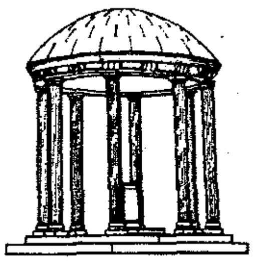
WE:
A Writing Environment for Professionals

TR86-025

August 1986

John B. Smith
Stephen F. Weiss
Gordon J. Ferguson
Jay D. Bolter
Marcy Lansman
David V. Beard



The University of North Carolina at Chapel Hill
Department of Computer Science
CB#3175, Sitterson Hall
Chapel Hill, NC 27599-3175

A TextLab Report.
UNC is an Equal Opportunity/Affirmative Action Institution.

Introduction

Technical and scientific professionals are writers. Regardless of title or job description, they write. Most spend 25-75% of their time doing something related to writing – gathering and organizing information, writing *per se*, revising, talking with others about something they have written, giving oral presentations accompanied by documents, etc. They write many different forms – letters, reports, specifications, plans of various sorts, proposals, justifications, articles, oral presentations, to name some of the more prevalent forms. These documents are important. They form the skeleton of the writer's organization. While that skeleton must be fleshed out by other activities, the collection of written documents forms the core. If new tools can lead to more effective documents and can help skilled professionals work more efficiently, the payoffs will be substantial.

Current tools for writing and producing documents fall into four major groups: editors, formatters, checkers, and organizers. The first two are well- established and need no additional comment. Checkers are less universal, but still wide-spread. The most common are the spell-checkers, but style-checkers are also beginning to appear. While those that use table lookup and limited pattern matching are of questionable value, checkers that will eventually include full parsers may have more impact, when they appear. The final group – the organizers – include structure editors and outline processors. The former tend to be mainframe-oriented and are often experimental or demonstration systems; Nelson's hypertext [1] and Engelbart's NLS [2] are early examples. More recently, the microcomputer outline processor has become widespread, but the jury is still out on its value.

Current tools for writing were not designed for professionals. Most were designed for technical writers concerned with layout and physical production or for microcomputer hobbyists. What is needed are tools designed specifically for the sophisticated professionals who use workstations within distributed environments.

We are developing a comprehensive Writing Environment (WE) for this application. Parts of this work are supported by IBM, NSF, and the Army Research Institute. In describing this system, we will emphasize five key concepts:

- the system is based on a cognitive model for written communication
- the system is highly visual
- the system was prototyped in Smalltalk and then ported to Objective C
- the system will be used a series of cognitive experiments
- they system can be extended to other applications

The emphasis placed on cognitive aspects in this description probably needs more explanation. WE is one instance of an increasingly important kind of software that provides users with an environment in which to think or with functions that supplement human cognitive skills. To be successful, these intelligence augmenting systems must reflect the cognitive processes of the people using them. We suggest that a modified development cycle is needed that begins with an explicit cognitive model of the user interacting with the system to perform specific high-level tasks, includes formal testing of the model as well as the software, and ends (the first cycle) with systematic refinement of both. Therefore, our discussion of WE will include not just a description of the system but also its underlying rationale and the methods we used to develop and test it.

Cognitive Model for Written Communication

WE is based on a cognitive model of written communication. The model was derived from a review and synthesis of the literature in cognitive psychology, composition theory, human/computer studies, as well as our own experience. However, it is put forth more as a question than as an assertion. We are testing the model in a series of cognitive experiments and will revise it accordingly. It stresses the structure of information, particularly the transformations writers and readers produce as they write and read documents, and views writing and reading as symmetrical processes in several important respects (Figure 1). In this section, we describe the model, briefly, and then explain how we have used it in designing WE.

Whether readers read a document from beginning to end or jump from one place to another, when they "settle down" to read a passage they do so linearly. That is, they decode a linear sequence of words. However, they do not comprehend linearly. Rather, they comprehend by relating bits and pieces of information to one another hierarchically. They see that several points do, indeed, add up to the conclusion the writer has drawn, or that a general point is supported by the evidence or examples cited. As the process continues, readers relate what they are reading to what they already know. This process is particularly active as new information is integrated into the network of associations that underlies long-term memory. Thus, readers *read*, *comprehend*, and *remember* what they read by transforming information in one structural form into another: from *linear sequence*, to *hierarchy*, to *network*.

The key to the reading process, however, is the hierarchical step. If a document signals its hierarchical structure through features included in it – such as a system of headings, overviews, topic sentences in paragraphes, etc. – readers use these clues to advantage. That is, they read and comprehend the document more quickly and the structure they infer for the document will match more closely that intended by the writer [3]. If such features are omitted from the document – no headings or inconsistent headings, flat narrative, few topic sentences, etc. – to the extent readers understand what they are reading, they will construct their own hierarchy for the document. However, the hierarchy they construct may or may not resemble that intended by the writer, and it takes time! Consequently, organizing expository information into a hierarchical structure and then signaling that structure is a particularly effective strategy for writers to follow.

Writing involves a similar series of transformations, but in reverse order. Writers normally begin with a need to write. The content is likely be scattered through the writer's long term memory or through various external sources, such as books, databases, or other people's heads. The "structure" of that information is likely to be a very loose associative network, derived as the information is brought to consciousness. A key step for the writer, then, is to gather information and to organize it. Most writers do so by constructing a hierarchy, in the form of an outline or a tree. Once the hierarchy has been constructed,

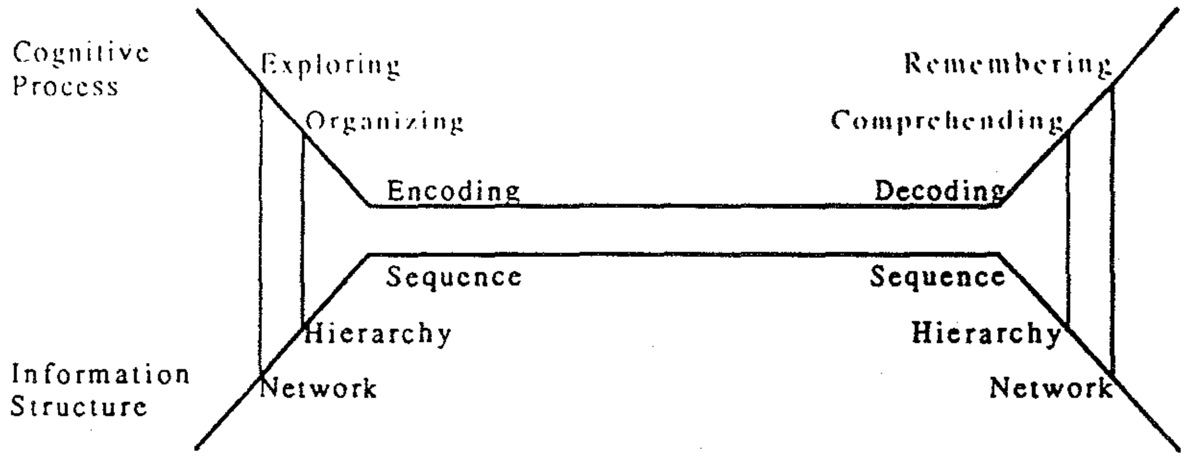


**Figure 1:
Cognitive Model for
Written Communication**

the task of writing becomes a traversal of the hierarchy during which the writer encodes the concepts into prose, graphics, or other forms. Thus, writing involves a similar but opposite sequence of transformations: *network*, to *hierarchy*, to *linear sequence*.

