A Cognitive Grammar for Writing: Version 1.0

TR89-011
April, 1989

John B. Smith
Mark C. Rooks
Gordon J. Ferguson

The University of North Carolina at Chapel Hill
Department of Computer Science
CB#3175, Sitterson Hall
Chapel Hill, NC 27599-3175
919-962-1792
jbs@cs.unc.edu

A TextLab Report

Portions of this research were supported by the National Science Foundation, Grant #IRI-8519517 and the Army Research Institue Contract #MDA903-86-C-345.

Table of Contents

Overview

A cognitive grammar, for purposes of this discussion, is a computer program that interprets the actions of a user working with an interactive application system in order to infer the cognitive activities taking place in the mind of that user. The "language" that is parsed is the set of concurrent protocols automatically recorded by the computer system for sessions in which users work with that system. The resulting parse tree is a representation of a users' strategy for a session or task. The terminal symbols of the grammar are basic user actions, such as selecting a particular menu option or designating a position in a window with the mouse. The nonterminals are symbols that designate the researcher's interpretation of the user's cognitive acts as indicated by the system actions selected. Thus, a cognitive grammar can be considered a formal descriptive model of users' cognitive interaction with a particular computer system in accord with a particular analytic perspective.

When used in conjunction with an application system that produces machine-recorded protocols, a cognitive grammar can provide a form of automatic protocol analysis. Thus, the researcher can study statistically significant samples of user sessions. Thus, the researcher can study the effects of long-term experience with a system, the differences in strategies among various groups of users, patterns of individual differences, and other actual-use issues. Such studies are impractical for think-aloud and other prtotocol methods that require human encoding. Shifting protocol studies from, literally, a hand craft to an automated procedure could be an important step toward basic principles that can guide development of more natural and more useful interactive systems and toward fundamental insights into complex mediated cognitive behavior.

The grammar described here is part of a larger project in which we are building an advanced hypertext writing environment and then using that environment to study writers' cognitive strategies. It is the principal tool we use to analyze machine recorded protocols for writers working with our system.

The discussion of the grammar is divided into two parts. In Part 1, we discuss its background. We first describe its theoretical basis with respect to compostion theory and cognitive psychology. Second, since the grammar characterizes the cognitive behavior of writers using a particular system, we briefly describe that system. Finally, we discuss the grammar as it fits within a set of analytic tools and as it relates to several methodological issues -- especially protocol analysis techniques. In Part 2, we describe the grammar, itself, and illustrate its use with a sample protocol.

We denote the grammar as version 1.0 to indicate that it describes our views at a particular time. This is a tough nut to crack, and we do not believe that our current understanding is complete. Consequently, we anticipate producing other versions in the future. In the meantime, we issue this report to document the current form of the grammar and to encourage discussion that can help us refine it.

Part 1: Background

1.0 Theoretical Basis

As noted above, a cognitive grammar implictly includes in its definition a particular analytic perspective, regardless of whether its developers are aware of that perspective or not. Here, we wish to make explicit the theoretical basis for the grammar described in this report.

Synthesizing concepts from cognitive psychology, reading comprehension, and composition theory, we suggest that writing (and other open-ended intellectual activities) draws on a number of different *cognitive modes* [Smith & Lansman, 1987]. We view a cognitive mode as a particular way of thinking that is engaged in for a specific purpose, is more or less constrained relative to other modes, emphasizes certain cognitive processes, which are used to create certain forms of (intermediate) cognitive products. Thus, a cognitive mode is an interdependent combination of goal(s), constraints, processes, and products.

Figure 1 shows seven modes that we believe are important for expository writing. As an illustration of the concept, consider the differences between exploratory and organizational thinking .

Many writers engage in an early exploratory mode of thinking in which the goal is to externalize ideas, consider various possibilities, and to gain a general sense of the material available to be included in the document. In this mode, constraints are loosened, relative to other modes, to encourage creativity and alternative perspectives. The cognitive processes that are favored are memory recall, associative thinking, categorizing, and noting basic subordinate and superordinate relations. Consequently, the intellectual products produced tend to be concrete representations of ideas, clusters of related ideas, and small conceptual structures, often represented graphically.

Organizational mode is a very different way of thinking. Here, the goal is to work out the overall plan for the document to be written. Consequently, organizational thinking is much more constrained than exploratory thinking. Thinking tends to be logical and controlled, emphasizing analysis, synthesis, and abstract construction. The product that is built is a single large, often hierarchical, structure.

While different modes represent different ways of thinking, they are not independent from one another. The cognitive products created in one mode often become the raw material that is worked on in another. For example, a small hierarchical relation created during exploration might be incorporated into the larger hierarchical structure being built during organization. Thus, intermediate products tend to flow between modes in an overall process of conceptual refinement.

