Ellis, J., & Williams, M. (1991). *Retrospective and prospective remembering: Common and distinct processes.* Unpublished manuscript, University of Wales, College of Cardiff.

Friedman, W. (1990). *About time.* Cambridge, MA: MIT Press.

Furnas, G. W. (1986). Generalized fisheye views. *Proceedings of the CHI '86 Conference on Human Factors in Computing Systems,* 16-23. New York: ACM.

Green, T. R. G. (1989). Cognitive dimensions of notations. In A. Sutcliffe & L. Macauley (Eds.), *People and computers V* (pp. 443-460). Cambridge, England: Cambridge University Press.

Greif, J. (1984). The user interface of a personal calendar program. In Y. Vassilou (Ed.), *Human factors and interactive computer systems* (pp. 207-222). Norwood, NJ: Ablex.

Hitch, G., & Ferguson, J. (1991). Prospective memory for future intentions: Some comparisons with memory for past events. *European Journal of Cognitive Psychology, 3,* 285-295.

Kelley, J. F., & Chapanis, A. (1982). How professional persons keep their calendars: Implications for computerization. *Journal of Occupational Psychology, 55,* 241-256.

Kvavilashvili, L. (1987). Remembering intentions as a distinct form of memory. *British Journal of Psychology, 78,* 507-518.

Loftus, E. (1971). Memory for intentions. *Psychonomic Science, 23,* 315-316.

Maylor, E. A. (1990). Age and prospective memory. *Quarterly Journal of Experimental Psychology, 3,* 471-494.

Meacham, J. A., & Colombo, J. A. (1980). External retrieval cues facilitate prospective remembering in children. *Journal of Educational Research, 73,* 299-301.

Meacham, J. A., & Leiman, B. (1982). Remembering to perform future actions. In U. Neisser (Ed.), *Memory observed* (pp. 327-336). San Francisco: Freeman.

Moran, T. P. (1983). Getting into a system: External-internal task mapping analysis. *Proceedings of the CHI '83 Conference on Human Factors in Computing Systems,* 45-49. New York: ACM.

Neisser, U. (1986). Nested structure in autobiographical memory. In D. C. Rubin (Ed.), *Autobiographical memory* (pp. 71-81). Cambridge, England: Cambridge University Press.

Norman, D. A. (1988). *The psychology of everyday things.* New York: Basic Books.

Norman, D. A. (1991). Cognitive artifacts. In J. M. Carroll (Ed.), *Designing interaction* (pp. 17-38). New York: Cambridge University Press.

O'Neill, W. M. (1975). *Time and the calendars.* Sydney: Sydney University Press.

Payne, S. J., & Green, T. R. G. (1986). Task-action grammars: A model of the mental representation of task languages. *Human-Computer Interaction, 2,* 93-133.

Payne, S. J., Squibb, H. R., & Howes, A. (1990). The nature of device models: The yoked state space hypothesis and some experiments with text editors. *Human-Computer Interaction, 5,* 415-444.

Tulving, E. (1972). Episodic and semantic memory. In E. Tulving & W. Donaldson (Eds.), *Organization and memory* (pp. 381-404). New York: Academic.

Zerubavel, E. (1981). *Hidden rhythms: Schedules and calendars in social life.* Chicago: University of Chicago Press.

HCI Editorial Record. First manuscript received November 8, 1991. Revision received October 8, 1992. Accepted by Clayton Lewis.

HUMAN-COMPUTER INTERACTION, 1993, Volume 8, pp. 101-145

Automated Protocol Analysis

John B. Smith, Dana Kay Smith, and Eileen Kupstas
The University of North Carolina

ABSTRACT

Over the past 8 years, The TextLab Research Group within the Department of Computer Science at the University of North Carolina has developed a collection of tools and techniques for recording users' interactions with graphics-based direct manipulation computer systems in machine-readable form and for automatically analyzing and displaying those data. This article describes these tools, discusses their methodological context, and considers their implications for software design and studies of human-computer interaction.

Tools discussed include the following: *tracking* users' behaviors and producing a machine-recorded protocol at the level of users' actions, *replaying* users' sessions from the protocol data, modeling users' strategies using formal *cognitive grammars,* analyzing user sessions by *parsing* them with the grammars, and *displaying* results in visual form—both static and animated—to facilitate interpretation and understanding by researchers. These tools are placed in a methodological context by reviewing issues associated with concurrent think-aloud, keystroke, X-Windows, and video protocols; other support systems for working with these forms of protocol data are also reviewed. The discussion concludes with our reflections on the methodology and its application to computer systems and research objectives different from our own.

Authors' present addresses: John B. Smith, Dana Kay Smith, and Eileen Kupstas, Department of Computer Science, The University of North Carolina, Chapel Hill, NC 27599-3175. E-mail: {jbs,smithdk,kupstas}@cs.unc.edu.

CONTENTS

1. INTRODUCTION

This discussion focuses on tools and techniques for working with machine-recorded protocols that represent users' interactions with graphics-based direct manipulation computer systems. Analysis of these data can show users' cognitive strategies and behaviors over periods of time ranging from a few seconds to several hours, or longer. Whereas this methodology can be used to refine user interfaces and test specific hypotheses, its strength lies in supporting naturalistic and longitudinal studies of individuals using systems for actual work, including use in the individual's normal working environment.

These tools and techniques were developed to support a particular program of research concerned with open-ended tasks, such as planning and writing technical documents or designing and building computer systems; however, the methodology they illustrate is general. Researchers wishing to examine other issues concerned with planning tasks could use our systems and tools directly in experiments of their own design. For other tasks and other research goals where our tools are not appropriate, analogous tools could be developed. We hope that the description of our work will make this job easier.

The gist of our approach is that a user works with an application system that has been modified to produce a machine-readable transcript of all *actions*, rather than keystrokes, performed by that user during the session. That session can take place in the laboratory or in the user's natural working environment. Once recorded, protocol data can be used to re-create, or *replay*, an approximation of the original session, but in a fraction of the original time. They can also be analyzed automatically using a *cognitive grammar* that constitutes a model of users' cognitive strategies for a given task using a particular computer system. The grammar, implemented as a parsing program, produces a *parse tree* that is a concrete representation of a particular user's strategy for a given session. Although parse trees can be examined directly, more often they are further processed by *filter programs* that count various symbols or combinations of symbols in accord with a particular analytic perspective; these derived data are then passed to a statistical utility for conventional analysis. Finally, these various data are presented to the researcher through a combination of *static* and *animated display tools* to facilitate visualization and interpretation.

These tools make possible a methodology that would be impractical otherwise. For example, one member of our research group has recently completed a study that included over 100 sessions by 29 subjects working on actual tasks carried out in their own offices around the world (J. Q. Walker, 1991). In addition to supporting large sample studies in naturalistic settings, these tools can also be used to study longitudinal effects, such as the effects of system expertise on strategy, patterns of user adaptation to new tools, effects of training on user behavior, and so on. They can also be used to address directly theoretical issues—such as the effects of thinking aloud on a user's behavior and the completeness and accuracy of these data—that have been addressed only indirectly using conventional approaches.

One result of this work is the growing sense that methodology in human–computer studies is a field of research in its own right. One should not, of course, lose sight of larger goals, such as building better systems or better theories. But, as we discuss later, our work in methodology has led us toward a new cognitive framework in which to describe users' computer-mediated behaviors, new principles for designing systems, as well as new ways of studying users' strategies. We cannot claim to have achieved it, but we can foresee the possibility that quantitative differences in the time and effort required for carrying out human–computer interaction (HCI) studies could lead to qualitative changes in the field. We find this possibility very exciting. In the discussion that follows, we hope to convey to the reader some of this same sense of opportunity and excitement.

We begin by reviewing related work with respect to both other forms of protocol data and other computer systems that have been developed for working with these different forms of data. The majority of the article,

however, is a discussion of the tools and techniques developed by our group for automated collection and analysis of machine-recorded protocols. The final section includes our reflections on what we have learned over the past 8 years regarding this methodology, including its limitations as well as its possibilities, and our plans for the future.

2. RELATED RESEARCH

Because human thought processes cannot be observed directly, researchers studying HCI have developed several approaches for observing these processes indirectly. The data observed strongly influence what the researcher can infer about the user's cognitive behavior. Consequently, choosing the right kind of data—at the right level of granularity and with the right descriptive parameters—is one of the most important decisions a researcher makes. In the discussion that follows, we briefly review several different kinds of data that have been used for studying HCI; in doing so, we point out some of the benefits and limitations of each and place our own approach, based on users' actions, in context. We also review computer tools that have been developed for working with these different types of protocol data.

2.1. Think-Aloud Protocols

Concurrent think-aloud protocols were developed by Newell, Simon, and others at Carnegie Mellon University during the 1960s to study complex, problem-solving behaviors (Newell & Simon, 1972). The goal of this method is to produce a written record of subjects' trains of thought based on the subjects' own verbalizations of their thinking as it occurs while they perform the task being investigated. Tasks that have been studied using this method include writing documents and computer programs, solving arithmetic problems, assembling physical devices, playing chess, and, more recently, using various computer systems. Under laboratory conditions, subjects are prompted by the experimenter to continuously narrate their thoughts; however, under naturalistic conditions, such as a subject's writing a paper at home, such prompting is impractical. Consequently, think-aloud protocols often differ significantly in the levels of detail reported by different subjects under different conditions.

Think-aloud protocols have been questioned on several other grounds, in addition to varying levels of detail. Nisbett and Wilson (1977) raised three kinds of questions. First, subjects may have incomplete knowledge of their thinking processes; for example, experts frequently have difficulty explaining how they solve complex technical problems. Second, subjects may not have an accurate understanding of those processes of which they are aware. Perhaps

most serious of all, however, is the possibility that the act of thinking aloud, itself, may distort the subject's behavior: The way a subject carries out a task while thinking aloud may not be the same as when the subject does not think aloud. This last objection calls into question the basic design of experiments that rely on think-aloud protocols.

Ericsson and Simon (1980, 1984) responded to these questions by reviewing a large volume of prior research, including some of their own studies. The argument they constructed from these materials asserted that concurrent verbal protocols do, in fact, constitute valid data for most tasks. They base their position on distinctions they make among three types of tasks, which they call Level 1, Level 2, and Level 3 conditions. Level 1 tasks are those in which verbalization of concepts is an inherent part of the task and the verbalization is successively stored in short-term memory in verbal form throughout the task. Under these conditions, they found no evidence that thinking aloud affects this type of cognitive processing or that resulting protocol data are incomplete or distorted. Level 2 tasks are those in which verbalization may be part of the cognitive process but would not normally be heeded as part of that task. Level 3 tasks are those in which verbalization is not part of the cognitive process used to perform the task and, hence, must be generated for the think-aloud process itself. For both Level 2 and Level 3 conditions, Ericsson and Simon concluded that think-aloud protocols can significantly change the cognitive process, especially for tasks that involve recognizing complex patterns and relationships presented visually and for abstract conceptualization (Ericsson & Simon, 1980, 1984).

For researchers concerned with computer systems in which users represent abstract ideas visually and work with them through direct manipulation of associated icons, Level 2 and Level 3 conditions strongly apply. Thus, using think-aloud protocols to study these users' cognitive behaviors should be expected, from a theoretical perspective, to result in distortions of the data, under at least some conditions. This conclusion does not argue definitively against their use of HCI studies, but it does suggest caution.

Think-aloud protocols also present problems of consistent interpretation among human judges who code the raw data. Although training can increase reliability and consistency among judges, encoding remains a subjective process that is prone to differences that can be statistically significant (Hayes & Flower, 1980). A further problem is costs, in both time and money, incurred in transcribing verbal protocols; an hour of think-aloud data typically results in 15 to 20 pages of transcription and requires 8 hr or more to be produced by a trained typist (Hayes & Flower, 1980; Swarts, Flower, & Hayes, 1984).

Thus, thinking aloud is a technique that must be used selectively for theoretical reasons, while the costs and effort involved may require, for many studies, that it be used conservatively. Nevertheless, concurrent think-aloud

protocols offer a source of rich and finely nuanced data that are unavailable by other means. Later, we suggest an approach, in which think-aloud and machine-recorded methods can be used in conjunction with one another, that can alleviate many of these problems.