Several conclusions can be drawn. First, writing involves both networks (directed graphs) and hierarchical structures but at different stages of the process. All earlier structure editors with which we are familiar have adopted one principle or the other, but not both. The hypertext family of editors – such as Nelson's hypertext system [1], its Brown University derivatives [4], and NoteCards [5], – support directed graphs. A similar group support hierarchical structures – such as Engelbart's NLS [2], Thinktank [6] and the other outline processors, and XS-2 [7]. While users can construct a hierarchy within a directed graph environment, they may find the environment more supportive when they can voluntarily relinquish some function during certain stages of the process in exchange for greater discipline. Consequently, we have constructed an environment that includes both, permitting writers to develop graphs and hierarchies separately but also to transfer conceptual structures from one mode to the other.

Another key conclusion is that writing requires a number of different cognitive skills – not just linguistic encoding skills. Writers think associatively, hierarchically on a small scale (individual inferences and deductions), hierarchically on a large scale (constructing a single large hierarchy), analytically (as they revise), etc. For many writers, particularly those in scientific and technical fields, these stages also include visual and spatial reasoning. This is particularly true during early exploratory thinking and during the organizational stage. Consequently, we have built our environment around the notion of an abstract space in which users can represent and manipulate concepts visually.

A third, and related, implication is that writing includes both bottom-up and top-down thinking. During early exploration, writers often think bottom-up as they trace paths of associations, gather information, explore various relations, etc. While an entire document can be organized hierarchically by continuing a bottom-up strategy, it cannot be "aimed" easily or reliably using this approach. To focus a document and to insure that it achieves a clearly recognize goal, experienced writers often begin with a single large objective and derive the hierarchical structure from that point. Thus, writers also need tools that let them work top-down. The point is not that one form of thinking or the other is best; both are needed but at different stages of the process. Consequently, the environment we are developing is strongly multimodal.

While cognitive psychology has had a strong impact on human factors studies and the design of computer interfaces, it has had less impact on the underlying architecture and function of systems. In WE, the cognitive model has influenced not just the interface; it is central to the entire design and is a concept that will be evaluated experimentally. Thus, the system itself and the theoretical basis on which it is built emerge as a question: *How do users write and think while working within this particular computing environment?* A substantial part of our effort is directed at answering this question, as we explain below in the section on **Cognitive Experiments.**

Description of WE

Three aspects of WE distinguish it from other writing support systems: the visual interface, its multimodal architecture, and an underlying relational database.

Visual Interface

The interface for WE is based on three major factors derived from the cognitive model:

- writers use a number of different cognitive skills in writing
- writing involves a series of transformations in which information in one structural form is changed into another
- structures can be more easily comprehended, constructed, and manipulated when they are represented visually (e.g., in a tree) than when they are represented linguistically (e.g., in an outline, as in 1.3.2.4).

Consequently, the user interface is distinctly visual and graphical, as opposed to language-oriented.

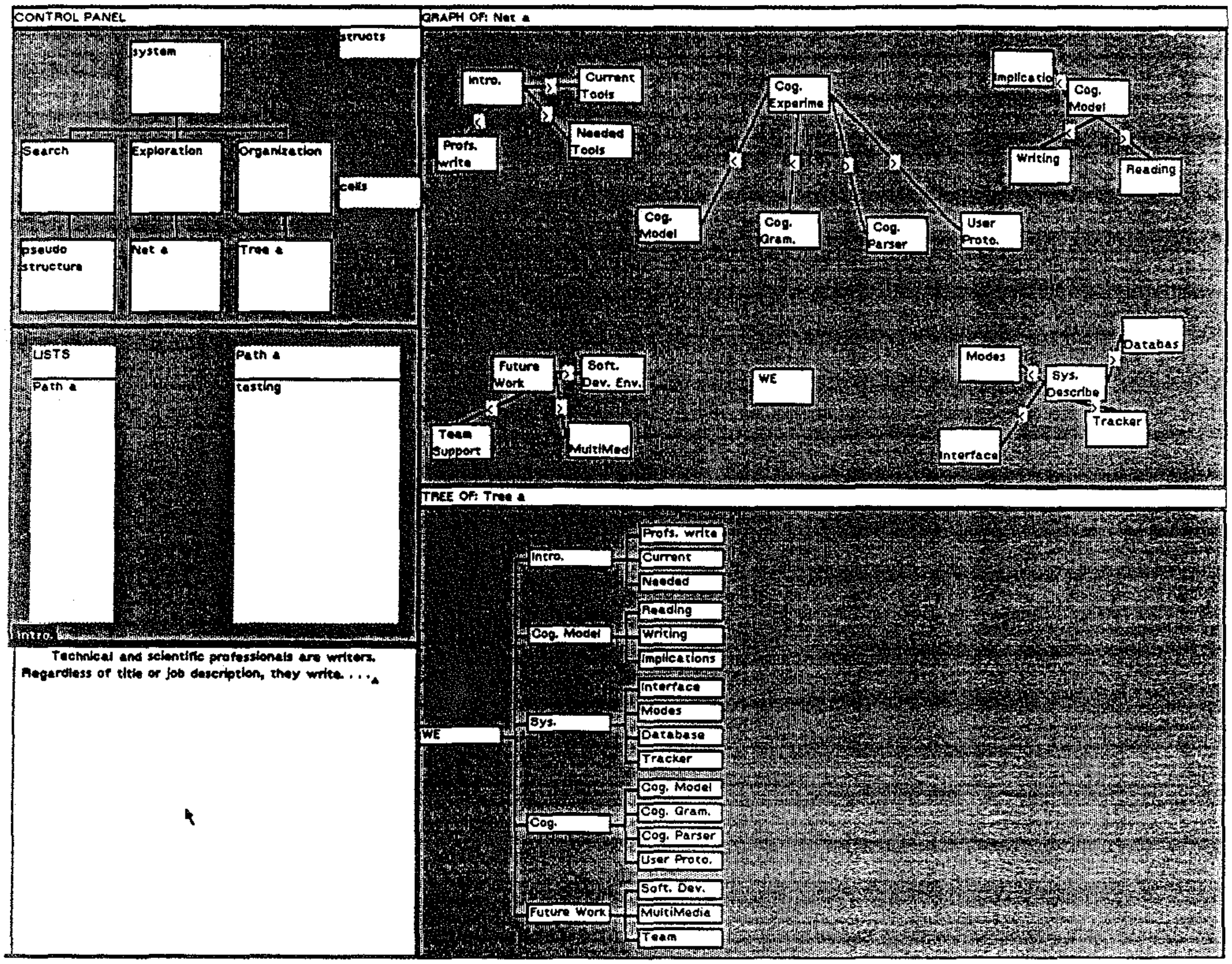
The default layout for the screen shows five tiled windows (Figure 2). The two largest are a *graph* window and a *hierarchical* window. The first supports operations that conform to the rules of a directed graph embedded in a Euclidean space. The second obeys the rules of hierarchies. A smaller window is available for either a text or a graphic editor used to write or draw the *content* of the document, associated as blocks of data with individual nodes. The fourth window is used to *search* the relational database for other structures or nodes that might be inserted into the current document. The last window is a *control panel* for managing the environment. Each window is described in more detail below in relation to its corresponding mode.