The cognitive grammar described in Part 2 incorporates this theory of cognitive modes by including nonterminal symbols that represent the *cognitive products* developed by writers, the *cognitive processes* used in their creation or transformation, and the *cognitive modes* engaged by writers during a session.

Figure 1:

Cognitive Modes for Writing

	Processes	Products	Goals	Constraints
Exploration	•Recalling •Representing •Clustering •Associating •Noting subordinate superordinate relations	•Individual concepts •Clusters of concepts •Networks of related concepts	•To externalize ideas •To cluster related ideas • To gain general sense of available concepts •To consider various possible relations	•Flexible •Informal •Free expression
Situational Analysis	•Analyzing objectives •Selecting •Prioritzing •Analyzing audiences	•High-level summary statement •Prioritized list of readers(types) •List of (major) actions desired	•To clarify rhetorical intentions •To identify & rank potential readers •To identify major actions •Consolidate realization •To set high-level strategy for document	•Flexible •Extrinsic prespective
Organization	•Analyzing •Synthesizing •Building abstract structure •Refining structure	•Hierarchy of concepts •Crafted labels	•To transform network of concepts into coherent hierarchy	•Rigorous •Consistent •Hierarchical •Not sustained prose
Writing	•Linguistic encoding	•Coherent prose	•To tranform abstract representation of concepts & relations into prose	•Substained expression •Not (necessarily) refined
Editing: Global Organization	•Noting large scale relations •Noting & correcting inconsistencies •Manipulating large scale structural components	•Refined text structure •Consistent structural cues	• To verify & revise large-scale organizational components	• Focus on large-scale features and components
Editing: Coherence Relations	•Noting coherence relations between sentences & paragraphs •Restructing to make relations coherent	•Refined paragraphs and sentences •Coherent logical relations between sentences and paragraphs	•To verify & revise coherence relations within intermediate sized components	•Focus on structural relations among sentences & paragraphs •Rigorous logical and structural thinking
Editing: Expression	•Reading •Linguistic analysis •Linguistic trasformation •Linguistic encoding	•Refined prose	•To verify & revise text of document	•Focus on expression •Close attention to linguistic detail

1.1 The WE System

While the theory of modes is quite general, strictly speaking the grammar incorporating that theory characterizes writers' cognitive interaction with one particular computer Writing Environment -- WE 1.0 [Smith, et.al, 1986; Smith, et.al, 1987]. Consequently, one should have a general sense of that system in order to understand the grammar.

The architecture of WE 1.0 closely matches the cognitive architecture implicit in the theory of modes. The system is multimodal. It includes different system modes that correspond with the major cognitive modes for writing. A network mode, shown in the upper left quadrant of Figure 2, is provided for exploration. In it, the user can represent ideas as nodes (small boxes with a label inside), move them around to form clusters, link them to one another to denote more explicit relationships and to form conceptual structures. The underlying rules for network mode are those for a directed graph, thus providing the user considerable flexibility with respect to the relations and structures that can be formed.

A tree mode, shown in the lower left quadrant of Figure 2, supports both the organization and the global editing modes. Here the user can build and edit the conceptual structure for the document as a whole, expressed graphically as a tree or organization chart, by creating parent and child relations among nodes or by moving structures created earlier in network mode into the tree. Since the underlying rules are those of a hierarchy, the user gives up flexibility in using tree mode but gains assurance that the structure produced will be a well-defined hierarchy.

An editor mode, shown in the lower right quadrant of Figure 2, is provided for writing and for editing sentences and paragraphs. By selecting a node in either tree or network mode, the user gains access to a text editor with which to write or edit a block of text that will be associated with that particular node.

Finally, a linear text mode, shown in the upper right quadrant of Figure 2, is provided for coherence editing. The system produces a continuous linear perspective of the text by stepping through the tree. In text mode, users can verify and revise, as needed, the transitions between text contained in separate nodes. They can also move segments of text from one node to another.