Tools to assist analysts working with verbal protocols appeared almost as soon as the methodology. The first such system, PAS-I, provided a set of powerful tools that included natural language parsing of verbal protocols and generation of graphs depicting sequences of subjects' mental actions (Waterman & Newell, 1971). Whereas PAS-I was limited to working only with cryptarithmetic protocols, PAS-II was a more general tool that also allowed input from the researcher during the parse and interpretation of the data (Waterman & Newell, 1973). The trend toward generality has continued. Today, the predominant form of tool is one that manages the codings and analyses for the researcher but assumes input from the human user for most, if not all, semantic information. These systems include VPA (Lueke, Pagery, & Brown, 1987), PAW (Fisher, 1988), and SHAPA (Sanderson, James, & Seidler, 1989). VPA, a system developed within IBM, is intended primarily for testing system usability; it permits coding of protocols in terms of a hierarchy of categories and key words defined by the user. PAW is intended primarily for research in programming skills. SHAPA is a general purpose tool that facilitates defining a structure for the protocol encoding vocabulary in terms of predicates and arguments, coding the protocols in terms of that structure, and collecting and aggregating data.

2.2. Events Protocols

To address problems of cognitive interference and the laboriousness of data preparation associated with verbal protocols, some researchers have used the computer system, itself, to collect protocol data for HCI studies. For command-driven systems, data are collected in the form of keystrokes. For some more recent graphics-based direct manipulation systems, protocols can be collected for a wider variety of input and display events. X-Windows protocols are an example of this type of data, but comparable data are also available for Macintosh applications and programs running under other operating systems and window management systems, as well. We refer to both keystroke and direct manipulation protocols as *events protocols,* but we discuss the two separately because they have arisen in different contexts.

Keystrokes

The keystroke protocol is based on data comprised of each key pressed by a subject while working with a computer system (Card, Moran, & Newell, 1983). This approach is attractive for several reasons. Protocols may be recorded passively, unlike think-aloud methods. They can often be recorded by the operating system or by an analytic shell, in which case the application program can be used without modification. However, keystroke data also raise several problems. First, they are very fine grained; thus, storing, managing, and interpreting these data are formidable tasks. Second, to infer what the keystrokes add up to in terms of the commands and functions supported by the application program requires an analytic program equivalent in power to the application program's own internal command interpreter. Third, if interpreting a particular user action is dependent on the current state of data in the application, the analytic program would also have to duplicate much of the application program's function and data structures. Although keystroke data solve the problem of distorting task behavior posed by think-aloud protocols, they share with that method problems of incompleteness and of producing large volumes of fine-grained data that must be analyzed or coded before being used.

Direct Manipulation Events

Since the original presentation of the GOMS model, computer systems have changed substantially. Today, users expect to control their computers through menu selections and direct manipulation of visual icons, rather than keyed commands. Many of the operating systems and/or window management systems that support user application programs manage input/output events through low-level protocols. X-windows protocols are an example of this form of data. Originally intended to ease software portability among multiple hardware and software platforms, the X-Windows system produces a flow of coded events between the application and the input and output devices being used, including, but not restricted to, keyboard, mouse, and display. X and other comparable protocols are extremely fine grained, denoting each movement of the mouse, each character typed or displayed, each line drawn, and so on. Consequently, although such data can be used to re-create a semblance of the display as it was presented to a user during a recorded session, interpreting these data presents the same problems outlined before for keystrokes, particularly inferring the effects of a user action on data being manipulated by the user. In addition to problems of interpretation, storing X and other low-level protocols can be problematic for studies that require long user sessions and/or large numbers of subjects. We have found in these experiments that storing an X protocol for a 1-hr session can produce as much as 3 million bytes of data. We are optimistic that compression techniques can reduce this amount by 80% to 90%, but, even so, storage currently remains an issue for large studies.

2.3. Video Protocols

Video protocols have been used alone as well as to supplement both think-aloud and keystroke protocols. For HCI studies, subjects are videotaped while they perform a task using a computer. These data can show what a person is doing when not thinking aloud or when not typing on the computer keyboard. They can also show what is displayed on the computer screen—information that is not available through think-aloud or keystroke records. A major benefit to collecting this type of data is that users' behaviors are captured in a complete context (Mackay, Guindon, Mantei, Suchman, & Tatar, 1988). Although video protocols provide a rich source of data, they also require extensive analysis and coding, raising the same issues of consistency, time, and labor discussed previously.

Some of the tools for verbal protocols can also work with video data. Here, we describe several tools for working directly with the video medium. This class of tool is still in the exploratory or developmental stages, but several examples reported in the literature illustrate the concept and are beginning to be used with actual data. The Experimental Video Annotator (EVA) allows the user to view a video protocol and record "time stamps" at points of interest as well as mark segments using previously established codes. A text editor also allows the user to record comments linked to time-stamped sequences. However, it was built as a prototype and requires an Athena workstation and special video hardware (Mackay, 1989). Trigg (1989) described a similar tool, but with the addition of hypertext links and anchors that permit researchers to associate and to view in close succession different sequences on multiple videotapes. A more advanced version of this tool is called VideoNoter (Roschelle, Pea, & Trigg, 1990). VideoNoter is a hypermedia system that allows the analyst to link segments of video to textual and graphical annotations. The resulting relations can be sorted, searched, modified, and so on, according to any analytic perspective. Suchman and Trigg (1991) discussed the use of this system in studying videotapes of a group of airline employees working in an airport operations room. Researchers at Texas Instruments have described a system that produces a video as well as a machine-recorded events protocol for Macintosh and PC-compatible applications; it permits researchers to index and retrieve video segments from a session as well as perform limited analyses of the events data (Hammontree, Hendrickson, & Hensley, 1992).

2.4. Action Protocols

Action protocols record data that are approximately an order of magnitude more general than keystroke, X, and other forms of events protocols. Sensors are embedded directly into the program to record data such as movements of the mouse from one window on the screen to another, selection of data objects and menu options, character strings typed in as names or labels for objects, and other similar information. Consequently, the approach is currently restricted to studies that involve computer systems for which the researcher has access to the source code or can persuade those who do to make the necessary modifications.

Action-level protocols address many of the problems observed for other forms of protocol data. They solve the problem of cognitive interference raised by think-aloud protocols by being passive and unobtrusive. Because they are recorded in machine-readable form, they also eliminate the need for manual transcription. Although these features are shared with keystroke, X, and other events protocols, they also address some of the practical problems associated with these latter forms by recording an order of magnitude fewer events. Problems of analysis and interpretation remain, but they are lessened because atomic units at the action level are produced by sequences of keystrokes or mouse events that have already been interpreted by the application program's own command processor. For example, the protocol can record the specific data object selected, not just its location or representation on the display. It is this aspect of action protocols that allows automated analysis through grammatical parsing, which we describe later. Although action protocols are not a panacea, these advantages may suggest their use as a supplement or alternative to other forms of data for a range of HCI studies.

Action-level protocols have been used in many different HCI contexts. J. H. Walker, E. Young, and Mannes (1989) studied methods for finding information in an on-line documentation system. Using action protocols, they were able to isolate types of commands and command sequences as they were used in natural, everyday activities. Siochi and Ehrich (1991) used action-level protocols in conjunction with a tool that filters out repeated sets of actions to evaluate interface designs. Finally, our project has made extensive use of action protocols and, thus, can be viewed in the context of the other work described here. Additional details for action protocols can be found in the upcoming Tracking section.

2.5. Group Protocols

Although most of our discussion of protocol methodology that follows is restricted to tools and techniques for working with data from individual subjects, to complete this survey we note, in passing, recent work with respect to protocol data for groups of users. Several projects have developed tools for studying groups from the points of view of cooperative work and/or collaboration. Most of these tools are currently oriented toward managing,

analyzing, and displaying results based on data that are hand coded from concurrent observations by a trained researcher or from subsequent analyses of video recordings. One such system is GroupAnalyzer, developed at the Capture Lab in the Center for Machine Intelligence in Ann Arbor, Michigan (Losada, Sanchez, & Noble, 1990). Using this system, trained observers code the behavior of each individual participating in a meeting according to a pre-established classification scheme, such as SYMLOG (Bales & Cohen, 1979). GroupAnalyzer manages the coded data, including time stamps for individual behaviors, and provides an interface for external programs that perform time-series and other analyses on the data. A similar tool, but with more extensive capabilities for displaying behaviors with respect to artifacts, is described in G. M. Olson and J. S. Olson (1991).

Our group is currently developing a set of tools for working with protocols from collaborative groups that combine machine-recorded protocols from multiple simultaneous users with hand-coded protocols. We are attempting to capture the varied activities of a group that, at any one moment, may find some members working individually at their respective workstations (and, hence, gathering multiple machine-recorded protocols) while others attend a meeting (which could result in a hand-coded protocol from a trained observer of the meeting). Some of our preliminary efforts in this direction are described in Blakeley (1990), J. B. Smith et al. (1990), and J. B. Smith (1992). As more computer systems are developed to support cooperative and collaborative work, additional tools will be needed for working with machine-recorded group protocols.

3. PROTOCOL TOOLS

In this section, we describe tools and techniques we have developed for working with machine-recorded action protocols, for automatically analyzing those data using parsing programs that implement cognitive grammars, and, finally, for displaying results in ways that help researchers visualize and interpret patterns and relationships in the data—hence, our focus on methodology. It is important to recognize that methodology is normally treated as orthogonal to most research concerns. In most HCI studies, one addresses a particular issue or hypothesis, using a particular group of subjects who perform a given task, using a particular computer system. The methodology used also provides the observational context that determines the forms of data that can be collected and, eventually, analyzed by the researcher. Like mortar between stones, methodology provides the matrix that holds the different components of the research design together. Normally, it remains in the background, if not invisible, in discussions of HCI research. Consequently, when we focus directly on methodology, we must be careful to distinguish between what is mortar and what is stone.

We developed the tools and techniques described here in order to consider a specific set of research issues using one of several computer systems also developed by our group—the "stones" in our research designs. Examples included in the discussion are drawn from this work. To make clear the distinction between what is general and what is specific with respect to our particular project, we begin by describing the motivation, assumptions, and systems from which this methodology grew. After that, when we discuss individual tools, we point out how they apply to the specifics of our project and how they apply or could be extended to other projects.

3.1. Background

For the past 8 years, our work has centered on building *intelligence-amplifying* (IA) computer systems to help people perform complex, open-ended tasks, such as writing technical and scientific documents, planning and building computer systems, and other similar conceptual development tasks. To build such systems, we must also build an understanding of users' conceptual processes and the strategies they use to carry out the task being supported, because, ideally, the organization and function of an IA system will be consistent with the user's cognitive processes and strategies.

More specifically, if one attempts to build a system that will be resonant with the ways its users think, one must develop a model not just of the task in question but of the way users will accomplish that task while using the system. Such systems are inherently theory based. To evaluate the effectiveness of the system and to validate the model of users' computer-mediated behaviors, one must also conduct empirical studies of users working with the system. Descriptions of our work in theory-based design can be found in J. B. Smith and Lansman (1989, 1992) and in J. B. Smith, Weiss, and Ferguson (1987). Descriptions of empirical studies can be found in Blakeley (1990), Lansman, J. B. Smith, and Weber (1990), and J. Q. Walker (1991).

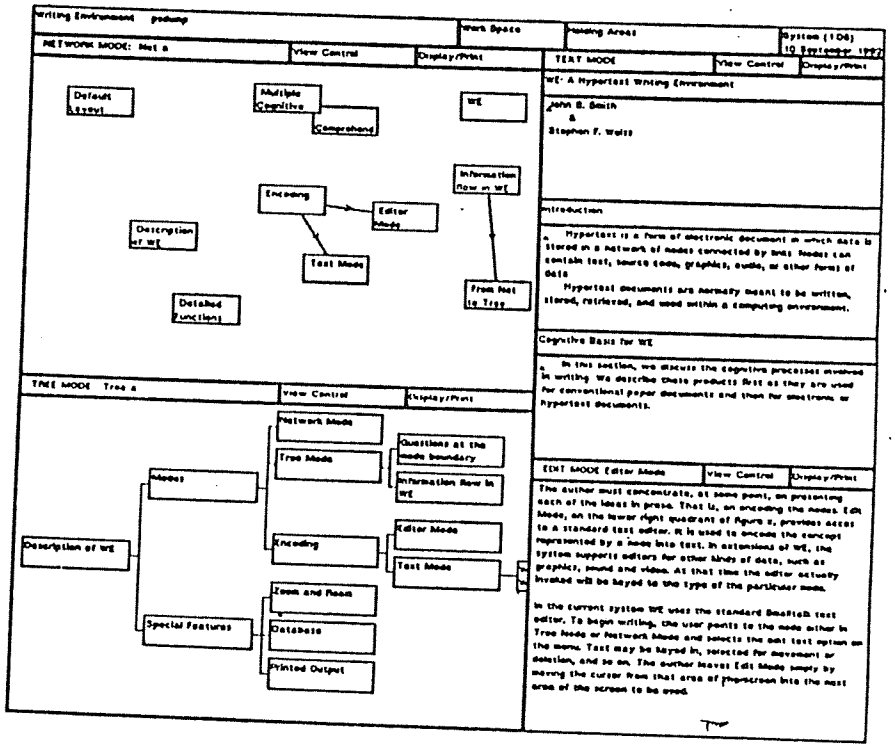
During this same period, our group has built three different systems, which are referred to in the discussion that follows. The first of these and the one we refer to most often is a hypertext-based Writing Environment (WE) intended to assist professionals who write technical and scientific documents as part of their work (J. B. Smith, Weiss, Ferguson, Botler, et al., 1987). We occasionally refer to a second writing support system, called PROSE II, which was developed by a graduate student to observe users in their normal working environments and to test a different model of cognitive strategy (J. Q. Walker, 1991). A third system, called the Artifact-Based Collaboration (ABC) System, which is still being developed, will be used to assist and observe collaborative groups carrying out a number of different design and conceptual construction tasks, including software development (J. B. Smith &

F. D. Smith, 1991; J. B. Smith et al., 1990). Because ABC includes automated protocol tools that are extensions of the single user tools described here, we occasionally comment on it as indicative of future possibilities, but it is not discussed in detail.