Users can easily change the default configuration by resizing and moving the various windows. Thus, the entire screen can be used for the directed graph window during, say, the early brainstorming stage of writing. Or the entire screen can be used to show a tree in hierarchical mode during organization. Another option is to split the screen between a directed graph and a hierarchical window so that small hierarchical substructures can be copied from one mode to the other. (See Figures 2-6, below.)

Modes

A second key architectural feature of WE is its multimodal structure. While the tide of opinion is currently running against such designs, separating the function of the system into separate domains is desirable for this particular application. Since writing involves several different kinds of thinking, we support each with functions specific to that cognitive mode. An hypothesis we will test experimentally is that users will prefer to "drop into" different modes of thinking for different activities, gaining flexibility in some cases, giving it up in others in exchange for greater rigor and consistency.

We expect most writers to begin a project by working in a directed *graph* window. This mode is particularly well suited for bottom-up thinking. Using a mouse, users can



Technical and scientific professionals are writers. Regardless of title or job description, they write. . . .

Figure 2:
WE Default Screen

open a window to cover the entire screen. They can then create nodes at any spot in the windows simply by pointing with the mouse, clicking for a menu, and selecting the "create node" option. (Since the last option selected on a particular menu is retained as the default, subsequent clicks produce additional nodes without further selection.) They can label each with a word or phrase, either when the node is created or later as an editing operation. Users can also move nodes into clusters of related concepts (Figure 3.1) and can join pairs of nodes with directed links to denote specific associations (Figure 3.2).

A second mode/window provides functions that conform to the rules for hierarchies (Figure 4). Users begin in this window by creating a root node and labeling it, as in graph mode. They can then create child nodes under the root, indicating the major divisions of the document. The process of division can be continued until the nodes represent sections that can easily be written, usually a few paragraphs, or represented in a single graphic. A number of structure editing functions are also provided. These permit users to move nodes or branches around in the hierarchy, add and delete both leaf and interior nodes, etc. Users may also import nodes or structures from graph mode into tree mode. That is, they can go back to a directed graph window created earlier and select a node that is a root for a small hierarchical relation; when they return to the hierarchical window, they can point to the place where the branch should be placed and the system will insert the subtree into the tree at that point.

The system provides four different visual representations for hierarchies. The first is a conventional horizontal tree in which parent/sibling relations are indicated by left to right relations (Figure 4). The second is a vertical tree that extends from top to bottom (Figure 5). Zoom and roam functions are provided for each. In fact, since users can open several different windows on the same structure, they can show a small schematic view of the whole tree in one, an enlarged view of a section in a second, and a still larger image of the particular branch being worked on in a third. This is particularly useful for large structures, for team development efforts, or other projects where managing technical complexity is an issue. A third view presents a Chinese box representation of the hierarchy in which child nodes are shown as small boxes inside the larger box representing the parent node (Figure