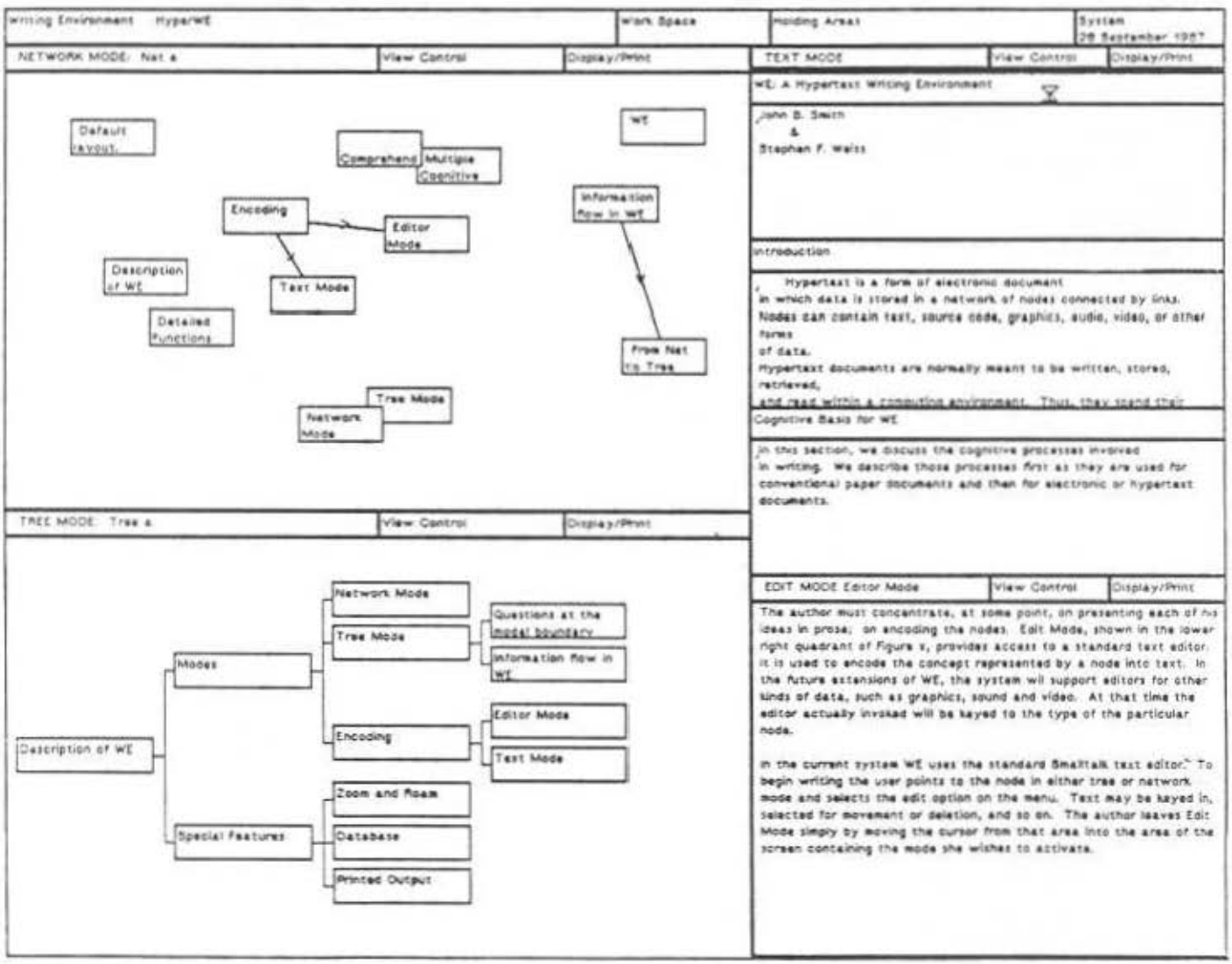
Thus, six of the seven cognitive modes shown in Figure 1 are supported by the four system modes of WE 1.0, shown in Figure 2. The one cognitive mode not covered is reader analysis mode, which is left to extrinsic heuristics, such as those described in [Smith & Smith, 1987]. Thus, system architecture matches underlying cognitive architecture implicit in the concept of cognitive modes and the flow of cognitive products among them.

1.2 Methodology

Cognitive grammars can provide alternative methods for studying human-computer interaction and other issues of mediated cognitive behavior. Consequently, in order to understand the grammar, one needs to understand how it relates to other methods. As mentioned above, our project is ultimately concerned with substantive issues dealing with writing and other open-ended design tasks. To help us with

Figure 2:

WE System Modes
(Network, Tree, Editor, & Text)



experiments that address these issues, we have developed several other tools that we use in conjunction with the grammar. These include a protocol recording function embeddded in WE, a session replay function, the cogntive grammar that is the primary topic of this report, and several display tools that assist interpretation of the parses produced by the grammar. In this section, we discuss these tools briefly, relating each to one another and to other methods used in studying human-computer interaction.

1.3 Concurrent Protocols

To help us gather protocols of users of the system unobtrusively and in a form ready for analysis, we installed a tracking function in the WE system that records each user action, its time, and other information of interest, such as the location in network mode where a node is created. These action-level data are written by the system to a file ready for analysis.

This approach contrasts with conventional methods for studying human-computer interaction. The most common form of data used for this purpose as well as to evaluate cognitive models, in general, has been think-aloud protocols. More recently, some researchers have begun to use keystroke records of user sessions. Both approches present problems that are overcome by action-level protocols.