The WE system was built according to a particular cognitive theory of writing (J. B. Smith & Lansman, 1989, 1992). Briefly, we view writing (as well as other knowledge construction tasks) as requiring a number of different types of thinking, which we call *cognitive modes*. Each mode is defined by its particular configuration of *goals*, which are achieved by producing particular types of cognitive *products*, developed using particular cognitive processes, in accord with a particular set of *constraints*. Seven specific cognitive modes were included in the model of writing we developed and used as the basis for WE. These included exploring concepts, exploring the rhetorical situation, organizing, drafting, and three forms of editing—verifying and revising structure, coherence relationships, and linguistic expression. Consequently, we viewed the overall writing process as a succession of cognitive modes engaged by the writer in accord with an overall strategy as well as in response to specific problems or conditions. Thus, a particular writer's strategy can be seen in the patterns that exist in the sequence of modes that person uses. These include global and more localized iterative and recursive patterns.

The WE system includes four system modes that corresponded with six of these seven cognitive modes. The default configuration is shown in Figure 1. Each mode is implemented in a separate window, each supports a different data model, and each provides functions appropriate for working with data in that particular form. In the upper left is a window, called *network* mode, intended to support exploratory thinking, including brainstorming, clustering of concepts, and denoting various associative and semantic relationships. Concepts are represented as *nodes*—electronic equivalents of "post-its"—that can easily be moved around the display to form clusters of related concepts. More explicit relationships are denoted by links. Organizational thinking is supported in a *tree* mode, shown in the lower left window; it is constrained to support only trees or hierarchical structures. Component structures can, of course, be easily moved back and forth between network and tree modes. The text for the document is written by opening a text *editor*, shown in the lower right quadrant, on the contents of any node included in either the network or tree structures. The user writes the document as a whole by writing text for all of the nodes in the tree. A printed version of the document is constructed by the system by going around the tree, collecting individual content segments, and passing the concatenated sequence to a printer. Consequently, structural editing is accomplished by simply reorganizing the nodes in the tree. Finally, the window in the upper right corner, called *text* mode, is used to view and edit the contents of adjacent nodes. Thus, writers can edit transitions between paragraphs and sections, move text from one node to

Figure 1. **The default layout of windows in the Writing Environment system, showing its four system modes: network, tree, editor, and text modes. Each system mode supports a different data model appropriate for one or more corresponding cognitive modes.**



another, and perform other similar forms of coherence editing. In summary, network mode supports conceptual exploration, tree mode supports organization and structural editing modes, edit mode supports both drafting and linguistic editing, and text mode supports coherence editing. The cognitive mode not included in the system design is rhetorical exploration, which could be included in a future version of the WE system.

The research issues that concerned us dealt with writers' cognitive strategies and tactics. In particular, we wanted to see whether the strategies of expert writers differed from those of novices, whether strategy made a difference with respect to document quality, whether writers' domain knowledge would affect their writing strategies, and other similar issues. Thus, we were interested in the order in which writers engaged different modes, the different types of conceptual products they developed in each, and the particular operations they performed on these products as they gradually transformed

inchoate ideas into coherent structures and, eventually, into well-formed prose. To address these issues required rather extensive experimental designs. A typical experiment lasted 3 to 4 half days in the laboratory, and we also conducted actual-use studies, with the PROSE II system, in users' normal working environments.

The "stones" of our research agenda, then, included the cognitive mode framework, the various multimodal hypertext systems, and a set of research questions that focused on users' large-grain strategic behaviors as well as finer grained tactics and processes. The methodology we describe, however, is sufficiently general so that it can support other stones. One does not have to buy into our assumptions, models, systems, or interests to use these or similar tools to pursue entirely different research objectives, based on different assumptions and perspectives. In the remainder of this discussion, we illustrate these tools with examples from our work, and we point out more general uses of them as well.

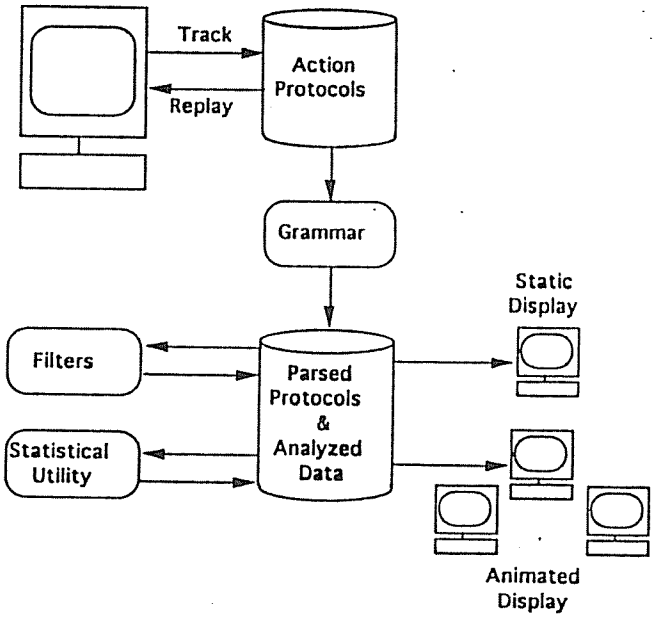
3.2. Individual Tools

In this section, we describe the different tools and techniques we have developed. Existing tools were written in Smalltalk, Version 2.5, under the UNIX operating system and run on platforms that support this environment; new tools are being written in C++. Our discussion is organized in terms of the flow of protocol data, shown in Figure 2, as it is recorded, analyzed, and displayed. A user works with an application system, such as the WE system, that has been modified to track user actions and produce a machine-readable transcript, called an action-level protocol. The recorded protocol can then be used to replay an approximation of the original session but in a fraction of the original time. These protocols can also be analyzed automatically by a cognitive grammar. Because the grammars are implemented as parsing programs, they produce a parse tree that represents a particular user's strategy for a given session. Parse trees can be examined directly, but they are often processed by filter programs that count various symbols or combinations of symbols; these derived data are then passed to a statistical utility for conventional analysis. Finally, these data are presented to the researcher through a combination of static and animated display tools to facilitate visualization and interpretation. In the sections that follow, we discuss each step of this overall analytic process in detail.

Tracking

The protocol data we work with are records of users' actions. Conceptually, an action is normally an intentional gesture performed by a user to create or alter a data object or to alter the display or state of the application system.

Figure 2. **Flow of protocol data through tools. Action-level protocols are recorded by the system and then used to replay a session. After being parsed by a grammar and, perhaps, processed by a filter, analyzed protocol data can be displayed in static or animated forms.**



Typically, actions fall into three categories: selecting a particular window (system mode), selecting a data object (e.g., a node or link in the WE system), or selecting a menu item and, thus, executing a command.

Actions have two primary advantages over lower level events, such as keystrokes or X protocols. First, they result in an order of magnitude less data but still provide detailed accounts of users' interactions with a system; this significantly reduces storage costs. More important, they take advantage of the application program's command interpreter and data structures. Thus, one can directly determine the particular command executed, rather than having to parse a sequence of characters or the spatial coordinates of a menu item to determine which command was designed. Issues with respect to data structures are more subtle. When a concept is represented in the display by a visual icon—such as a node—the data structure will identify that icon as a unique object. When it is moved, its appearance or name changed, and so forth, the data structure will still "know" that it is the same object. At the level of X protocols, however, identity is lost. The system only knows at this level that the object is a box located at a particular screen position. Thus, continuity is lost when the location and/or appearance of the object change.

At best, maintaining contextual data within an analytic program would be extremely complex and laborious, and it may be impossible for sessions that begin with previous data. The principal disadvantage to action protocols is that the researcher must have access to the source code or persuade those who do to insert the required sensors into the code.

Individual actions are associated with various parameters. For example, associated with a node selected in the display are the identity of the particular node and its location—For the WE system, the latter is its *x,y* coordinates in network mode or its position in the hierarchy in tree mode. Other actions have other appropriate parameters associated with them. Each action is also associated with two time values: the time at which the action is started and the time at which it is completed. (These time values are with respect to a session clock that begins at zero when a session is started.) Typically, these values differ by only the few seconds required by users to perform these very basic operations.

Each action results in a record, or line, in the machine-produced protocol for the session. Each such record includes the name of the action, the start and stop times for the action, and relevant parameters in the form of attributes and values. These data, recorded automatically by sensors inserted into an application program, are formatted in accord with the rules of a *protocol description language* that is general across all of our tools and then output to secondary storage. Each protocol file begins with a header that includes the user's name, the version of the application program being used, the date the manuscript was created, the time of day at which the session began, and other information relevant to the tools themselves. These data constitute an action-level transcript.

Finding the right way to calculate the time attribute(s) for user actions has been difficult. The main problem has been determining the duration of an event. In earlier versions of the protocol language, we assumed that the duration of an action was the amount of time extending from the beginning of one action to the beginning of the next action. Consequently, we recorded only the time at which the action started for each record in the protocol. The duration for the action was then calculated by subtracting the time values on successive lines. This procedure proved to be inaccurate; users often perform mental actions between events that are not part of the preceding action (e.g., planning, pondering, or daydreaming). We have since corrected this mistake by including a code in our sensors to note when each computer action is completed as well as when it started. The particulars of this determination, of course, are specific to the particular action. Thus, if one examines successive lines of this revised action protocol transcript, one might observe one action beginning at 0:02:33 (read 2 min, 33 sec into the session) and ending at 0:02:39, indicating that the action took 6 sec to perform. If the time

parameters for the next action read 0:02:54 and 0:03:03, one could then deduce that a pause of some 15 sec occurred between the two actions.

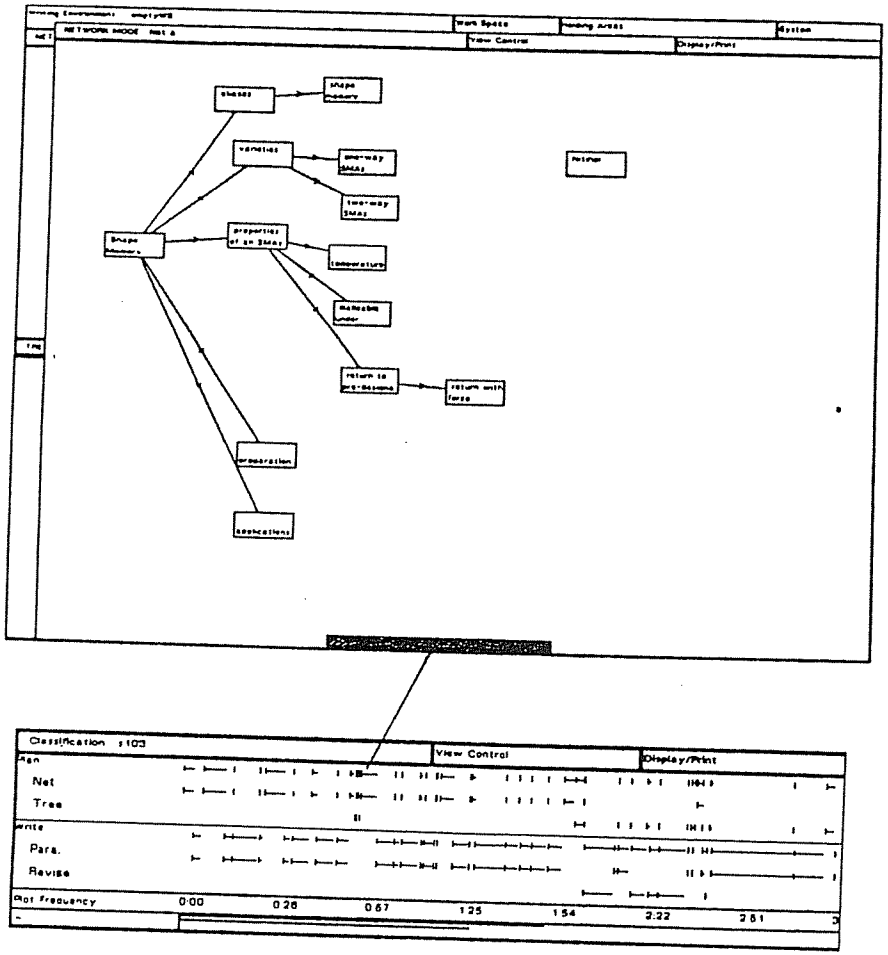
Pause information can, of course, be both interesting and important for a variety of research objectives. For example, J. Q. Walker (1991) found that users working in their natural work environments spend only about half the time actually performing operations on their workstations, whereas the rest of their time was spent in pauses of various durations. Durations, in turn, strongly differentiated among a hierarchy of activities. For example, individual actions were typically separated by pauses of approximately 6 sec, supporting the intercommand cutoff of 5 sec used by Card et al. (1983). Users tended to work in rather continuous episodes of several minutes before pausing for longer periods, suggesting that during these intervals they were either evaluating what they had just done or planning their next episode rather than simply executing an action sequence. Episodes tended to be grouped into phases of similar activity that lasted from 5 to 15 min, perhaps lending support to the cognitive mode concept described before. Thus, pauses can provide insights into users' strategies and patterns of behavior that complement insights gained by observing users' actions.