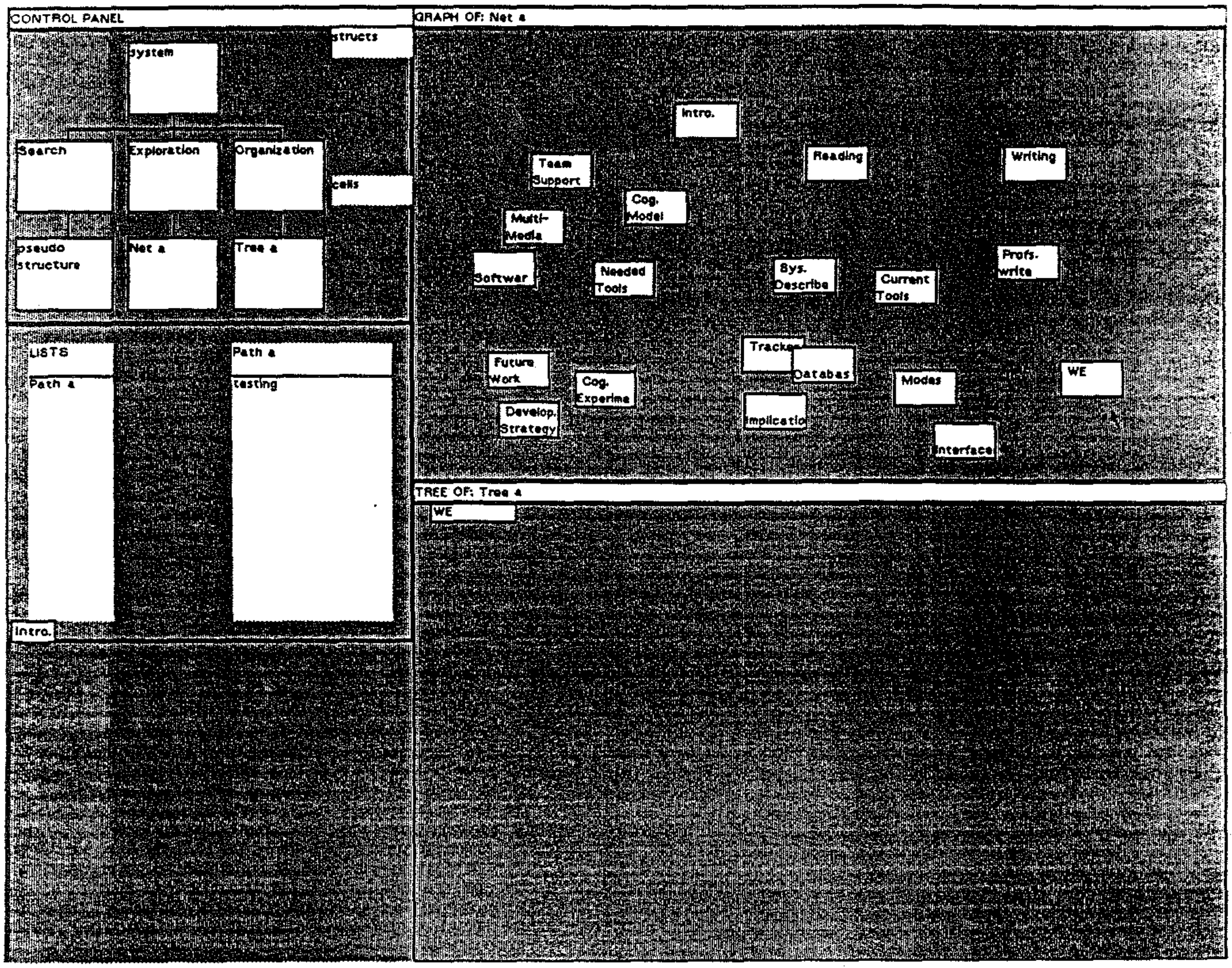


Figure 3:
WE Spatial Graph Mode

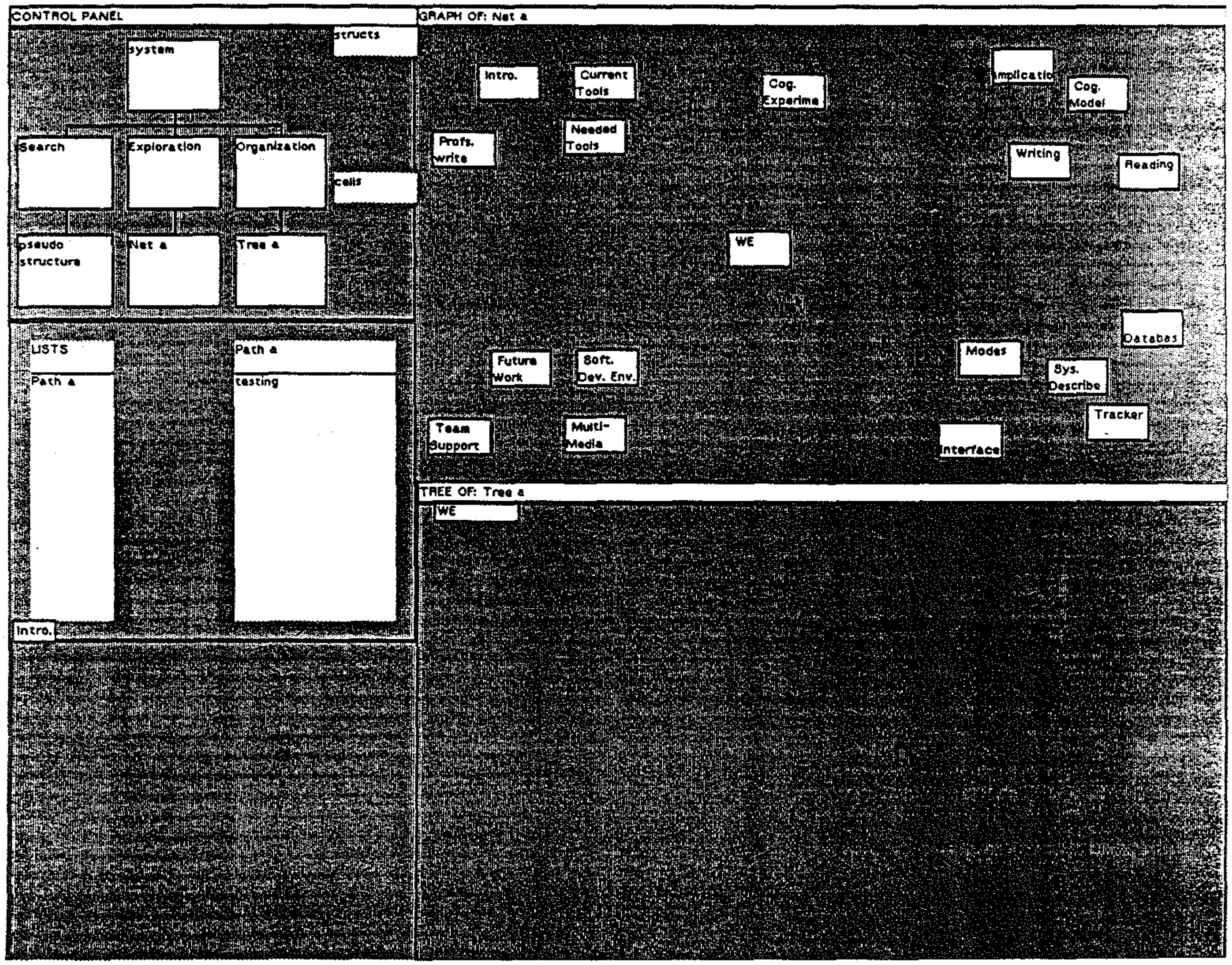


Figure 3.1:
WE Spatial Graph Mode,
Conceptual Clusters

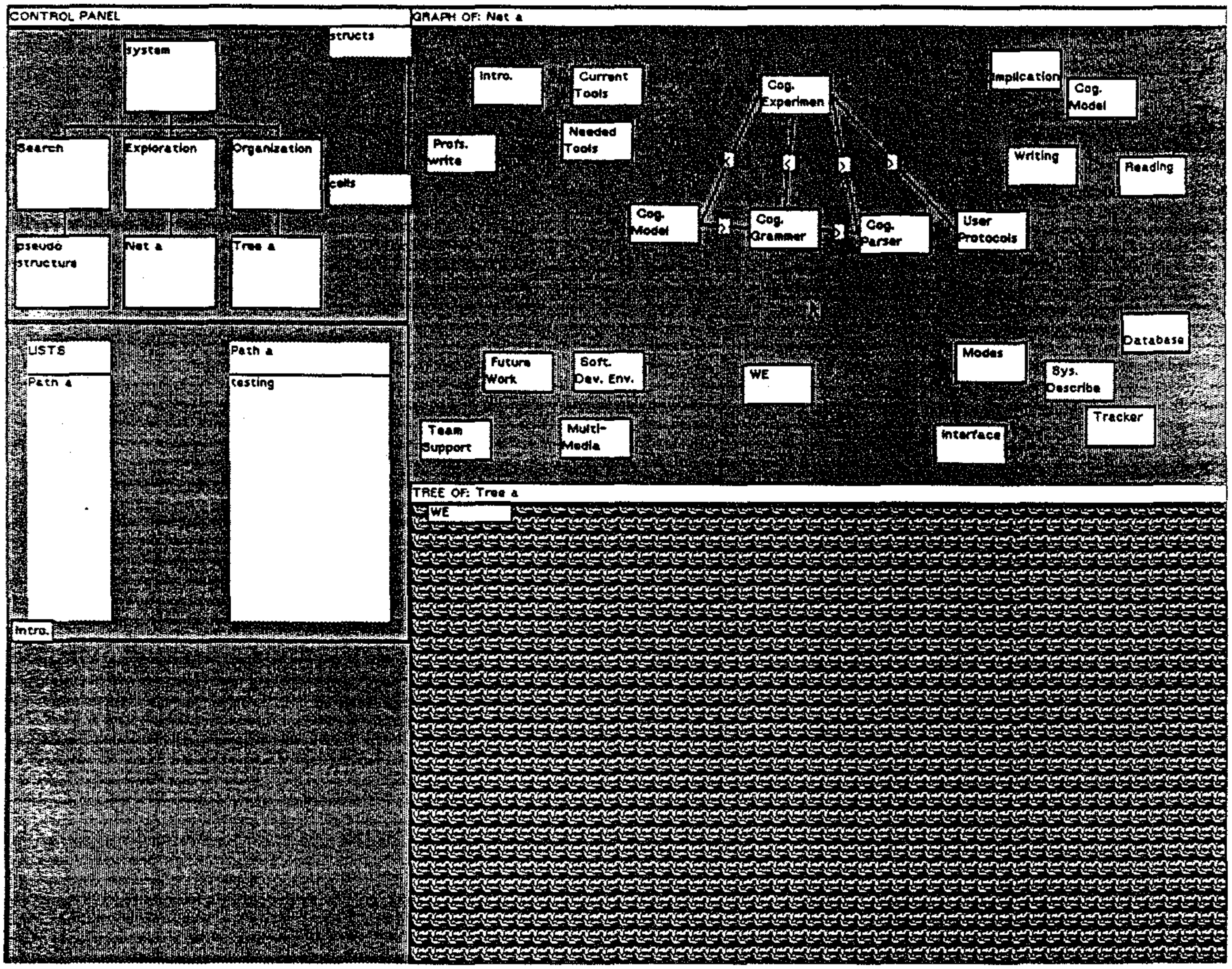