Think-aloud protocol methods were developed by Newell, Simon, and others at Carneige-Mellon University to study complex, problem-solving behavior. Using this technique, the researcher asks subjects to narrate their thinking continuously while performing a given task. By doing so, the researcher is no longer forced to observe only the external behavior of subjects but is given a window into their minds and can observe, at one-step remove, subjects' thinking. While this technique has provided a rich source of information, it has also generated considerable controversy as to the validity of the data and the possible interference verbalization may have with the task being performed [Nisbett & Wilson, 1977]. Ericsson and Simon [1980] have answered their critics by arguing that concurrent verbal protocols do constitute valid data for what they term Level 1 verbalization -- verbalization of concepts that are stored in short term memory in verbal form. They found in their studies no evidence that concurrent think-aloud protocols affect this type of cognitive processing or that such data are incomplete or distorted. However, for Level 2 and Level 3 conditions -- respectively, verbalization of data that would not be heeded as part of the cognitive process and verbalization of data that is not part of the cognitive process and, hence, must be generated -- think-aloud protocols did significantly change the cognitive process, especially recognizing complex patterns and relationships presented visually [Ericsson & Simon, 1980; Claparede, 1933; Henry, 1934]. Computer systems that run on workstations with high-resolution graphic displays normally use representations that are highly spatial and provide control by direct manipulation of spatially located icons. Consequently, using think-aloud protocols to evaluate cognitive models for users of this type of system should be expected, from a theoretical perspective, to result in significant distortions.

Think aloud protocols also present problems of reliability. Typically, they are not used in their raw form but, rather, are coded according to some set of categories in which the model under examination is defined [Swarts, Flower & Hayes, 1984]. While training and practice increase reliability and consistency, encoding remains a highly subjective process that is subject to error on the order of 25% [Hayes & Flower, 1980].

A third issue is practicality. An hour of think-aloud data often requires fifteen pages for transcription [Hayes & Flower, 1980]. This is an enormous explosion of information that places significant limits on the number of subjects that one can study and the range of questions that can be examined.

Keystroke protocols record every keystroke performed by the user of a system. While they solve the problems of validity and reliability raised by think-aloud protocols, they still present practical problems. They produce substantially more data than action protocols. They also increase the complexity of any grammar that would be used to parse them since the grammar would have to include the full interpretitve capability of the user interface as the first stage in a parse.

Thus, we believe that automatically generated action-level protocols are an attractive alternative to both think-aloud and keystroke protocols.

A second tool we built is a replay program that takes the session transcript created by the tracker and recreates the session. Thus, we can visually inspect subjects' use of the system by watching the session as if it were a video recording of the screen, except we can speed it up, slow it down, or step through it manually, action by action. We have found this tool far more useful than we had anticipated. Using the "fast" replay option, we can view a two-hour session in 8-10 minutes. This time compression produces a very clear, albeit intuitive, sense of a writer's strategy. To test and refine these impressions, we use more precise analytic techniques, described below.

Our third tool is the cognitive grammar that is the subject of this report. We describe it in detail in Part 2, below; we give a brief, general description of it here to indicate its relation to the three other tools being discussed.