We are currently extending our tracking functions and protocol transcription formats to study how multiple users interact with one another in developing large, complex structures of ideas over periods lasting from months to years. Action protocols may be appropriate for analyzing particular periods of work, but they may be impractical for extended studies because of the volume of data produced. We are exploring the use of a larger grain record that records the sequence of modes engaged by one or more users in order to read or work on a particular artifact. Thus, one could record mode/artifact data for an entire collaborative project and action-level protocols only for selected periods of work. To provide for this option, we are considering making the granularity of the protocol events that will be recorded by future systems a variable that can be changed dynamically.

Replay

Replay is the inverse function of tracking. Our systems were designed to read a stored protocol and re-create an approximation of the original session. During replay, the screen appears exactly as it appeared to the user during the session in which the protocol was recorded; however, none of the program's controls is active. Each user action is replicated from the beginning of the session, so the viewer can watch the user's strategy unfold. A sample replay screen for the WE system is shown in Figure 3. In this session, the user has resized the network mode to cover almost the entire screen. Along the bottom of the replay screen is a box showing the current action and its time relative to the beginning of the original session—in this case, some 58 min into the

Figure 3. **Example of a replay screen for the Writing Environment system, showing the user's view of the session some 58 min after it began. Below the replay screen is a distribution of all actions for the session, classified as either *planning* or *writing*, with the current action marked.**



session. Below the replay screen is a second display in which each action has been classified as either a planning or writing action. The current action in the replay has been marked to show its context within the overall sequence of actions, so classified. We discuss these and other visualization tools later and in more detail.

The replay function offers the viewer several control options. By default, the replay starts at the beginning of the session and replays continuously until

all actions have been shown and the session is completed. However, the viewer can stop and start the replay, manually step through the actions, and control the pace of continuous replay. The manual option permits the researcher to step through the session for one or for a designated number of actions per mouse click. The continuous option permits the user to control the speed of the replay in several ways. The interval between events can be set so that it duplicates the pace of the original session including the full duration of all of the pauses between actions. Alternatively, the length of pauses can be set to some proportional value; for example, a value of one tenth will shorten a pause of 50 sec on the original session to only 5 sec in the replay. This option permits the observer to view the session in a fraction of the original time. A third option lets the observer indicate a constant number of seconds to be inserted between actions; this interval can range from 0, in which case the session will replay as fast as the system can reproduce the user's actions, to any specified number of seconds. Although the total time required for replaying a session will vary, generally a 3-hr session can be replayed in 10 to 15 min when pauses are set to zero. At any time during a replay, the viewer may interrupt the process and change from one replay option to another or change the parameter value for a given option. In summary, replay allows the researcher to view a session in manually controlled steps or continuously, with original sessions pauses set to a duration proportional to the original or to some constant interval of time, and to change from one option to another during the replay.

We have been surprised at how useful replay is for analysis. Shortening the time required to view sessions helps the researcher infer overall behavioral patterns and strategies, and the proportional time feature helps one get a comparative sense of the amount of time required for different tasks and operations and for the user's rhythm of work punctuated by pauses. Replay is similar in some respects to video protocols that show the display. However, it does not have the flicker and distortion associated with video, and it gives the viewer much more varied and precise control of the replay. As we discuss later, it can also be coupled with additional analytic displays (as suggested by the abbreviated event/times display shown in Figure 3) to give a composite view of a user's behavior that cannot be done with video.

Presently, replay does not reproduce work carried out within the text editor; as a result, time spent in either the editor or text modes is indicated by a blinking cursor for the appropriate duration, but no actual events are shown. We plan to extend our tracking and replay capabilities in the future so that we may show the user's actions during actual writing.

Recently, we have begun using replay to gather cued retrospective think-aloud protocols as additional data for studies and to validate our grammars, as described later. We collect this form of think-aloud protocol immediately after a working session is completed. The user and the researcher

view a replay of the session together, and the user narrates his or her actions, cued by the replay, for the entire session or for portions of particular interest to the researcher. In this way, the researcher can ask the subject about particular events or episodes without interrupting the user's train of thought during the original session. This approach avoids the potential problem of task distortion associated with concurrent think-aloud methods (e.g., Ericsson & Simon, 1980, 1984; Nisbett & Wilson, 1977) and provides many of the benefits. In a future experiment, we will catalog similarities and differences in the data generated by concurrent think-aloud, cued retrospective think-aloud, and machine-recorded protocols. We hope to address directly an important set of methodological issues that, to the best of our knowledge, has only been debated abstractly in the past.

Grammars and Related Tools

Grammars are at the core of our theoretical and methodological thinking. On the theoretical side, grammars constitute formal models that describe users' cognitive interactions with a computer system for a particular conceptual task. On the methodological side, grammars are used by parsing programs to analyze machine-recorded protocols. This capability to automate analysis makes possible longitudinal and naturalistic studies for large numbers of subjects that would not be practical otherwise.

We view a protocol transcript as analogous to a statement or discourse in natural language. From this perspective, individual user actions can be thought of as words, and sequences of actions as cognitive phrases, sentences, or discourse that depict the user's interactions with the system through which he or she attempts to achieve a hierarchy of goals. More specifically, the actions recorded within the protocol description language are the terminal symbols in the grammar, and higher level cognitive behaviors are the nonterminal symbols implied by sequences of actions. These grammars, then, represent formal descriptive models of users' cognitive behaviors used to carry out a particular task while using a given computer system. Thus, they describe particular forms of *computer mediated-cognition,* as opposed to cognition in general.

Grammars are used to parse sequences of protocol symbols, analogous to parsing sequences of words using a natural language grammar. The resulting parse trees provide concrete representations of a user's cognitive behavior, tactics, and strategies for a given session. These representations enable us to make comparisons between sessions for different users or for the same user under different conditions. This use of grammars contrasts with that of Foley and Wallace (1974), Reisner (1981), and Kieras and Polson (1985), who have built generative grammars to specify valid sequences of user actions but have not applied them to user protocols. In the remainder of this section, we briefly describe three different grammars developed by our group; we comment on the issue of validation common to all such grammars later on in the Reflections section.

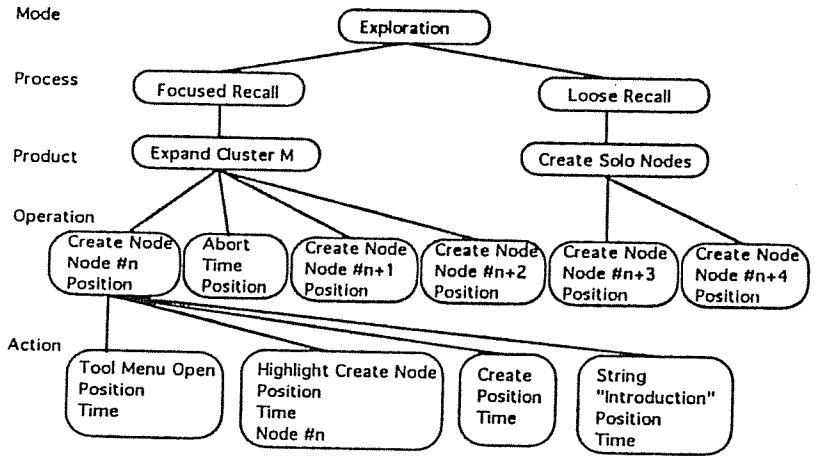
Our first grammar was expressed within the formalism of production rules supplemented by functions that recognize different types of graph structures, such as disconnected sets of nodes, connected graphs, trees, and so on (J. B. Smith, Rooks, & Ferguson, 1989); the recognizer functions made the grammar, effectively, context sensitive. The grammar was implemented using the OPS83 expert system shell. We are nearing completion of a second grammar that is expressed as an augmented transition network (ATN), and we are currently working on an associated general-purpose parser that will work with any grammar expressed as an ATN. To assist with the refinement of this ATN model and with development of future ATN grammars, we are also building a visual, direct-manipulation grammar development tool. Both grammars work with action protocols produced by the WE system. A third grammar was developed for the PROSE II system and implemented directly in C code (J. Q. Walker, 1991).

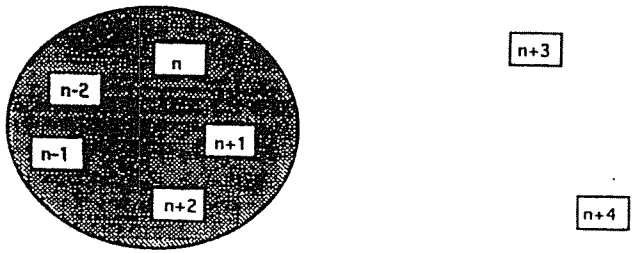
All three grammars model the task of expository writing, including planning, writing, and revising activities, but with emphasis on structural, as opposed to linguistic, aspects of the process. The first two grammars assume an underlying cognitive modes framework for the kinds of thinking required for writing; the third uses a somewhat different set of assumptions based on a hierarchy of typed activities characterized by their durations as well as their configurations of elements.

WE Production Rule Grammar. Our first grammar viewed a *session* as being composed of a series of cognitive *modes* in which a sequence of cognitive *processes* are used to produce particular cognitive *products* or changes to products. Thus, the top four levels of the grammar describe the cognitive behavior of the user and are general with respect to the task, as cast within this particular theoretical framework, described earlier. To transfer changes in cognitive products that take place in the head to the representation of those products supported by the computer system, the user performs a series of system *operations,* each composed of one or a short sequence of system *actions.* Actions, of course, are the protocol data recorded by the tracking function, described earlier, and operations—inferred from the actions—are slightly more general functions that could be implemented in different ways in different systems. Thus, the bottom two levels of the grammar describe the system.

An illustration of the various levels of this grammar and their relation to a sequence of user actions is shown in Figure 4. In this depiction, which starts at the level of mode rather than session, we are looking at a user's actions across time. The figure is focused on the action that resulted in the creation of node n. The user has recently created nodes $n - 1$ and $n - 2$, and in the

Figure 4. An illustration of the different levels in the Writing Environment production rule grammar. The actions that resulted in the creation of node *n* are shown along with the parser's interpretations of them. The figure also illustrates that similar sequences of actions can result in different, context-sensitive interpretations, depending on their parameters—in this case, the locations in the display where the nodes were created.





near future, he or she will create nodes $n + 1$ through $n + 4$. These seven nodes and their approximate spatial locations with respect to one another are shown at the bottom of the figure. Note that five of the nodes are interpreted as forming a *cluster*. At the top is a portion of the parse tree that shows the grammar's interpretation of the user's actions that produced these nodes.

To create node *n* requires several actions, which are shown at the bottom of the parse tree. These include opening a menu, selecting the "create node" option, and typing a name for the new node. These actions comprise the system operation, *create_node,* shown in the second level of the tree. Within this particular context, this operation is interpreted as an indication that Cluster

M has been expanded (Level 3), as a result of a *focused recall* cognitive process (Level 4), occurring within an *exploration* cognitive mode (top level). We can also see that the user's activities that produced nodes $n + 1$ and $n + 2$ are interpreted similarly. However, the activities that resulted in nodes $n + 3$ and $n + 4$, by virtue of the locations on the screen where the nodes were placed, were interpreted differently, indicating that a *loose recall* cognitive process has occurred. Finally, all of the activities involved in creating nodes $n - 2$ through $n + 4$ are interpreted by the grammar as occurring within an instance of the *exploration* cognitive mode.