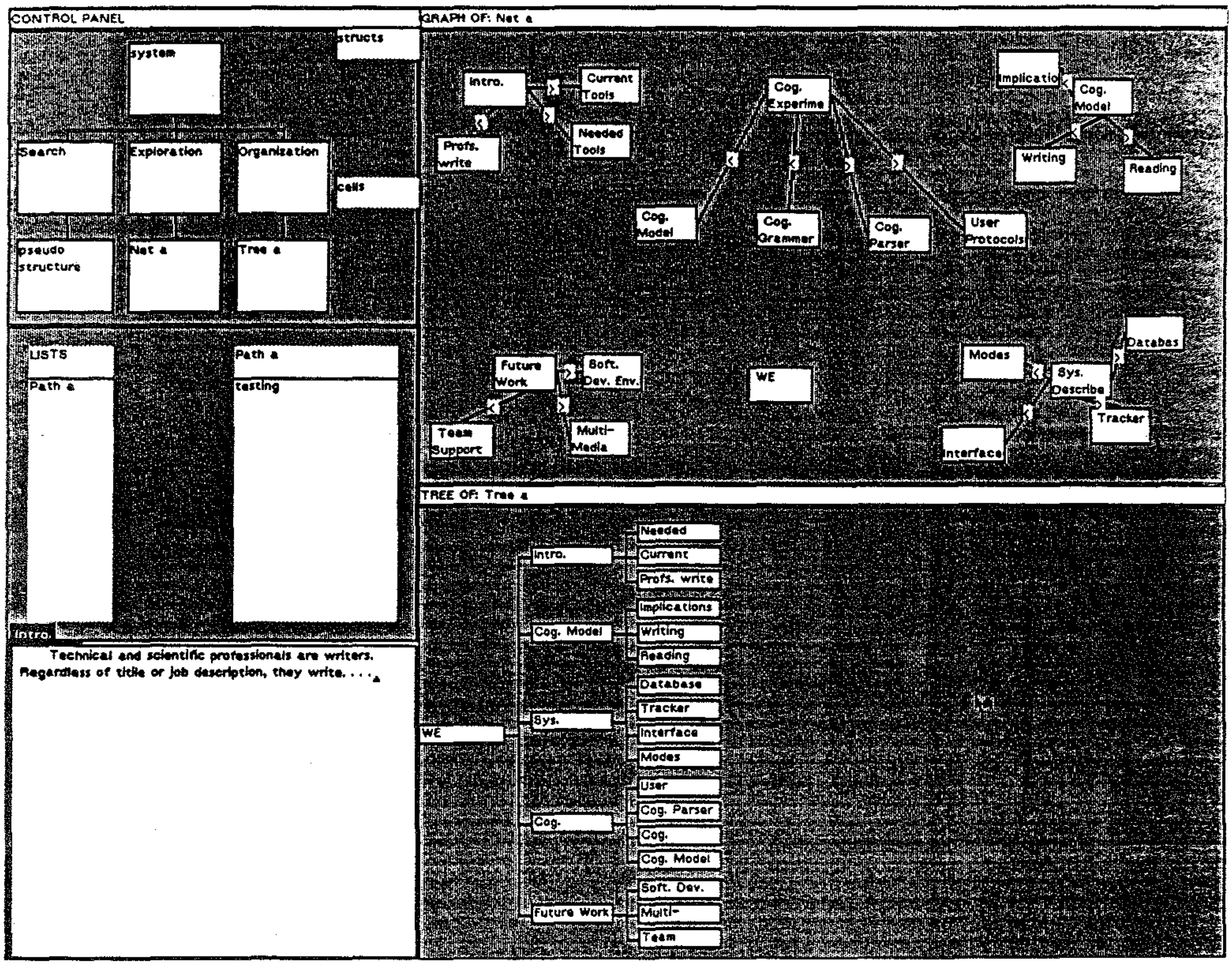


Figure 3.2:
WE Spatial Graph Mode
Conceptual Clusters with
Associative Links



Technical and scientific professionals are writers. Regardless of title or job description, they write. . . .

