programming environment, lexical scoping of program code within other pieces of code impose a natural partial order with respect to containment of the various routines or functions of the software system being developed/debugged. Given a function, it is often of interest to the programmer to know which functions call it, so that each of those may be edited in turn. Conversely, the programmer may want to know which functions are called by a particular function, for similar purposes. A primitive facility of this type, called Masterscope, was included in the Xerox InterLISP-D system in the early 1980's.

If the various functions (or any other "units" of code) as well as their relationship to one another were represented as a directed graph, it would be much easier to obtain this information and systematically edit many functions in succession without loosing track of the complex sequence of necessary edits. Suppose that instead of representing static lexical scoping, one would graph a dynamic (run-time) calling sequence; it is not difficult to imagine that such a facility could be used as an interactive visual debugger and would greatly facilitate certain debugging situations. For example, the programmer may wish to single-step through the hierarchy of function calls and keep the growing tree of function-calls visible; this tree would correspond to the sequence of function calls induced by the progressing computation.

7.9. File Systems

Most file system are hierarchical in design, where directories can recursively contain files and other directories. If files are considered as nodes and the directory containment is relation is considered as edges, a file system naturally gives rise to a directed acyclic graph. There are numerous situations where it would be desirable to visually observe a picture of this graph and be able to manipulate the files/directories by operating on this graph (using the mouse.) For example, this scheme would save considerable time and error in moving files and directories around, an operation that typically requires memorizing and typing long path names.

7.10. Visual Programming

Many computer manufacturers have made their user interfaces highly visual, incorporating windows, icons, menus, and bitmaps as an integral part of the operating system itself; the most well-known example of this design philosophy today is the Apple MacIntosh. A large and diverse number of research projects around the world have continued to develop these ideas, and indeed these efforts collectively constitute a substantial and rapidly growing area of research called *visual programming.*

Research in visual programming has produced systems where actual programming of the machine is carried out in a visual manner, and rather than type in code, icons are dragged, objects are clicked, and various entities can be grouped, ordered and connected mainly via mouse interaction. An example of such a prototype system is called HI-VISUAL and was presented in the 1987 Workshop on Visual Programming in Linkoping, Sweden [Hirakawa, Iwata, Yoshimoto, Tanaka, and Ichikawa, 1987]. It is clear that a grapher would integrate very nicely into such an environment.

8. Other Graphers and Related Work

A notable effort to produce a graph browser called Grab was introduced in [Meyer] and further developed in [Rowe, et al, 1987], where a system to visually display graphs was implemented. Unfortunately for AI researchers, it was written in C. An additional problem was the usage of numerous time-consuming heuristics (to optimize edge-crossings, for example), rendering the system very slow when laying out large graphs.

Another scheme for drawing graphs is proposed in [Lipton, North, and Sandberg, 1985]. To draw a graph, this scheme entails detecting and exploiting various properties of the given graph with respect to symmetry and the induced automorphism group. While possessing some mathematical elegance, such a scheme can hardly be expected to yield an efficient implementation. It is recognized that systems which run very slowly but optimize layouts to some degree have their applications, but for our purposes, we regard speed as having paramount importance: users are not likely to tolerate layout times measured in hours.

An experimental graph-layout system was produced by the Symbolics Corporation in early 1985 for internal use. However, its heavy dependence on flavors and other specialized Symbolics features, has made it completely non-portable. Additionally, this system used so much space, that attempting to use it on a graph with more than a couple of hundred nodes would typically lead to hopeless disk thrashing (due to massive swapping). In contrast, the ISI Grapher has been successfully used on graphs of up to 25,000 nodes without incident.

In the University of Karlsruhe, West Germany, a knowledge-based graphical editor named EDGE has been developed [Tichy and Ward, 1987] and extended to handle various types of specialized graphs such as Pert charts [Tichy and Newbery, 1987]. This system offers several interesting capabilities such as zooming and ability to specify hierarchical abstract graphs, where certain subgraphs may be treated as individual units.

David Harel of the Weizmann Institute of Science has developed an extensive methodology to model, design, analyze, and display complex systems [Harel]. This impressive methodology, called *statecharts* and *higraphs*, was implemented in a commercial system called Statement1 and is currently being marketed by Ad Cad, Inc [Ad Cad]. Statement1 is a very large and versatile package, retailing in the low five-figures; it allows the specification of the structure and semantics of arbitrary systems, and allows analysis of system behavior under numerous conditions, as well as automatic checking of certain correctness and consistency properties. The system entities that may be modelled using the statechart paradigm within Statement1 include states, events, conditions, transitions, actions, activities, signals, variables, modules, and channels.

Various other schemes to layout trees were proposed previously, such as the ones in Reingold and Tilford [1981], Vaucher [1980], and Wetherell and Shannon [1979]. The earliest similar implementation of a grapher the author is aware of is the Xerox InterLISP-D grapher, running on the Xerox 1100 series workstations in the late 1970's. A very extensive annotated bibliography of graph layout algorithms is given in [Eades and Tamassia]; a particularly large body of research has been devoted to recognizing and graphing planar graphs.