In the example, we can see that, prior to creating node *n,* the user had defined two earlier nodes (labeled $n - 1$ and $n - 2$) and that he will soon produce two more nodes (labeled $n + 1$ and $n + 2$). All five nodes are viewed as part of a single *cluster* by virtue of their spatial proximity to one another and, thus, are interpreted as instances of *focused recall,* memory accesses of concepts that are closely associated semantically for this user. After forming the cluster, the user then produced two more nodes (labeled $n + 3$ and $n + 4$) that are viewed by the grammar as not being part of the cluster, because they are spatially distant from the nodes in the cluster. The grammar makes this distinction using one of the recognizer functions, described earlier. These two nodes are, thus, interpreted by the grammar as indicating what we term *loose recall,* suggesting that, for this user, they are not closely related semantically to the concepts within the cluster. As can be seen by this example, the same system operation may be interpreted differently by the grammar and supplemented by the recognition functions, depending on factors such as where on the screen a particular operation takes place.

This production rule grammar has been used to analyze some 50 protocols, each representing an approximately 3-hr session recorded under semi-naturalistic conditions in the laboratory. After initial debugging, it has successfully parsed each protocol. Results of these studies have been described in Lansman et al. (1990), Lansman (1991), and Weber (1992).

WE ATN Grammar. The cognitive grammar described before works satisfactorily, but the architecture of the OPS83 program that implements it is awkward, making maintenance difficult. Also, our ideas concerning its model of the writing process have evolved. Consequently, we began work this past year on a second grammar that will define a revised model. It is being developed as a set of ATN graphs.

An ATN is a specialized form of a finite state machine (FSM) that allows tests on transitions, operations on register values, and recursive calls to other ATNs. ATNs were first described in Woods (1970). The main application of ATNs has been in natural language understanding systems. An ATN grammar is expressed as a set of graph structures whose nodes represent states and whose links represent transitions. Descriptive labels that denote the

Document, which generates a call to the second graph, identified as Write Document. When this activity is completed (as well as any additional calls to other lower level graphs), the traversal of the link in the top graph will be completed and the grammar will *stop.* This same pattern of descending and ascending by starting to traverse a link, calling a lower level graph, completing its traversal, and returning to the higher level graph to complete the traversal of that same link is followed throughout the ATN grammar. Let us now follow one path through the grammar down to its terminal symbols.

Within the context of the *Write Document* graph, users can *explore concepts, develop* the *structure* of the document, develop its *expression,* or they can consciously address the *tools.* Assume that our writers decide to *explore concepts.* They then engage a submode in which they may *brainstorm,* build *clusters* of ideas, or *build component structures,* each leading to a lower level graph for that activity. If they decide to *brainstorm,* they can represent a concept (*define* it), *revise* an existing concept, or discard (*delete*) an existing concept from further consideration. Defining a concept is viewed as a basic cognitive process in our model; representing that concept is done by performing an *operation* comprised of a small sequence of *actions* (not shown in the figure). This operation is subjected to one or more *tests;* if the conditions are satisfied, the transition in this lowest level ATN network is completed. The grammar would then "unwind" itself back to the highest level or to an appropriate intermediate level and begin processing the next symbol from the input stream—in this case, an action protocol event.

Other strategic choices would, of course, lead to other behaviors, other protocol sequences, and other paths through the ATN. But we hope this simple example will provide an intuitive sense of the model expressed in the ATN grammar. Because other models could be expressed in other ATN network structures, our strategy is to develop a general parsing program that can be used with different ATN grammars.

PROSE II Episode Grammar. As a dissertation project, J. Q. Walker (1991) developed a different writing support system that he used in a study of users' strategies under actual working conditions. Walker's system, which he called Prose II, was developed for IBM PC-compatible computers and ran as a Microsoft Windows® application under either MS-DOS® or PC-DOS®. To collect data, Walker solicited volunteers from a world-wide corporate computer network and then distributed the Prose II system via the network to interested individuals. Users worked with the system in their offices over several weeks on whatever tasks they happened to be doing. The Prose II system saved protocols as files on the user's hard disk, which were returned to him via the same network. Although Walker's overall approach was similar to that described before for the WE system, his grammar was based on a different underlying cognitive theory, illustrating that the methodology is not limited to task models expressed in terms of cognitive modes.

Walker's grammar was based on a hierarchy of user activities derived, in part, from concepts discussed in Card et al. (1983). Higher level activities, in Walker's grammar, were determined with respect to both sequences of lower level elements as well as pauses of different durations. Walker's grammar, which was of fixed depth four, began at the top with a *session.* Each session was divided into *phases,* which were divided into *episodes,* each composed of a continuous sequence of several *commands.* Commands, of course, applied to the PROSE II system, and episodes, phases, and session described cognitive and/or conceptual construction activities of the user.

In summary, our group has now developed three different grammars for working with machine-recorded protocol data from two different computer systems. This methodology makes it practical to gather and analyze large numbers of detailed protocols from subjects working under naturalistic as well as experimental conditions. Although these grammars require considerable effort to develop and debug, they offer the dual advantages of (a) forcing researchers to define their analytic models precisely and (b) ensuring that those models are applied consistently across the data. They also make possible studies that would probably be too costly or too time consuming to be done using conventional methods. In the upcoming Reflections section, we discuss some of their limitations as well as additional opportunities we foresee.

Analytic Tools

Analysis is an iterative, open-ended process, but each iteration normally consists of two computer-processing steps. First, data—either entire parse trees or horizontal slices viewed as a protocol sequence—are filtered to produce one or more measures. Second, these measures—usually aggregated for all subjects taking part in an experiment or for subjects in one condition or category—are then analyzed using a standard statistical package.

Once a protocol for a session has been parsed, the resulting data can be viewed from two perspectives. First, it can be thought of as a parse tree, in which the *session* is the root symbol, consisting of a sequence of *modes* or *phases,* and so on. Although the parse tree can be analyzed directly, we often focus on a particular level of the parse tree, such as the conceptual product level or the mode level. Consequently, a second perspective that can be applied to grammars with fixed depth (e.g., the WE production rule grammar) views a horizontal slice of the parse tree as a sequence of protocol symbols analogous to the original action protocol recorded by the tracker. These symbols, however, are nonterminals in the grammar and, thus, represent a particular level of abstract inference, such as the effects of a given sequence of actions on a conceptual product, which we call the *delta product* level, or the sequence

of actions that ultimately comprise a particular instance of a cognitive mode, the *modal* level. Each level of the parse tree, thus, defines the user's behavior from a different cognitive perspective.

Filters. Filters, applied to entire parse trees or to horizontal slices of parse trees, produce values or distributions of values that quantify particular aspects of the user's cognitive behavior. Some of the specific filters we have developed are described next. They illustrate the kinds of measures that can be derived from the parsed data produced by grammars. For additional details, including their use in specific experimental studies, see Lansman et al. (1990).

Top-Down Index. Using the WE, writers can develop ideas for their paper in whatever order they like. One strategy for creating ideas is to create superordinate topics first, followed by subordinate topics. We call such an approach a *top-down strategy.* An alternative method is to generate subordinate topics first, group them, and then create appropriate superordinate topics. This would be a *bottom-up strategy.* Of course, a user may switch back and forth from one strategy to another during a session. The *top-down index* indicates the degree to which writers followed one or the other of these strategies. It is defined as the percentage of nodes (omitting the root node) that was created after the superordinate node was determined. If writers followed a purely bottom-up strategy, their top-down scores would be 0.00; if writers followed a purely top-down strategy, their scores would be 1.00; a value of 0.50 would indicate equal use of these two strategies. In our experiments, subjects' scores of top-downness ranged from 0.5 to 1.0, with a mean of 0.80 (Lansman et al., 1990).

Stage Index. This index assesses the extent to which planning time precedes writing/revising time. (The name, *Stage Index,* derives from a theory of writing that asserted that writers progress linearly through three stages: planning, writing, and revising. The index let us measure the degree to which this theory actually applies.) To understand the Stage Index, imagine computing for every minute of writing time the proportion of total planning time that preceded that minute of writing. These proportions are averaged across all the minutes of a writing session to compute the Stage Index. For example, if a subject had completed all planning before beginning to write, then for each minute of writing the proportion of planning that preceded that minute would have been 1.0, and the average, or the Stage Index, would be 1.0. The index can vary between close to 0 and 1.0. (It cannot be 0 because subjects using the WE must create at least one node in either Network or Tree Mode before they can begin to write.)

Struggle Index. This index assesses the difficulty, or struggle, writers have in developing an organizational structure for their papers. One indication of difficulty in developing a structure is a large number of actions in Network and Tree Modes versus a relatively small structure that is finally produced. The index is defined as the total number of planning operations divided by the total number of nodes created.

Conceptual Versus System Operations Index. Operations in Network and Tree Modes can be classified as being either a conceptual operation or a system operation. Conceptual operations are operations that contribute directly to the development of the conceptual structure for the writer's document. System operations are those that are concerned with controlling WE, per se. The index is defined as the ratio of conceptual operations to system operations.

This set of filters could be extended. They are dependent on the goals and perspectives of specific studies. Those described earlier constitute our current inventory, but, as we address new research issues, we expect the list to grow.

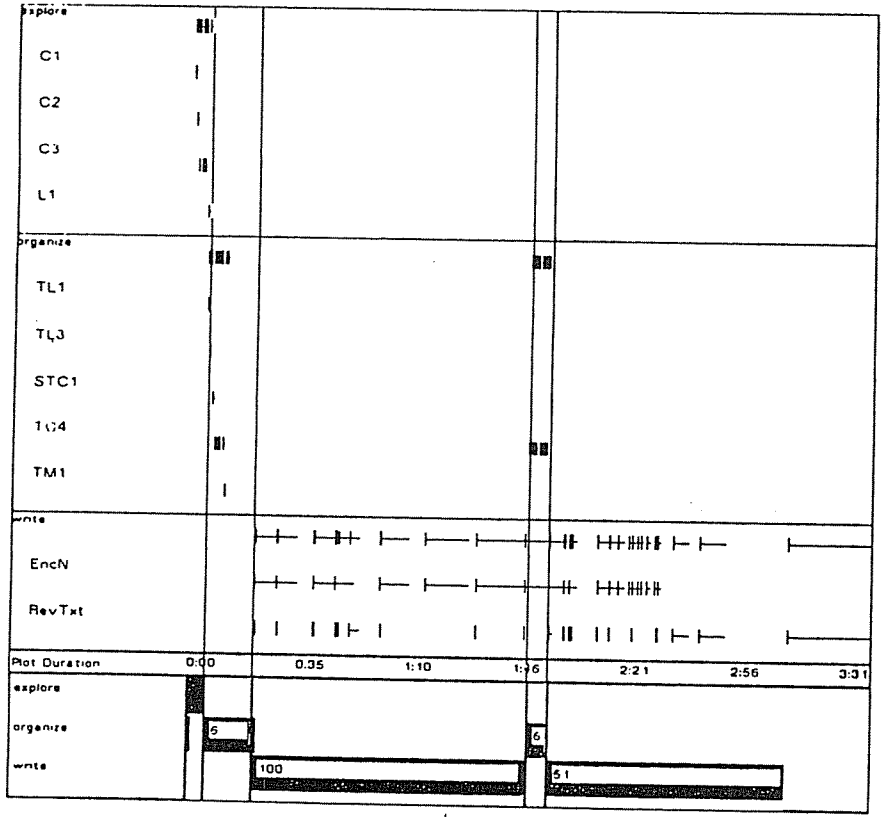
Statistical Analysis. Data—consisting of values produced by filters as well as simple counts of protocol symbols at various levels of the parse tree—are analyzed using a standard statistical package. These data are written to a file, and the file is processed by the statistical utility in a conventional manner. In the future, we hope to incorporate control of statistical analysis into a protocol management system. This could have methodological implications by making it significantly easier to carry out exploratory analyses. Some of our ideas about such a system are described in Young and J. B. Smith (1989).

Display Tools

Although data may be analyzed by automated tools, they must eventually be studied and interpreted by human beings who decide what they mean. To assist researchers with this essential task, we are developing an open-ended collection of visualization tools. In this collection, we differentiate between static tools and animated tools. The first produce a fixed image of one or more protocols, or associated data, from a specific analytic point of view. The second is a set of dynamic displays, usually shown on multiple workstation screens, coordinated by a replay of a session—As the session unfolds in time, the different visualization tools update their displays accordingly, showing various parsed or analyzed protocol data.