Figure 4:
WE Hierarchy Mode
Horizontal Tree

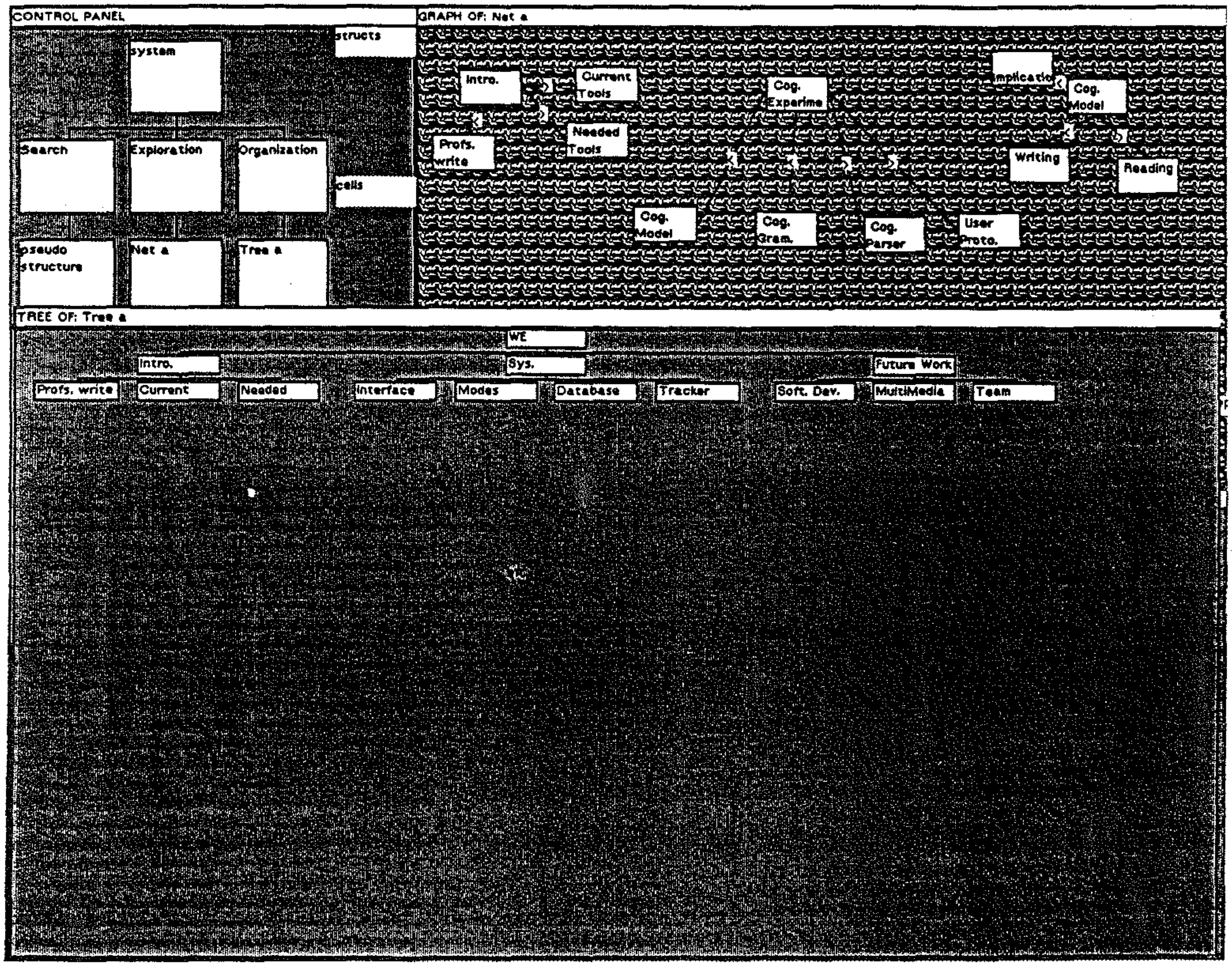


Figure 5:
WE Hierarchy Mode
Vertical Tree

6). Since the system shows only three levels of depth with this view, it provides a form of information hiding. The last view is a standard outline view.

At any point, in either graph or hierarchical mode, the user can open a node and insert *content*. This is done by invoking either a conventional text or graphic editor. Typically, users write a paragraph or several paragraphs or create a single visual image. In this mode, the function provided is that of the particular editor. When users finish with a content unit, they close the node and the content is saved in a file system. Thereafter, whenever a node is moved using any of the structure editor functions, the associated content is also moved along with it. Since a node is a typed object bound to a particular editor/display program, the kinds of data that can be associated with a node can be extended simply by extending the set of types and associated editor/display programs. We describe several planned extensions, in the section on **Future Work**.

A fourth mode helps users *search* the relational database in which nodes, links, and structures are stored. We explain its purpose and function in the following section. Here, we merely call attention to its existence.

All four modes – *graph*, *hierarchy*, *content*, and *search* – are "held together" by a *control panel*. The control panel includes two major fields: a mode tree and a pair of stacks. The mode tree represents the different modes, as first-level children, and the specific named instances of each (i.e., windows), as second-level children. It provides a variety of management functions. For example, to move a buried window to the forefront, users merely point to it in the mode tree and select the appropriate operation. Thus, users can quickly get an overview of the entire "screen space" they have created, including windows covered by other windows. The stacks receive the nodes and structures created by the yank operation explained earlier. They permit users to make copies of several different nodes or structures while working in one mode/window and then selectively move them at their leisure into the structure being created in another.