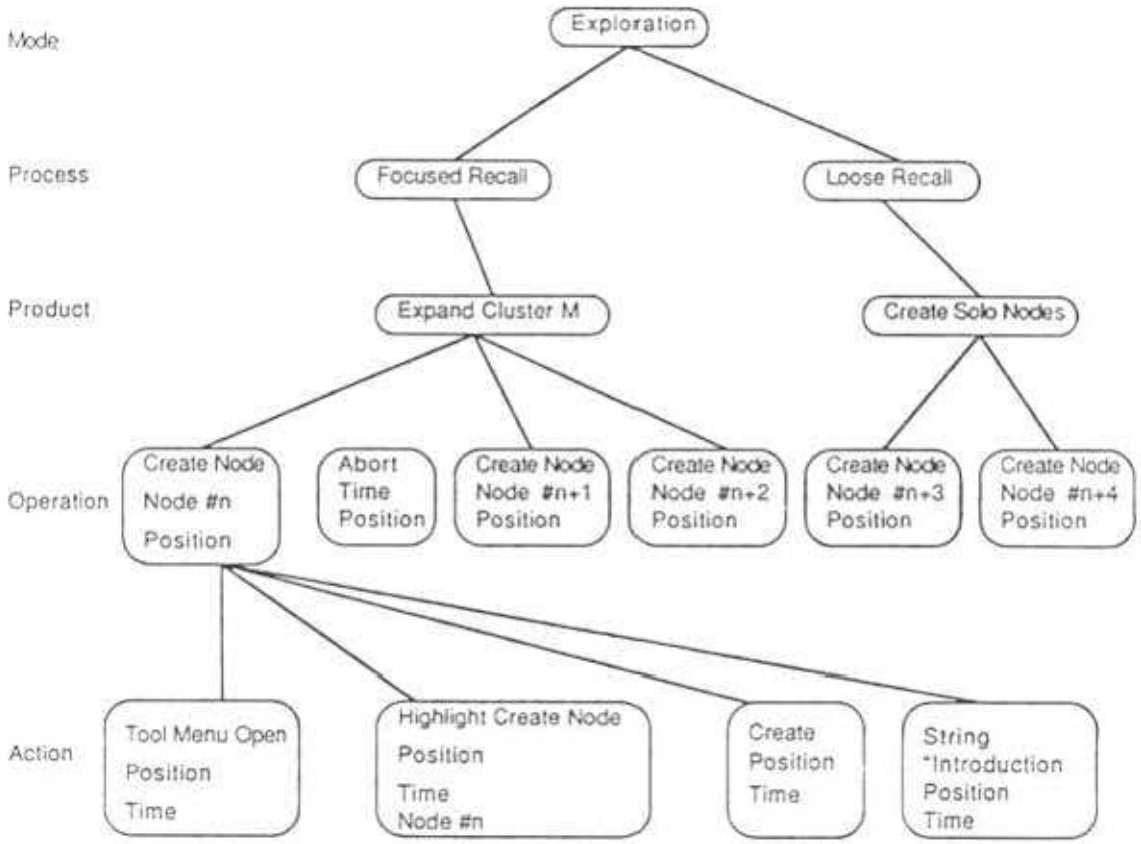
The grammar is defined in terms of five levels of abstraction, as shown in Figure 3. It takes as input a sequence of symbols, each of which represents a single user action. It then maps short sequences of *actions* -- for example, click with the mouse somewhere in the space for network mode, select the "create node" option from the menu, type a label for the node, and then type "return" -- into a system *operation*--create-node. Operations are then mapped into changes to the set of cognitive products; we call this the *delta product* level. This interpretation is highly context sensitive; for example, creating a node can be interpreted as adding to the set of individual concepts or as adding to an existing cluster of related ideas, depending on its spatial proximity to other nodes/concepts. Note the different interpretation for the five nodes that form a cluster, indicated by the shaded circle in Figure 3, and the two, presumably unrelated nodes created in other parts of the space. From sequences of delta product symbols, the grammar infers the *cognitive process* used to produce changes in the conceptual products being constructed. Finally, sequences of process symbols are mapped into *cognitive mode* symbols.

The parse trees produced by the grammar as output provide concrete representations of users' cognitive strategies for the sessions being analyzed. In those structures, we can see users shift from one cognitive mode to another, the sequences of processes being used in each mode, and the particular cognitive products being created or modified by those processes.

These data provide rich material for analysis and interpretation. We are just beginning the fourth methodological step of developing tools to help us analyze and understand this information. As an illustration of both the kind of insights the grammar can provide and of one particular interpretive tool we have developed, consider Figures 4 and 5. They show horizontal slices of the parse trees from sessions by two different subjects. The particular slice is the delta-product level, showing the intermediate products created by the subjects during the session. Down the left side of the diagram are symbols that designate the various types of products

Figure 3:

Cognitive Grammar Parse Tree



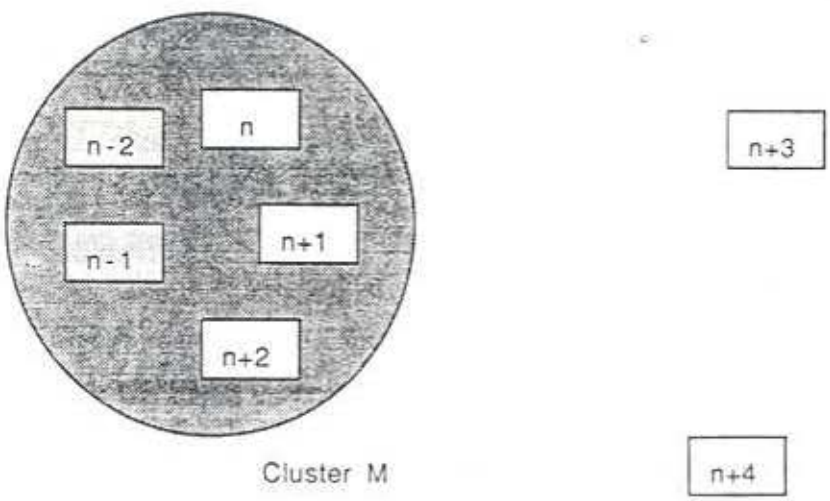


Figure 4:

Protocol Display Example

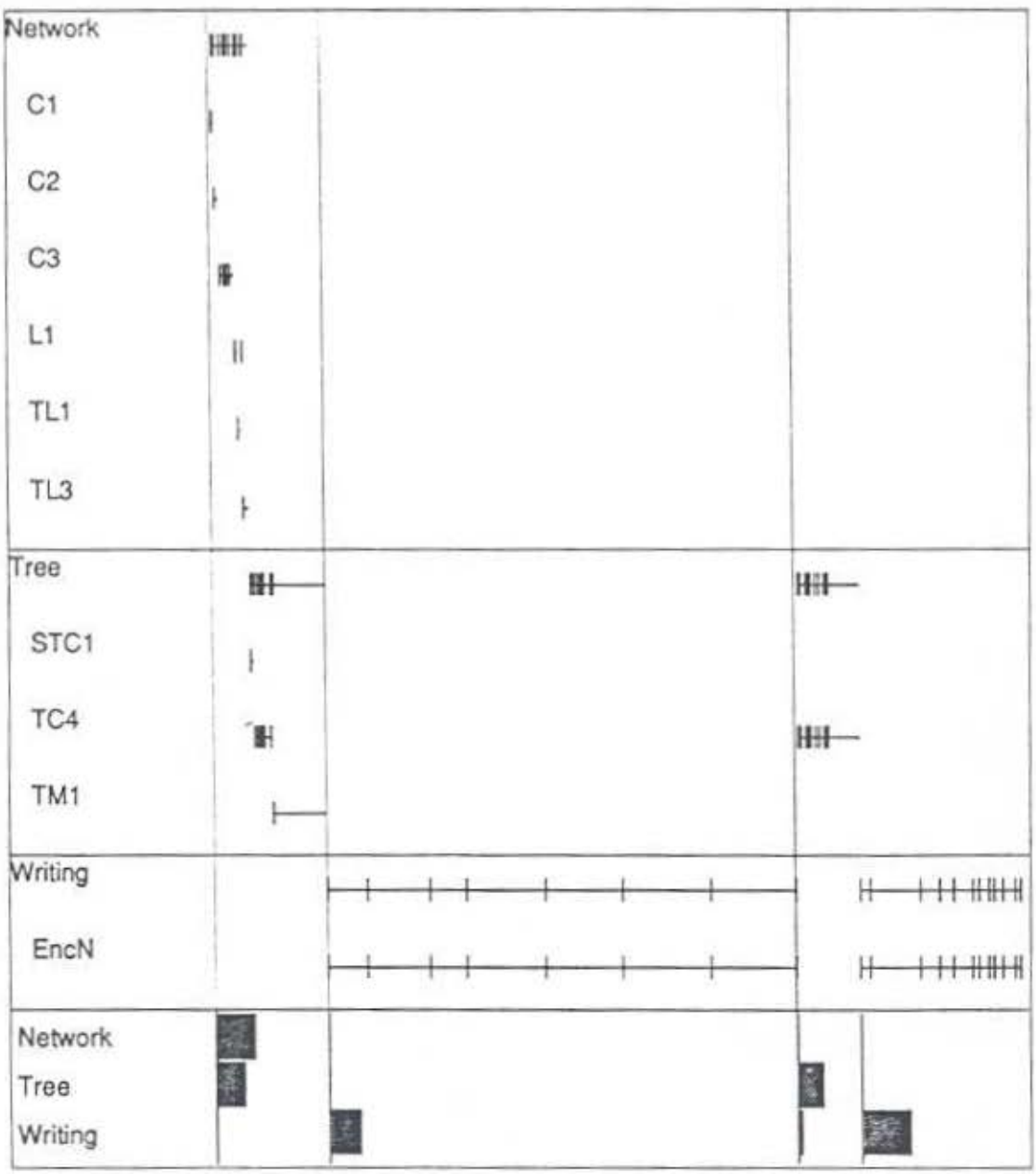
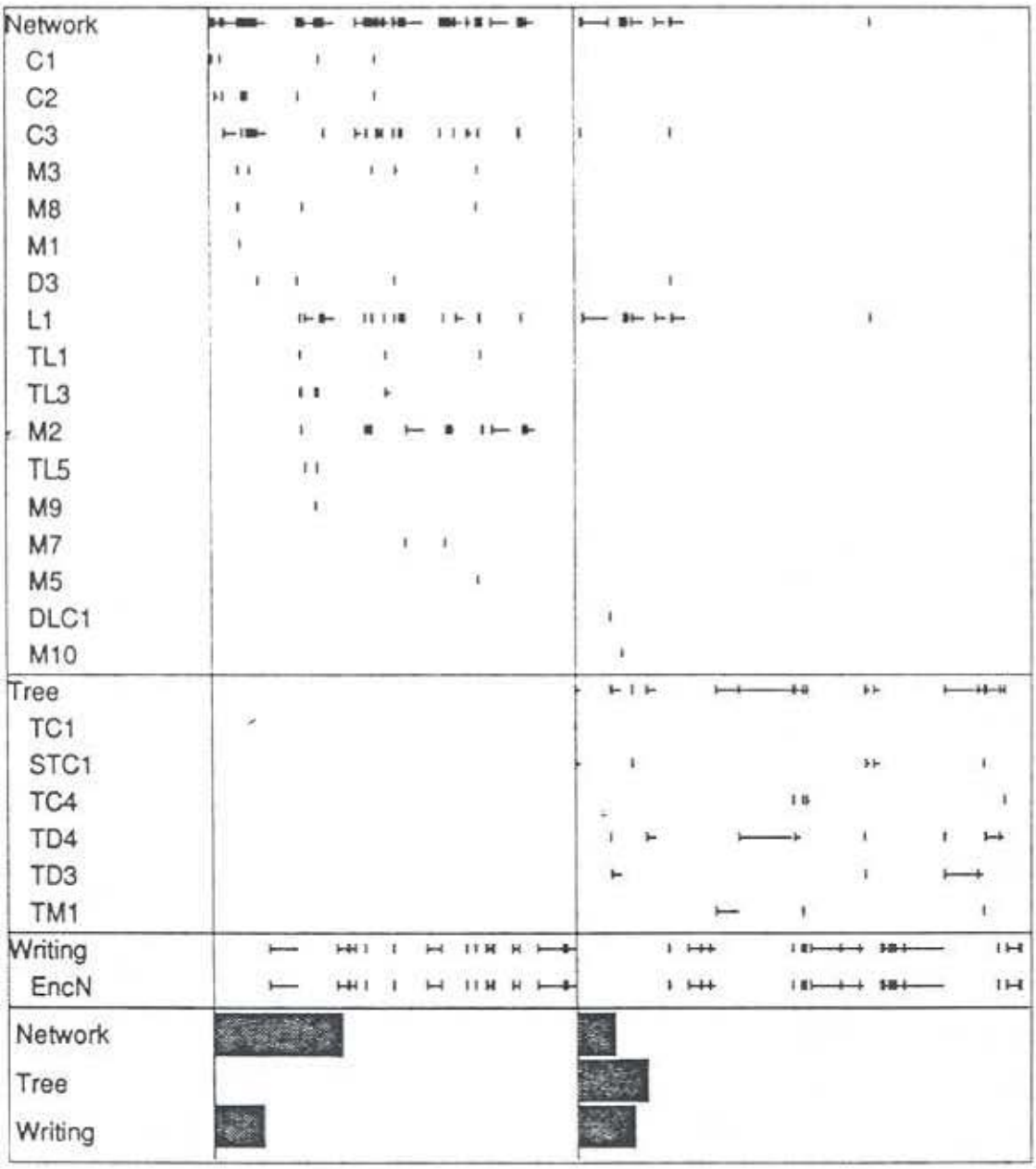