Static Tools. The *events-time distribution tool,* illustrated in Figures 6, provides a static representation of the frequency with which user events, grouped by analyst-determined categories, are distributed over a session. An

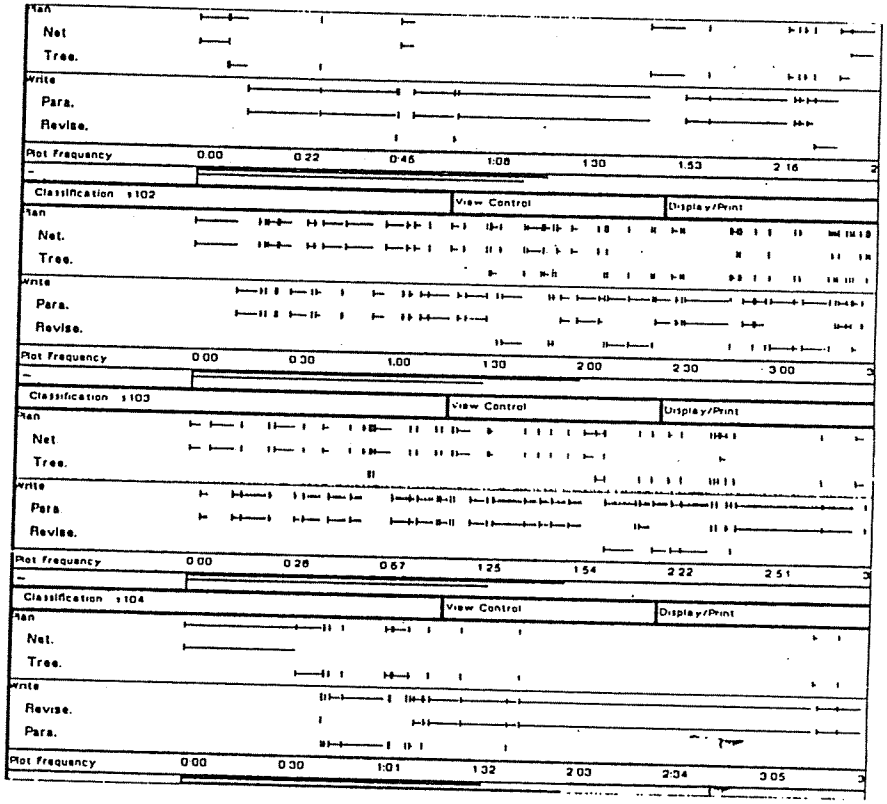
Figure 6. The events-time display provides an overview of all actions for a session, organized into categories defined by the researcher. Session time extends from left to right. The user's actions have been classified as explore, organize, or write. Each action corresponds to a tick mark, indicating when it occurred and its duration. Aggregate counts for the three classes are shown at the bottom.



event may correspond with a user action as recorded by a tracker or to symbols within a horizontal slice of a parse tree, as discussed previously, such as discrete changes to conceptual products (*delta product* events) are produced by sequences of actions.

The display is divided into two panels. In the left panel is a taxonomy of events, defined by the researcher. Frequency distributions for each type of event are displayed in the right panel of the screen, in which session time extends from left to right. The instances of each event type are represented by tick marks (short vertical lines) at the time each event begins. A horizontal line extending from the center of each tick mark (the tail) indicates the duration

Figure 7. Abbreviated events-time displays for four subjects, permitting comparison of their different planning and writing strategies.



of the event. The panel is divided into horizontal bands that correspond to the categories of the taxonomy defined in the left panel. A row of tick marks, thus, appears for each type of event and for the cumulative total for all events within a category. Changing the structure of the tree in the left panel changes the organization of the distributions in the right panel, and the displayed data are updated accordingly. The bottom section of the right panel shows a histogram that can display either the total number of events within each category or the total duration of events within the category. In Figure 6, the events have been organized into three categories—*explore, organize,* and *write.*

To display the behaviors of several subjects for purposes of comparison, we have developed an abbreviated version of the events-time distribution tool. This tool organizes events into only two categories: *planning* (composed of exploration and organizing events) and *writing* (composed of writing, per se, and revision events). Figure 7 shows an example screen of abbreviated

distributions for four subjects. The particular classification of events is under the control of the researcher and can be changed to fit his or her research perspective.

Animation Tools. The protocol data we are concerned with are inherently temporal: Actions take place one after the other in time. Consequently, users' strategies can be described as patterns in behaviors that occur over time. To enable researchers literally to see these patterns in temporal protocols, we have developed a set of animation tools in which time functions as the independent variable.

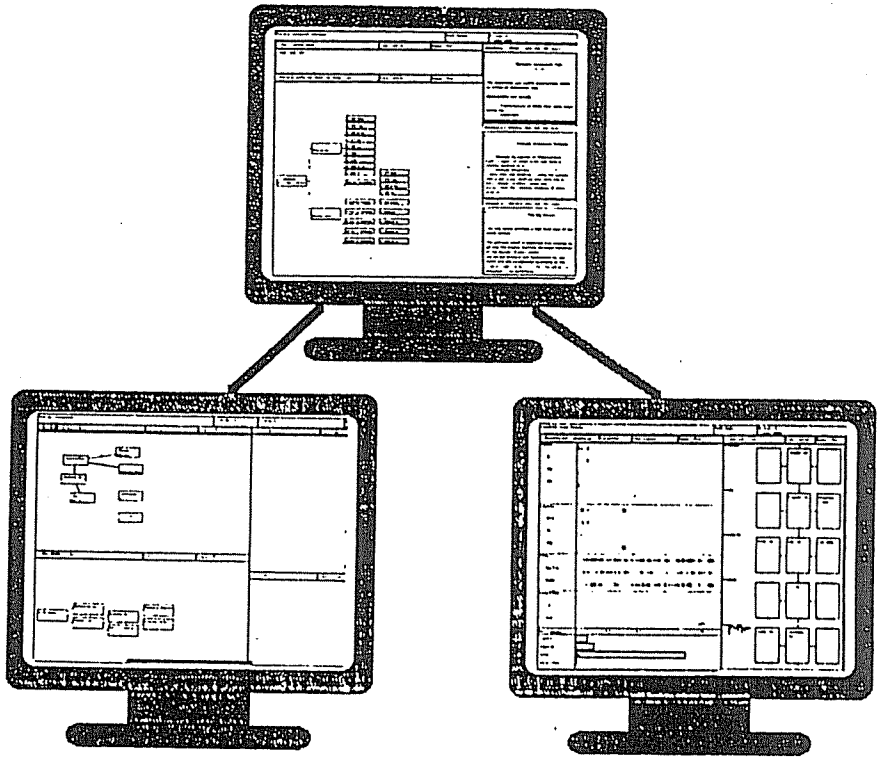
Animation tools are actually a collection of display tools from which the researcher may select. We typically work with a setup of three coordinated workstations. One workstation runs a replay of the session being viewed. The displays on the other two workstations include various data representation windows plus control and configuration windows. The various windows can be arranged to suit the researcher's preferences. Figure 8 shows a configuration (which is discussed later) for three workstations; the drawing is somewhat stylized in that the workstations are normally placed side by side on a table, not one above the others as shown in the figure.

The left screen in the configuration is the replay display. It is identical to the replay tool discussed earlier. Here, it serves as the central clock that coordinates all of the associated displays. As the subject's actions unfold in time on the replay screen, the other display tools, shown on the other screens, update their displays accordingly. We have found this display particularly useful for providing a sense of what a user was actually doing in a session that resulted in the more abstract views or references shown in the other displays.

The animated display is controlled by the researcher from a workstation that is normally in the middle (shown as the top screen in Figure 8; it is enlarged in Figure 9). The largest window, the configuration window, allows the researcher to select the particular protocol for display and the various display tools to be included in the configuration. Above it is the control window, which is used to stop and start the replay and, hence, the animation. As long as the cursor is in this control window, the replay will continue, action will follow action, and the other windows will update their displays accordingly. However, the researcher may stop the replay and, thus, all of the animation tools by moving the cursor to any of the other windows.

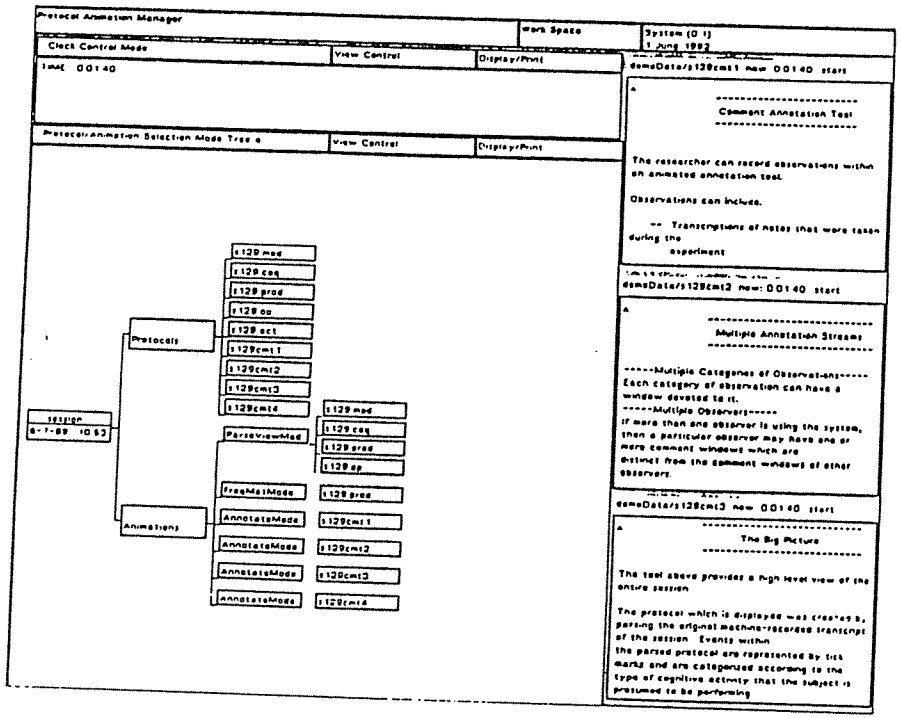
Two comment windows are shown on the right side of the screen. They are text editors in which written comments or other data can be noted and linked to a particular time or event in the protocol. Thus, the researcher may stop the animated display, by moving the cursor from the control window to one of the comment windows, and then write an observation about what was just seen in the session. When an annotation window is opened, it is stamped as having been opened at Time T1 relative to the session clock values included in each

Figure 8. **Configuration of three workstations that provide animated display of a user's session coordinated with various analytic perspectives. The top screen controls a replay of the session, shown on the lower left screen; the top screen also includes several windows for comments. The right screen shows an animated events-time display and a vertical slice of the parse tree for the session. All displays are updated with each user action shown in the replay.**



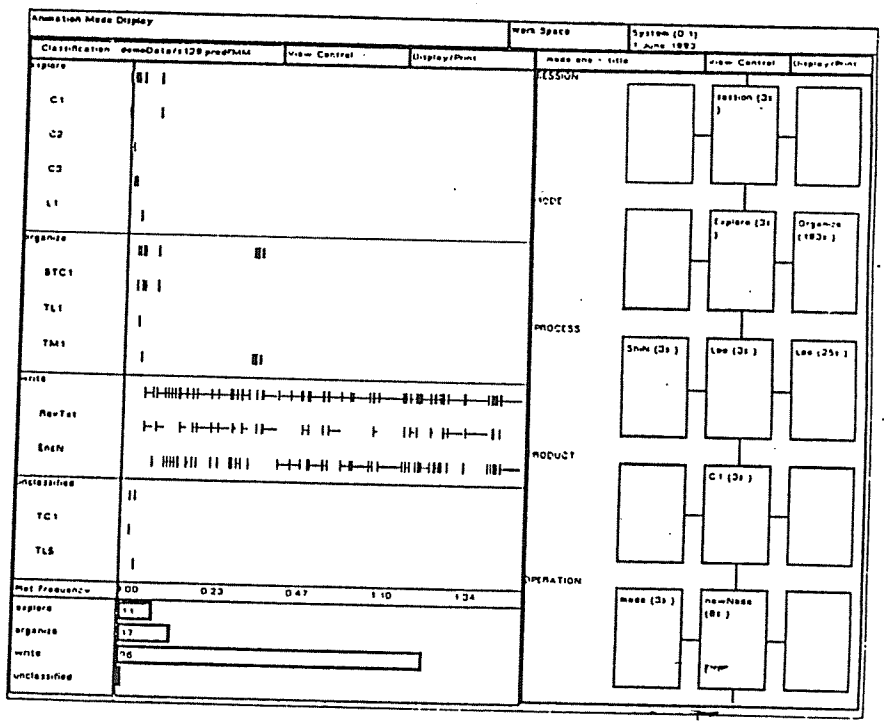
protocol record. When the researcher no longer wants the comment to be shown, he or she can close the window and it is stamped as closing at Time T2, again relative to the clock for the original session. When the session is replayed in the future, the comment will appear at Time T1 of the replay and will be displayed until Time T2. The researcher may create any number of comment windows that can be used to record different kinds of information. For example, a researcher might have three separate comment windows in which a transcription of a subject's concurrent think-aloud protocol is recorded in one, a cued retrospective protocol in the second, and the researchers' own observations in the third. In the future, we will explore linking video and voice data to the time line and coordinating its display with session replay.