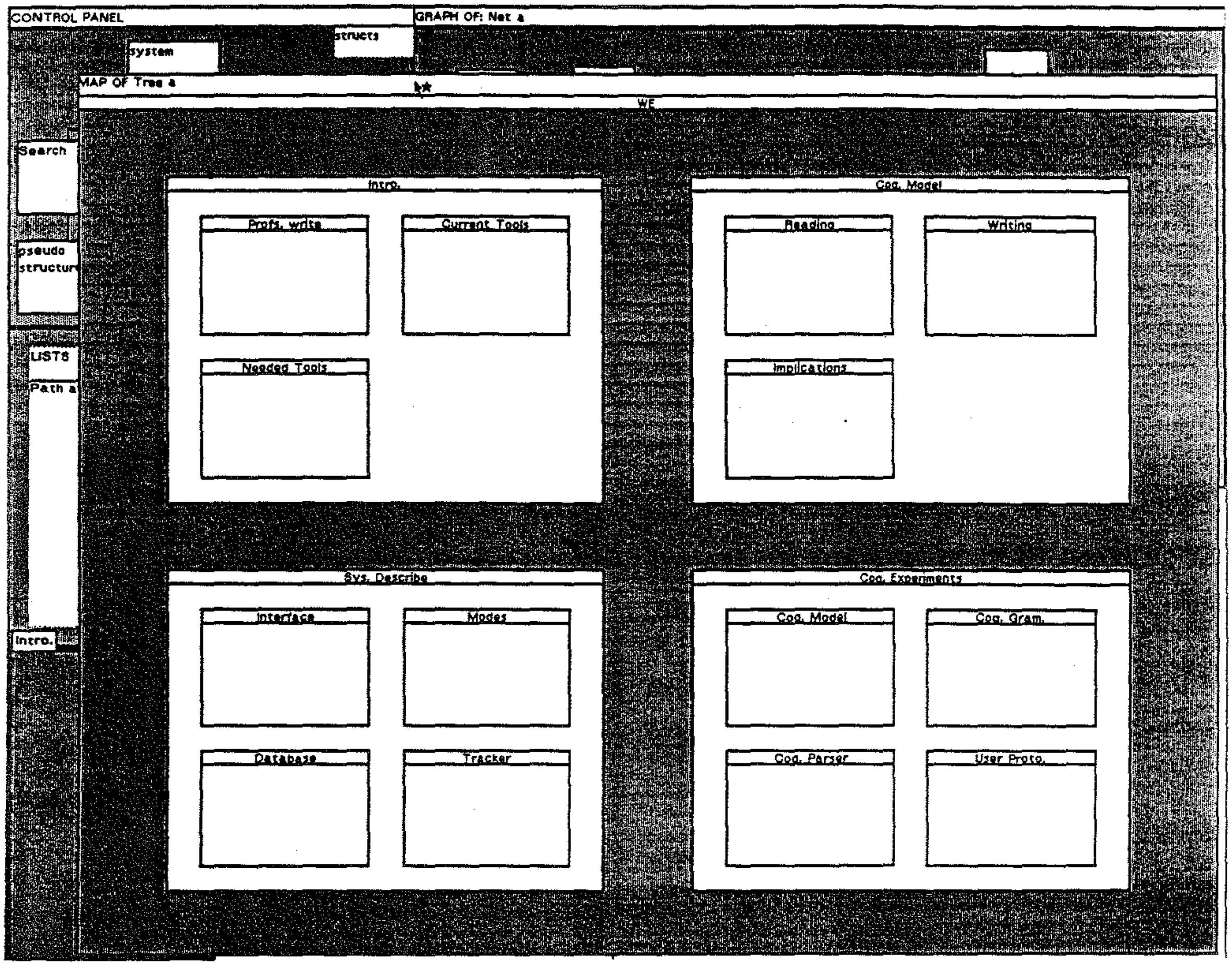


Figure 6:
WE Hierarchy Mode
Chinese Box

Database

A third major innovation in WE is the use of a relational database system as the store for all structural information. The database holds three kinds of entities: ***structures***, ***nodes***, and ***links***. Structures are typed, named sets of links (and, by implication, associated nodes). The type indicates whether the structure is a graph, hierarchy, or path; this information is used by the system to determine the operations that can be performed on the particular structure. Each node is also viewed as a typed object. Associated with it are various attributes that identify the type of content "within" that node and, thus, bind it to an editor/display program; its spatial dimensions in graph-mode space; and both its associative and hierarchical links. Links are attributed pairs of node identifiers. The node identifiers define the directed arc; and the attributes indicate the kind of link (e.g., graph), the structure of which the link is a part, and other system information.

Users can search the database for a structure, node, or link by its identifying label or by its attributes. This is done through the search window/mode, mentioned in the preceding section. In the current system, the database is confined to a single project, but we will extend its definition to permit teams and departments to store collections of documents and other kinds of data. Thus, future users will be able to search the database for information relevant to the current project. Once a usable node or structure is found, it can be imported into the environment and included in the structure currently being developed. A longer term goal is to merge another system we are developing – MICROARRAS [8], an advanced full-text retrieval and analysis system – with WE to support content-based searches, as well.

Implementation

We have followed an unusual path in implementing WE. First, we designed and implemented a prototype system in Smalltalk running on a SUN-3 workstation. Smalltalk provides an object-oriented environment that encourages information hiding and hierarchical modular design in which each level of the system is implemented in terms of the tools defined at lower levels. It also provides a complete development environment including a sophisticated system browser, extensive graphic tools, and access to the full Smalltalk source. Since Smalltalk is an interpreter, changes can be made and tested quickly and easily. The prototype system, shown in figures 2 - 6, provides full functional capability and can support documents up to about fifty nodes. Using it, we were able to test our original design by actually using the system to see how various features worked in conjunction with one another. However, since Smalltalk is not suited for large, high performance applications, we planned from the beginning to port the system to other software and hardware environments.

To facilitate this move, we developed device-independent toolkits for drawing and for managing user interaction with the system. Both toolkits were designed as Smalltalk classes. In Smalltalk, they were implemented directly using methods provided by the system. To port them to other environments, we are writing drivers that use the graphics and window management facilities provided by the target system. We have completed the porting of both toolkits to Microsoft Windows for the IBM PC/AT, and we are currently moving them to X Windows for the SUN workstation.

Finally, we are porting the entire system from Smalltalk to Objective C, a synthesis of Smalltalk and C developed and marketed by Productivity Products International, Inc. Objective C provides a large-grain structure of classes, methods, and inheritance characteristics nearly identical to Smalltalk. But, it also provides the small-grain capability to repalace system primitives with C functions for greater speed and processing efficiency. While we can foresee the possibility of translating Smalltalk classes into Objective C automatically, for the present we must still rewrite the syntax manually. This is largely a direct, line-for-line translation that requires virtually no changes to overall system architecture.