Figure 5:

Protocol Display Example



created, such as isolated nodes, clusters of nodes, additions to the tree being built, etc. Each instance of a product is represented by one of the short vertical lines. The horizontal axis represents time. Thus, we can see each user's strategy as it is reflected in the kinds of products created and when they were produced. In Figure 4, the user follows a nearly perfect "waterfall" or "stages" strategy, doing all of his exploratory thinking first, followed by organizational thinking used to build the top of his tree, followed by writing. He then goes back to the tree, fills out the more detailed levels, and completes his writing. This strategy is markedly different from that of the second subject who constantly moves back and forth between structure operations and writing.

Thus, the grammar is an integral part of a set of tools for recording protocols, replaying them, analyzing the data, and displaying results in a form in which they can be comprehended, interpreted, and compared.

In the preceding discussion, we first described the theoretical basis for the grammar in terms of cognitive modes. Since the grammar characterizes the cognitive strategies of users working with a particular system, we next described the WE 1.0 system to which it applies. Finally, we placed the grammar within a methodological context by discussing how it fits both within a set of tools we have developed and within a set of issues pertaining to the study of human-computer interaction. In Part 2, we give a detailed technical description of the grammar.

Acknowledgments

A number of organizations and individuals have contributed to the work described in this report. We are grateful to the National Science Foundation (Grant # IRI-8519517) and the Army Research Institute (Contract # MDA903-86-C-0345) for their support of various parts of this research. We wish to acknowledge our faculty colleagues, Marcy Lansman, Stephen F. Weiss, and Jay D. Bolter, for contributing ideas and perspectives. We are especially grateful to Oliver Steele for his early work on parts of the grammar and the tracking and replay functions and to John Q. Walker III for his work in developing an alternative grammar for another writing system. Finally, we wish to thank the graduate students who have worked with us on WE 1.0 and on our cognitive experiments using the grammar: Paulette Bush, Yin-Ping Shan, Irene Jenkins, Matt Barkley, and Barry Elledge.