Figure 9. **Enlargement of the workstation display, shown at the top of Figure 8. It controls the animation and includes multiple windows for comments from the researcher and others entered during replay. As comments are entered, they are time stamped according to the current action in the replay so that, in subsequent replays, they can be synchronized with the session.**



The screen on the right in Figure 8 and the enlarged one in Figure 10 contain two additional animated displays. On the left is a dynamic version of the events-time distribution tool described before. It is similar to the static tool except that, as the session unfolds, the vertical line (shown near the left edge in Figure 10) slowly moves across the display to the right to indicate the current session time and the current action in the replay. To the right of the events-time distribution is a second window, showing a vertical slice of the parse tree produced by the WE production rule grammar, described earlier. It shows four levels of the parse tree corresponding to the *cognitive mode*, *process*, *cognitive product*, and *operation* levels of the grammar. In the center column is the grammatical interpretation for the current event. The preceding and succeeding symbols at each level of the parse tree are shown to the left and right, respectively. As the replay runs, the information within these boxes is updated. Thus, the parse tree display tool enables the researcher to compare

Figure 10. **Enlargement of the workstation display, shown on the right in Figure 8. It includes an animated events-time display and a vertical slice of the parse tree for the session. A vertical line indicates the current action in the events-time display, and the slice of the parse tree is updated continuously throughout the replay.**



the interpretation produced by the grammar for a segment of the session, as shown in the parse tree, with the actual events, as shown in the replay. This capability can be used for a variety of analytic purposes as well as to validate and refine the particular grammar.

As previously noted, specific display tools can be selected and combined in various ways. Over the next few years, we plan to develop new displays that can be included within animation configurations including tools for working with multiple protocols collected from collaborative groups.

Data Management

Machine-recorded protocols permit researchers to gather large volumes of data. This can be a curse as well as a blessing; anecdotes abound of researchers with, literally, drawers full of keystroke data that have never been analyzed. Although automated tools, such as the grammars described in this

article, can help, problems remain. For studies of differences among groups of subjects, researchers must keep up with particular sets of protocols for particular subjects collected under particular conditions. For longitudinal studies, protocols must be selected and sequenced according to time of generation. During analysis, the researcher must keep track of which data have been processed by which filters and analytic programs. For animated display, consistent sets of data must be assembled and passed to the system. Thus, although these issues confront any HCI study, automated protocol tools magnify the need for effective data management.

Our efforts in protocol data management are still tentative. To date, we have carried out two exploratory projects concerned with managing protocol data and their analyses. The first concerned development of a prototype system for sorting, selecting, and sequencing protocols. The second involved a mathematical analysis of these tasks that could provide a formal basis for a comprehensive protocol analysis environment (see Hawkes, 1991).

The sort-and-select tool lets the researcher work with one file directory of protocols at a time, which normally represents the data for a single experiment. The researcher organizes and/or selects from the files in that directory by successively designating attributes. Protocols are then ordered or chosen according to the values in their headers for the specified attribute. Each designation of an attribute produces a further ordering (for a sort) or reduction in number (for a select) in the set of protocols. The final ordered/selected set of protocols is represented as a tree. Once a set of protocols is selected and ordered in this way, it can then be passed on to other analytic programs for processing.

To provide a well-defined basis for a protocol management system, we have analyzed the formal characteristics of trees of objects ordered or selected according to attributes and values, as described before. We have considered under what conditions an operator with given characteristics can be validly applied to protocols or other data with particular characteristics. For example, after selecting and/or ordering a set of protocols, as described earlier, the researcher might apply a filter program to count the number of planning actions in each protocol and output the number for each. A second filter might produce a vector of values, representing the distribution of planning events over the duration of the session. Problems could arise if the researcher attempted to analyze the second set of data, rather than the first, using an analysis of variance program. We see this work as eventually leading to a graphics-based direct manipulation tool that will allow the researcher to carry out different analyses by simply constructing data-flow diagrams in which different analytic processes and sets of protocols are represented by different node types.

Both of these tools would apply to UNIX-based systems. Walker, in work described earlier for PC-compatible systems, developed a simple interface

between his protocol data and a standard spreadsheet program. This allowed him to organize and display his data in different ways, count various items or patterns, and display his results in table and graph forms.

4. REFLECTIONS

In this section, we consider the tools and techniques we have described from a broader, more reflective point of view. Although we believe this methodology offers considerable promise for many different kinds of user studies, it is clearly better suited for some than for others. Consequently, we comment first on some of the characteristics that limit as well as suggest this approach. After that we discuss the issue of validating protocol grammars. The key questions here are: By what standards should we accept or reject what they tell us about users' cognitive interactions with systems? and How can we refine them to correct mistakes and make them more comprehensive and more sensitive?

4.1. Applicability

Any methodology is better suited for some problems than for others. How can a researcher tell whether it is appropriate for his or her needs? The use of any set of tools is, ultimately, limited by the imagination of the user. Consequently, our thoughts on the subject should not be taken as definitive but, rather, as suggestive, based on our experience to date.

First, instrumenting systems to record action protocols and building grammars to analyze these data both require considerable time and technical expertise. Consequently, from a simple costs–benefits perspective, this methodology is better suited for larger, more extensive studies that involve substantial amounts of data than for smaller, more focused studies that result in relatively small amounts of data. A major payoff comes after the tools have been developed or adapted and one can automatically analyze large volumes of data. For smaller studies, this approach may not be worth the effort.

The break-even point, however, may change. As protocol analysis environments become available that provide standard sets of tools and/or support development of new ones, the investment in the researcher's time and effort will go down. A more important breakthrough, however, would be to negotiate with major application developers a standard for producing action protocols as an option in their software. The common object interface movement is a start in this direction, but an action protocol standard will have to go further to outline the user selections and changes to data to be reported. Such a standard would benefit vendors as much as researchers, in that it would facilitate their own usability studies as well as studies by outside

researchers using their systems. Thus, although developing a standard will require considerable political as well as technical effort, it has a fighting chance simply because it is in the self-interest of both developers and researchers.

A second issue concerns the task being studied. The methodology probably works better for tasks that are conceptually complex than for simple transactions or for short, dissociated tasks. The action protocol data make it possible to relate long sequences of actions to one another and to infer high-level goals and effects, including effects on specific data objects. Thus, the researcher can trace development of a data object over a session or across multiple sessions. These concerns come into play for design, development, problem-solving, and other conceptual construction tasks, but they are not at issue in matters of simple interface efficiency. The methodology is better suited for tasks such as expository writing, software development, VLSI, and architectural design; it is less well suited for developing a better automatic teller machine or a better workstation for long-distance operators, although these latter tasks are important and increasing their efficiency can produce substantial financial savings.

A third issue concerns observational context. Although these tools and techniques can be used in the laboratory, they can also be used in naturalistic settings in which video, think-aloud, or observer-based methods are impractical or inappropriate. With the growing interest in the HCI community in situated behavior, developing methods for nonintrusive observations of naturalistic system use is an increasingly important concern. A potential problem that could arise from this methodology is unauthorized monitoring of workers or other individuals participating in a study conducted in an actual-use setting. Although issues of privacy and confidentiality are routinely considered in designing experimental studies, the methodology described here makes those concerns even more important.

Although the approach is applicable to a wide range of tasks and problems, let us shift the focus to studies for which it is less well suited. Already noted are its inefficiency for small studies that involve limited amounts of data and for tasks that consist of short, independent activities. Another limiting factor is access to the source code of the system being used in the study. Currently, we know of no way to collect action protocols or to replay a session using these data without modifying the system. User interface toolkits can be modified or supplemented to make this task easier, but sensors must still be placed at appropriate points in the program. As already discussed, developing a standard for action protocols and persuading system developers to adopt it would alleviate this problem, but, for now, the issue remains. We should point out, however, that if other forms of protocols, including those produced by hand, are encoded using a standard format, such as the protocol description language described earlier, these data could potentially be handled

by many of the tools we would expect to find in a standard suite, including grammars and some display programs.

Another limitation includes noncomputer behaviors. Relying on action protocols or other system-recorded data will miss what a user does when not interacting with the system. For example, if one were interested in the effectiveness of a printed tutorial in helping users learn a new system, one may want to see what users do between system actions, observe facial expressions, note conversations with other users, and so on. For studies such as these, video or observer-recorded data may be more appropriate. Similarly, if one is interested in factors such as motivation or intentions that lie behind system actions, one may need think-aloud data. Thus, the methodology may miss important information for some studies.

The issue, however, may be more one of supplementing the approach than replacing it. The researcher may wish to merge additional forms of data with action protocols during the analysis. Support for doing this could be provided by integrating techniques such as those described for VideoNoter (Roschelle et al., 1990) and the TI system (Hammontree et al., 1992) into the tools discussed. For example, video and think-aloud recordings could be incorporated and synchronized with session replay as a form of animated display, analogous to the coordinated comments option, described before.

Although the methodology we have described is quite general, it is more appropriate for some studies than for others. Some limitations are inherent, but most are matters of tradeoff. As better tools and better environments for building and using them become available, these techniques should become applicable to a larger and larger class of problems.

4.2. Validation

Because grammars play a key role in the methodology here, we need to consider how to validate and refine them. It is probably impossible to "prove" a user model implemented as a grammar, but a body of evidence can be assembled to support a claim of validity and to refine and correct it. Thus, validating a grammar is a matter of constructing a particular kind of rhetorical argument, rather than following a rigorous, well-defined procedure that will lead to a definitive answer. At least four kinds of evidence can be accumulated.

First, we can test the adequacy of a grammar. If the claim is that a grammar models users' behaviors for a given task they perform while using a given system, based on data (i.e., protocol) produced by that system, then the grammar should successfully parse any protocol produced under those circumstances. The test for this condition is to parse a substantial number of user protocols and show that the parser does not break or fail to parse any

protocol. As mentioned earlier, we tested the WE production rule grammar by successfully parsing approximately 50 protocols, after initial debugging. J. Q. Walker (1991) described in detail his experience of debugging and refining the PROSE II grammar that he eventually used to parse more than 100 protocols. Thus, although one can never be sure that the grammar will not fail on the next protocol encountered, one can reduce the chance of this happening by correcting oversights in the rules, generalizing features, and so on, so that the probabilities of failure steadily decrease.

A second body of evidence can be gathered by comparing the interpretation of a user's session with the objective data generated in the session. With respect to the writing systems developed by our project, users generated various component structures and plans for documents expressed as graph structure and then wrote text to flesh out those structures. These semantically rich data can be examined by the researcher or by an independent judge and then matched against some of the inferences drawn by the grammar. For example, if the grammar infers that a particular set of nodes constitutes a cluster of semantically related concepts, an analyst can examine the identifying labels of the individual nodes to see if they are apparently related. A more in-depth analysis can be performed on the document to see whether or not they are related in the narrative. If the parse and the data are inconsistent, it is possible in a context as complex as this that the user originally saw the group as related but later changed his or her mind. Task data can lend support to a grammar's interpretation of user behavior, but it must be carefully evaluated and weighed in relation to other evidence.

A third test introduces a weak form of prediction. Arising from the strong experimental tradition of cognitive psychology, HCI research has traditionally been most comfortable with models that predict users' behavior. This is seen, for example, in the selection rules that make up one of four basic components in GOMS models. GOMS models have typically predicted one of two things. They have predicted the amount of time it will take a user to perform simple tasks such as making an editing change using a text editor. These predictions are derived by decomposing tasks into sequences of discrete operations and determining the time required for them to take place as derived from fundamental cognitive measures. Examples of such derivations can be found in Card et al. (1983) and, in more general terms, in Newell (1990). Second, GOMS models have been used to predict the particular method—a habitual sequence of operations used to accomplish a basic task—a user will engage subject to a prescribed set of conditions. For example, one can predict which of several known methods a user will engage to make a particular kind of editing change, given the relative proximity of the cursor to the spot where the changes are to be made.