9. Obtaining the sources

The ISI Grapher currently runs on several different kinds of workstations, including Symbolics, TI Explorers, SUNs, HP Bobcats, Apollo workstations, and the MacIntosh II. Information regarding the ISI Grapher as well as the source code may be obtained by contacting the author: Gabriel Robins, USC/Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, California, 90292-6695, U.S.A., ARPAnet address GABRIEL@VAXB.ISI.EDU. Other papers regarding the grapher are also available upon request [Robins, 1987] [Robins, 1988]. To obtain the MacIntosh implementation (among others), contact ExperTelligence Inc., 5638 Hollister Avenue, 3rd Floor, Goleta, California 93117, U.S.A., (805) 967-1797.

10. Summary

We described how various end-user applications are built based on the ISI Grapher, and illustrated the process of application-building via several examples of existing Grapher-based applications. Next we summarized numerous current research projects which already utilize the ISI Grapher in prototype systems. Finally we enumerated various broad application areas that would greatly benefit from the introduction of a tool such as the ISI Grapher.

11. Acknowledgements

I owe **Ron Ohlander** many thanks for his solid support and leadership throughout the ISI Grapher effort. I would like to thank the following individuals for providing me with specific information regarding their projects: **Norm Sondheimer, Bill Mann, Bob Mac Gregor, Bob Neches, Len Friedman, Paul Rosenbloom, Bob Kasper, Ray Bates, Chin Chee, Kai Yue, Mike DeBellis, G. Streck, David Mittman, Pasi Tyrvainen, Andrew Burger**, and **John Allen**. Special thanks go to **Dennison Bollay, John Forge**, and **Dean Ritz** of ExperTelligence Inc. for finding the initiative to port the ISI Grapher to the MacIntosh under the ExperCommon Lisp® system, and undertaking to market the resulting product. It is due to their energy and efforts that the ISI Grapher is now commercially available to the public, the first tool of its kind to run on any personal computer. **Doug Johnson** and **Mark Watson** deserve thanks for porting the ISI Grapher to Allegro Coral LISP®. **James Laurus** deserves credit for porting the ISI Grapher to SUNs under Franz and X, **Chin Chee** has patiently ported the Grapher onto HP Bobcats, and the Nokia group have undertaken ports of the Grapher to Apollo workstations; I thank them all. I also thank **Larry Friedman** for his help with the typesetting. Special Thanks go to **Victor Brown** for his very professional assistance throughout the proofreading and publication process.

12. Bibliography

Ad Cad, Inc., The Languages of Statement1, Cambridge, MA, July, 1986.

Ad Cad, Inc., Statement1 User's Guide, Cambridge, MA, August, 1986.

Cumming, S., and Albano, R., A Guide to Lexical Acquisition in the JANUS System, USC/Information Sciences Institute, Marina Del Rey, CA, Technical Report ISI/RR-85-162, 1986.

Beckstein, C., Goerz, G., and Tielemann, M., FORK: A System for Object- and Rule-Oriented Programming, University of Erlangen-Nuernberg, Germany.

Brown, M., and Sedgewick, R., A System for Algorithm Animation, SIGGRAPH '84 Conference Proceedings, Minneapolis, Minnesota, Vol. 18, No. 3, pp. 177-186, July 23-27, 1984.

Eades, P., and Tamassia, R., Algorithms for Graph Drawing: an Annotated Bibliography, Technical Report No. 82, University of Queensland, Australia, July, 1987.

Harel, D., Statecharts: A Visual Approach to Complex Systems, CS86-02, The Weizmann Institute of Science, Rehovot, Israel, March, 1986, (also appearing in Science of Computer Programming.)

Hirakawa, M., Iwata, S., Yoshimoto, I., Tanaka, M., & Ichikawa, T., HI-VISUAL Iconic Programming, Workshop on Visual Languages, Linkoping, Sweden, August 18-21, 1987.

Hovy, E., Planning Coherent Multisentential Text, Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics Buffalo, New York, June, 1988.

Johnson, D., Genealogy of Computer Science, Association of Computing Machinery, Newsletter of the Special Interest Group on Automata and Computational Theory.

Kaczmarek, T., Mark, W., & Wilczynski, D., The CUE Project, Proceedings of SoftFair, July, 1983.

Kaczmarek, T., Bates, R., and Robins, G., Recents development in NIKL, AAAI, Proceedings of the Fifth National Conference on Artificial Intelligence, August, 1986.

Kasper, R., An Experimental Parser for Systemic Grammars, Proceedings of the 12th International Conference on Computational Linguistics, Budapest: August, 1988 (also available as USC/Information Sciences Institute Reprint RS-88-212).

Laird, J., Soar User's Manual (Version 4), Technical Report ISL-15, Xerox Palo Alto Research Center, 1986.