Cognitive Experiments

As we noted earlier, WE was designed in accord with a cognitive model of the writing process. We are using the system as an observational instrument in a series of formal experiments to evaluate that model as well as other cognitive hypotheses and to test specific system features and representation schemes. In this section, we will not describe these experiments in detail, but rather the technical features of the system that support them.

A built-in tracking facility permits us to record the actions of users at a functional or operational level. Thus, we can observe the sequence of operations employed to create nodes, move them into spatial clusters, link them into associative relations, etc. Each operation is recorded along with the time it was performed and its associated parameters and stored in the same relational database as the document. These data constitute a high-level concurrent protocol of the session, collected unobtrusively and in a machine-readable form ready for analysis.

Traditional approaches to concurrent protocols have employed video recordings of users interacting with a system, "thinking aloud" protocols in which users attempt to narrate the thinking processes they are using, and keystroke records. All three result in enormous volumes of data. Both video tape and thinking aloud protocols also require extensive encoding to produce machine-readable data that can be analyzed. Thinking aloud protocols present further theoretical problems for situations where verbalization is not an integral part of the task being performed, such as tasks in which users manipulate spatial forms [9]. This is exactly the situation presented by our system – writers, particularly during the exploratory and organizational phases of writing, often think spatially and abstractly, rather than verbally. For these reasons, we believe the relatively large-grained record produced by the tracker, representing the operational history of a session will provide more usable and reliable data for our purposes than more traditional protocols.

The cognitive model on which the system was built is expressed as a grammar. While it superficially resembles the GOMS model of Card, Moran, and Newell [10], it goes beyond their framework. One distinction is the extension to a quasi context sensitive grammar. Context free productions are not powerful enough to handle user operations for this application. More importantly, the grammar can be used to develop a parser to analyze the protocols generated by the tracker. The trees that result from parsing the sequence of operations performed by a user during a session constitute a formal representation of that user's strategy for the session. Thus, we have a concrete way of comparing the strategies of different groups of users, such as those of experts and novices. Additional display and statistical analysis techniques will permit us to play back a user's session, graph distributions of specific operations over time, look for "cognitive rhythms", and note combinations of functions frequently used together.

On the basis of this information, we will revise the cognitive model, as appropriate, and then refine the system. Thus, we hope to set-up a development loop in which the system is designed in accord with a well-defined model of the user's interaction with the system at a cognitive level, implemented in a fast prototype environment for initial testing, ported to an actual-use configuration for more extensive experimentation, and then systematically revised in accord with empirical results.

Future Work

While the system we have described is intended as an aid for professionals who write, it can be extended to other applications. Basically, the system provides a general visual interface for creating, editing, and displaying directed graphs of abstract nodes that can be associated with typed data. A number of other applications can be modeled in these terms. We plan to extend our work into three other areas.

First, we want to extend the system from a single user system to a multiple user system for distributed environments. The central database underlying the system can facilitate team development of a structure and collaborative efforts. We also want to add a simultaneous teleconferencing facility in which several team members can view the same display on their respective workstations while they work on the same underlying data structure. This will be done in an environment in which switchable voice and video can be added to permit them to discuss their work and to see one another. We will also try to extend the cognitive model to characterize the cognitive/communication acts of a team of individuals working together to construct a single, integrated conceptual structure and then test that model, analogously.

Second, we will extend the system to include other forms of data. Since a node is an abstract, typed entity, other forms of content can be included by extending the set of node types and by providing the necessary display and edit functions. The system can, thus, include sound and video sequences from conventional video disks as well as emerging cd/roms by including in the nodes the instructions necessary for the bound function to display that data.

A third application will extend the system to form a vertically integrated environment for software development. The primary extension necessary is to make the graph multi-dimensional. In this way, one two-dimensional plane can be assigned to functional specifications, a second to source code, a third to executable modules, a fourth to test results, etc. While each level represents a large field of research, we will limit our work to a small subset of tools in each – such as Objective C and C in the source level – so that we can concentrate on issues of interaction between levels.

Acknowledgments

A number of people have contributed their ideas and their labors to this project. We wish to thank the following graduate students: John Walker, Valerie Kierulf, Greg Berg, Paulette Bush, Yin-Ping Shan, and Katie Clapp. We also wish to thank Myra Reaves for her help in preparing the manuscript for this report.

References:

[1] Nelson, T.H. (1981). *Literary Machines*, Swarthmore, PA: Nelson, T.H.

[2] Engelbart, D., & English, W. (1968). A research center for augmenting human intellect. *Proceedings of 1968 FJCC*, 33, Part I, Montvale, N.J.: AFIPS Press, pp. 395-410.

[3] Meyer, B.J.F., Brandt, D.M. & Bluth, G.J. (1980). Use of top-level structure in text: key for reading comprehension of ninth grade students. *Reading Research Quarterly*, 1 pp. 72-103. Kieras, D.E. (1980). Initial mention as a signal to thematic content in technical passages. *Memory and Cognition*, 8(4), pp. 345-353. Williams, J.P., Taylor, M.B. & Ganger, S. (1981). Text variations at the level of the individual sentence and the comprehension of simple expository paragraphs. *Journal of Education Psychology*, 73(6), pp. 851-865.

[4] Feiner, S., Nagy, S. & van Dam, A. (1982). An experimental system for creating and reporting interactive graphical documents. *ACM Transactions on Graphics*, 1(1), pp. 59-77.

[5] Trigg, R., Suchman, L. & Halasz, F. (1986). Supporting Collaboration in NoteCards. In *Proceedings of CSCW '86*, Austin, TX: CSCW '86 Conference Committee.

[6] *Thinktank* (1984). Palo Alto, CA: Living Videotext, Inc.

[7] Stelovsky, J. (1984). XS-2: The user interface of an interactive system. Ph.D. Dissertation, Zurich: Swiss Federal Institute of Technology.

[8] Smith, J.B., Weiss, S.F. & Ferguson, G.J. (1986). *MICROARRAS: An Overview*. Technical Report #86-017, Chapel Hill, NC: UNC Department of Computer Science.

[9] Nisbett, R.E. & Wilson, T.D. (1977). Telling more than we can know: Verbal reports on mental processes. Psychological Review, 84, pp. 231-259. Ericsson, K.A. & Simon, A.S. (1980). Verbal reports as data. *Psychology Review*, 83(3), pp. 215-251.

[10] Card, S., Moran, T., & Newell, A. (1983). *The Psychology of Human-Computer Interaction*. Hillsdale, NJ: Erlbaum Associates.