Models that can predict the amount of time required for a simple activity or which of several methods will be used under varying conditions are

substantial accomplishments. But the phenomena they address and the phenomena addressed by the grammars described here are vastly different in scale and complexity. Users' strategies for planning, writing, and editing documents extend over many hours, if not days, and are subject to a bewildering array of unanticipated developments. Imagine, if you will, trying to predict when a person will run into a dead end while following a particular line of thought or when someone will be reminded of an idea that can be incorporated into a document from a stray conversation overheard in the background. A predictive model that could handle behaviors of this sort in semantically rich contexts would constitute a form of artificial intelligence capable of simulating the expository writing task. This is not to say that all aspects of complex behavior are unpredictable. Indeed, we can predict behaviors such as the average length of time spent in continuous work episodes based on attention span and limits of short-term memory. But predicting large-grained phenomena or semantically rich behaviors lies far beyond current capabilities. Consequently, GOMS and other traditional forms of predictive models and the descriptive grammars discussed here should be viewed as complementary perspectives that may eventually move toward one another; at present, however, they are at opposite ends of the spectrum with respect to the granularity of users' behaviors they address.

What protocol grammars can *predict,* in a rather loose sense of that term, is that a user will confirm the interpretation of his or her actions generated by the grammar. Thus, if a subject is asked to view a replay of a recent session, the subject can be asked to narrate his or her thinking, intentions, and so on, cued by the replay. The subject could also be asked to confirm or disconfirm specific portions of the parse or even to review the entire parse. Although this procedure may seem unusual, it is similar to that used to verify natural language grammars. After being subjected to several kinds of formal tests, natural language grammars are ultimately tested against native speakers' perceptions that the sentences they generate or the parses they produce are idiomatic or acceptable. An analogous test can be applied to protocol grammars.

The final, and perhaps ultimate, test of any model is its utility—that it will ultimately tell us something that is interesting and useful. The grammars described herein have told us many things that we did not know about the subjects, tasks, and systems we studied. The results were interesting to us. We have cited several published discussions of results, and readers have told us they found them interesting as well. But as a methodological issue, we must await more extensive development and use of grammars similar to the ones described here to see if they will meet this requirement in general.

By verifying a grammar against a substantial set of protocols, by comparing its interpretation against the objective data, and by comparing its inferences against what a subject says he or she was doing or thinking during

a session, we do not confirm or disconfirm a grammar; rather, we gather evidence that lends credibility to parts of the grammar and point out areas that require correction or refinement. Validation is probably never completed, because additional rules or nuances can always be added to make the grammar more sensitive or more comprehensive. It is in this cycle of theorizing, building systems, and testing that knowledge is ultimately produced.

5. CONCLUSION

In summary, we described tools and techniques that, taken together, constitute a methodology for studying computer-mediated cognition for complex tasks. We placed this work in context with issues and tools associated with other forms of protocol data. We then described the individual tools developed by our research group, tracing the flow of data from initial protocol tracking through automated analysis to animated display to facilitate visualization and understanding. The discussion concluded with our reflections on the applicability of this methodology and issues of validation.

As we look to the future, we will continue to view methodology as a legitimate area of research in its own right, although guided and informed by substantive concerns. We expect to focus our attention on issues related to collaborative groups, to systems to support those groups, and to building an understanding of the conceptual and social processes that comprise collaboration. Thus, we hope eventually to build a process model of computer-based collaboration. To address so lofty a goal will require significant prior research in methodology, including more powerful formalisms in which to express such a model.

Work on collaboration methodology is relevant to conventional HCI methodology, because the individual user can be treated as a special case. Over the next few years, we plan to build a comprehensive environment for managing collections of protocols, including machine-recorded, hand-coded, and video data. Although its main goal will be to support extended HCI analyses, it will also provide a context in which new tools can be developed and incrementally tested against actual protocol data.

Acknowledgments. A number of individuals have contributed both ideas and effort to the work described here; they include Gordon Ferguson, Marcy Lansman, Steve Weiss, Dick Hayes, Irene Weber, Oliver Steele, Mark Rooks, Doug Shackelford, Rick Hawkes, Matt Barkley, Hong Li, Mu-Yu Yang, John Hilgedick, and John Walker. We are also grateful for the thorough and thoughtful comments from Jan Walker and her anonymous reviewers; they offered a number of suggestions that, we feel, greatly improved the article. The problems and errors, of course, remain the authors'.

Support. This research was supported by the National Science Foundation (Grant Nos. IRI-8519517, IRI-8817305, and IRI-9015443), The Army Research Institute (Contract No. MDA903-86-C-345), and The Office of Naval Research (Contract No. N00014-86-K-00680).

REFERENCES

Bales, R. F., & Cohen, S. P. (1979). *SYMLOG: A system for the multiple level observation of groups.* New York: Free Press.

Blakeley, K. D. (1990). *The application of modes of activity to group meetings: A case study* (Tech. Rep. No. TR90-045). Chapel Hill: University of North Carolina, Department of Computer Science.

Card, S. K., Moran, T. P., & Newell, A. (1983). *The psychology of human-computer interaction.* Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Ericsson, K. A., & Simon, H. A. (1980). Verbal reports as data. *Psychological Review, 87,* 215-251.

Ericsson, K. A., & Simon, H. A. (1984). *Protocol analysis: Verbal reports as data.* Cambridge, MA: MIT Press.

Fisher, C. (1988). Advancing the study of programming with computer-aided protocol analysis. In G. Olson, E. Soloway, & S. Sheppard (Eds.), *Empirical studies of programmers* (pp. 198-216). Norwood, NJ: Ablex.

Foley, J. D., & Wallace, V. L. (1974). The art of natural graphic man-machine conversation. *Proceedings of the IEEE, 62,* 462-471.

Hammontree, M. L., Hendrickson, J. J., & Hensley, B. W. (1992). Integrated data capture and analysis tools for research and testing on graphical user interfaces. *Proceedings of the CHI '88 Conference on Human Factors in Computing Systems,* 431-432.

Hawkes, R. M. (1991). *Mathematical basis for a system to manage automated protocol analysis* (Tech. Rep. No. TR91-024). Chapel Hill: University of North Carolina, Department of Computer Science.

Hayes, J. R., & Flower, L. S. (1980). Identifying the organization of writing processes. In L. Gregg & E. R. Steinberg (Eds.), *Cognitive processes in writing* (pp. 3-30). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Kieras, D., & Polson, P. (1985). An approach to the formal analysis of user complexity. *International Journal of Man-Machine Studies, 22,* 365-394.

Lansman, M. (1991). *Organize first or write first? A comparison of alternative writing strategies* (Tech. Rep. No. TR91-014). Chapel Hill: University of North Carolina, Department of Computer Science.

Lansman, M., Smith, J. B., & Weber, I. (1990). *Using computer-generated protocols to study writer's planning strategies* (Tech. Rep. No: TR90-033). Chapel Hill: University of North Carolina, Department of Computer Science.

Losada, M., Sanchez, P., & Noble, E. E. (1990). Collaborative technology and group process feedback: Their impact on interactive sequences in meetings. *Proceedings of CSCW '90,* 53-64. New York: ACM.

Lucke, E., Pagery, P. D., & Brown, C. R. (1987). User requirement gathering through verbal protocol analysis. In G. Salvendy (Ed.), *Cognitive engineering in the design of human-computer interaction and expert system.* Amsterdam: Elsevier.

Mackay, W. E. (1989). EVA: An experimental video annotator for symbolic analysis of video data. *SIGCHI Bulletin, 21*(2), 68–71.

Mackay, W. E., Guindon, R., Mantei, M., Suchman, L., & Tatar, D. G. (1988). Video: Data for studying human-computer interaction. *Proceedings of the CHI '88 Conference on Human Factors in Computing Systems,* 133–137.

Newell, A. (1990). *Unified theories of cognition.* Cambridge, MA: Harvard University Press.

Newell, A., & Simon, H. A. (1972). *Human problem solving.* Englewood Cliffs, NJ: Prentice-Hall.

Nisbett, R. E., & Wilson, T. D. (1977). Telling more than we can know: Verbal reports on mental processes. *Psychological Review, 84,* 231–259.

Olson, G. M., & Olson, J. S. (1991). User-centered design of collaboration technology. *Journal of Organizational Computing, 1*(1), 61–83.

Reisner, P. (1981). Formal grammars and human factors design of an interactive graphics system. *IEEE Transactions on Software Engineering, 7*(2), 229–240.

Roschelle, J., Pea, R., & Trigg, R. (1990). *VIDEONOTER: A tool for exploratory video analysis* (Research Rep. No. IRL90-0021). Palo Alto, CA: Institute for Research on Learning.

Sanderson, P. M., James, J. M., & Seidler, K. S. (1989). *SHAPA: An interactive software environment for protocol analysis* (Tech. Rep. No. 89-09). Urbana-Champaign, IL: Engineering Psychology Research Laboratory.

Siochi, A. C., & Ehrich, R. W. (1991). Computer analysis of user interfaces based on repetition in transcripts of user sessions. *ACM Transactions on Information Systems, 9*(4), 309–335.

Smith, J. B. (1992). *Collective intelligence in computer-based collaboration: An introduction* (Tech. Rep. No. TR92-011). Chapel Hill: University of North Carolina, Department of Computer Science.

Smith, J. B., & Lansman, M. (1989). A cognitive basis for a computer writing environment. In B. K. Britton & S. M. Glynn (Eds.), *Computer writing aids: Theory, research, and practice.*

Smith, J. B., & Lansman, M. (1992). Designing theory-based systems: A case study. *Proceedings of the CHI '92 Conference on Human Factors in Computing Systems,* 479–488.

Smith, J. B., Rooks, M. C., & Ferguson, G. J. (1989). *A cognitive grammar for writing: Version 1.0* (Tech. Rep. No. 89-011). Chapel Hill: University of North Carolina, Department of Computer Science.

Smith, J. B., & Smith, F. D. (1991). *A hypermedia system for artifact-based collaboration* (Tech. Rep. No. 91-021). Chapel Hill: University of North Carolina, Department of Computer Science.

Smith, J. B., Smith, F. D., Calingaert, P., Hayes, J. R., Holland, D., Jeffay, K., & Lansman, M. (1990). *UNC collaboratory project: Overview* (Tech. Rep. No. 90-042). Chapel Hill: University of North Carolina, Department of Computer Science.

Smith, J. B., Weiss, S. F., & Ferguson, G. J. (1987). A hypertext writing environment and its cognitive basis. *Hypertext '87 Proceedings,* 195–214. New York: ACM.

Smith, J. B., Weiss, S. F., Ferguson, G. J., Bolter, J. D., Lansman, M., & Beard, D. V. (1987). WE: A writing environment for professionals. *Proceedings of the National Computer Conference '87,* 725–736. Reston, VA: AFIPS Press.

Suchman, L., & Trigg, R. (1991). Understanding practice: Video as a medium for reflection and design. In J. Greenbaum & M. Kyng (Eds.), *Design at work: Cooperative design of computing systems* (pp. 65–89). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Swarts, H., Flower, L. S., & Hayes, J. R. (1984). Designing protocol studies of the writing process: An introduction. In R. Beach & L. S. Bridwell (Eds.), *New directions in composition research* (pp. 53–71). New York: Guilford.

Trigg, R. H. (1989). Computer support for transcribing recorded activity. *SIGCHI Bulletin, 21*(2), 72–74.

Walker, J. H., Young, E., & Mannes, S. (1989). A case study of using a manual online. *Machine-Mediated Learning, 3,* 227–241.

Walker, J. Q., II. (1991). *Automated analysis of computer-generated software usage protocols: An exploratory study.* Unpublished doctoral dissertation, University of North Carolina, Department of Computer Science, Chapel Hill.

Waterman, D. A., & Newell, A. (1971). Protocol analysis as a task for artificial intelligence. *Artificial Intelligence, 2,* 285–318.

Waterman, D. A., & Newell, A. (1973). PAS-II: An interactive task-free version of an automatic protocol analysis system. In *Proceedings of the Third IJCAI* (pp. 431–445). Menlo Park, CA: Stanford Research Institute.

Weber, I. (1992). *The effects of domain knowledge on writers' cognitive strategies.* Unpublished MS thesis, University of North Carolina, Department of Psychology, Chapel Hill.

Woods, W. A. (1970). Transition network grammars for natural language analysis. *Communications of the ACM, 13,* 591–602.

Young, F. W., & Smith, J. B. (1989). *Structural data analysis: A cognition-based design for data analysis software* (Tech. Rep. No. TR89-027). Chapel Hill: University of North Carolina, Department of Computer Science.

HCI Editorial Record. First manuscript received September 17, 1991. Revision received August 11, 1992. Accepted by Jan Walker. Final manuscript received September 25, 1992. — *Editor*