Laird, J., Newell, A., & Rosenbloom, P., Soar: An Architecture for General Intelligence, Artificial Intelligence, Vol. 33, pp. 1-64, 1987.

Lipton, R., North, S., & Sandberg, J., A Method for Drawing Graphs, ACM Computational Geometry Conference Proceedings, pp. 153-160, June, 1985.

Lounamma, P., Nurminen, J., and Tyrvainen, P., DMG - A System for Diagnostic Modelling and Generation, Nokia Research Center, Helsinki, Finland, March 1988.

Mac Gregor, R., & Bates, R., The LOOM Knowledge Representation Language, Proceedings of the Knowledge-Based Systems Workshop, St. Louis Missouri, April 21-23, 1987.

Mann, W., An Overview of the Penman Text Generation System, Proceedings of the National Conference on Artificial Intelligence, pp. 261-265, August, 1983.

Mann, W., and Matthiessen, C., Nigel: a systemic grammar for text generation, Benson, R., & Greaves, J. (Eds.). Systemic Perspectives on Discourse: Selected Theoretical Papers from the Ninth International Systemic Workshop. Norwood, NJ: Ablex, 1985 (also available as USC/Information Sciences Institute, Technical Report ISI/RR-83-105, 1983).

Mann, W., and Thompson, S., Rhetorical Structure Theory: A Theory of Text Organization, In The Structure of Discourse, Edited by Livia Polanyi, Ablex, Norwood, N.J., ISI/RS-87-190, 1988.

Meyer, C., A Browser for Directed Graphs, Technical Report, Department of Electrical Engineering and Computer Science, University of California, Berkeley.

Mittman, D., AUDREY: An Interactive Simulation and Spatial Planning Environment for the NASA Telerobot System, Proceedings of the Artificial Intelligence and Advanced Computer Technology Conference, Long Beach, California, May, 1988.

Neches, R., FAST Workstation Project Overview, Technical Report, USC/Information Sciences Institute, December, 1987.

Neches, R., DeBellis, M., & Yen, J., BACKBORD: Beyond Retrieval by Reformulation, Proceedings of the Workshop on Architectures for Intelligent Interfaces: Elements and Prototypes, Monterey, CA, March, 1988.

Reingold, E., & Tilford, J., Tidier Drawing of Trees, IEEE Transactions on Software Engineering,

SE-7, no. 2, pp. 223-28, March, 1981.

Robins, G., The NIKL Manual, Intelligent Systems Division Report, USC/Information Sciences Institute, April, 1986.

Robins, G., The ISI Grapher: A Portable Tool for Displaying Graphs Pictorially, Invited Talk in Symboliikka '87, Helsinki, Finland, August, 17-18, 1987, (reprinted in Multicomputer Vision, Levialdi, S., Chapter 12, Academic Press, London, 1988).

Robins, G., The ISI Grapher Manual, ISI Technical Manual/Report ISI/TM-88-197, USC/Information Sciences Institute, Marina Del Rey, February, 1988.

Rowe, L., Davis, M., Messinger, E., Meyer, C., Spirakis, C., & Tuam, A., A Browser for Directed Graphs, Software - Practice and Experience, Vol. 17(1), pp. 61-76, January,1987.

Sondheimer, N., and Nebel, B., A Logical-Form and Knowledge-Base Design for Natural Language Generation, Proceedings of the National Conference on Artificial Intelligence, Philadelphia, August, 1986.

Steier, D., Laird, J., Newell, A., Rosenbloom, P., Flynn, R., Golding, A., Polk, T., Shivers, O., Unruh, A., & Yost, G. R., Varieties of Learning in Soar: 1987, Proceedings of the Fourth International Workshop on Machine Learning, Edited by P. Langley, Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1987.

Supowit, K., & Reingold, E., The Complexity of Drawing Trees Nicely, Acta Informatica, Vol. 18, pp. 377-392, 1983.

Tichy, W., & Ward, B., A Knowledge-Based Graphical Editor, Technical Report, Universitat Karlsruhe, Fakultat Fur Informatik, Germany, January, 1987.

Tichy, W., & Newbery, F. Knowledge-Based Editors for Directed Graphs, First European Software Engineering Conference, De Strasbourg, France, September 9-11, 1987.

Van Sickel, P., Sierzega, K., Herring, C., and Frund, J., Documentation Management for Large Systems of Equipment, Aluminum Company of America, Alcoa Technical Center, Alcoa Center, PA, February, 1988.

Vaucher, J., Pretty-Printing of Trees, Software - Practice and Experience, 10, pp. 553-561, 1980.

Wetherell, C., & Shannon, A., Tidy Drawing of Trees, IEEE Transaction on Software Engineering, 5, pp. 514-520, September, 1979.

INFORMATION
SCIENCES
INSTITUTE

4676 Admiralty Way/Marina del Rey/California 90292-6695