References

Claparede, E. (1933). "La genese de l'hypothese," *Archives de Psychlogie*, 24, 1-155.

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

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

Henry, L. K. (1934). "The role of insight in the analytic thinking of adolescents," *Studies in Education* 9, 65-102.

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

Smith, J. B. & Lansman, M. (1987). *A cognitive basis for a computer writing environment.* Chapel Hill, NC: UNC Department of Computer Science Technical Report # TR87-032.

Smith, J. B. & Smith, C. F. (1987). *A strategic method for writing.* Chapel Hill, NC: UNC Department of Computer Science Technical Report # TR87-024.

Smith, J. B., Weiss, S. F., Ferguson, G. J., Bolter, J. D., Lansman, M., Beard, D. V., (1986). *WE: A writing environment for professionals.* Chapel Hill, NC: Department of Computer Science Technical Report # TR86-025.

Smith, J. B., Weiss, S. F., & Ferguson, G. J., (1987). *A hypertext writing environment and its cognitive basis.* Chapel Hill, NC: Department of Computer Science Technical Report # TR86-033.

Swarts, H. Flower, L. S., & Hayes, J.R. (1984). "Designing protocol studies of the writing process: An introduction," in R. Beach & I. Bridwell (Eds.), New Directions in Composition Research. New York: Guilford Press, pp. 53-71.

Part 2: Technical Description

2.1 Introduction

In Part 2 we describe in detail our Cognitive Grammar for Writing, Version 1.0. We first provide an overview of the architecture of the grammar. Then, in subsequent sections, we discuss in detail each major component.

The grammar can be considered as five separate grammars that work in conjunction with one another or as a single grammar that produces five separate, but logically connected, levels of analysis. Three different levels are shown in Figure 3, above.

The terminal symbols recognized by the grammar are encoded representations of actions performed by the user of the WE 1.0 system. Each symbol denotes a single primitive act, such as selecting a specific menu option or typing a label. These symbols also carry additional information in the form of attributes specific to the particular act, such as the string entered as a label or the location in the background of a window selected by the user by pointing with the mouse. All symbols carry the time at which the action was enacted. The sequence of such symbols for a session constitutes a transcript of that particular session adequate to recreate it. The set of all such sequences is, thus, the "langauge" that is parsed by the grammar; and instances of it are designated the Action Level Transcript in the discussion that follows..

The Action Level Transcript is first parsed to produce what we term the Operation Level Transcript. Each operation respresents a pattern or sequence of several action symbols. Each is the kind of operation one would normally designate in the specifications of a user interface -- such as create_node -- that, in turn, would require several user actions -- such as, select position with mouse, select menu option, type label, type carriage return. This stage of the grammar also deletes errors, cancelled command sequences, etc. Thus, the transcript consists of a sequence of symbols analogous to those for actions. It may be treated as a "language"; however, each symbol on the Operation Level is linked by a symbolic pointer to the sequence of Action Level symbols it characterizes. Consequently, it may also be considered to be a broad hierarchical bush that will eventually be linked by higher levels to form the parse tree for the session as a whole.

The Action and Operation levels of the grammar are closely tied conceptually to the computer system. The third level, which we call the Delta Product Level, shifts the analytic perspective from the system to the mind of the user. In this level, we begin to characterize the user's cognitive behavior based on our interpretation of his or her interaction with the system. This level of the grammar actually contains two steps. In the first, each operation symbol is mapped to a corresponding symbol that interprets the effects of the system operation on the set of intellectual products the user is constructing. These products are well-defined, consisting of structural forms that are important for writing, such as an isolated concept, a cluster of ideas, a relation between two ideas, a primitive subordinate/superordinate relationship, a block of text, etc. The effects of individual changes in the set of cognitive products are then aggregated. Thus, each Delta Product symbol represents the changes produced by one operation or a sequence of operations on one of these conceptual forms. The sequence of these symbols can be considered a language in which each symbol is linked to one or more Operation Level symbols; thus, these links extend the hierarchical bush another level upward.

The fourth level of the grammar is the Cognitive Process Level. Here, each symbol represents the cogntive process inferred to be active in producing one or more changes to the set of cognitive products, identified in the Delta Product Level. Thus, for example, a sequence of additions to a group of nodes in close spatial proximity to one another is interpreted as an instance of sustained "Focused Recall" whereas, if the nodes are "far apart", the process is assumed to be unfocused or free recall. Again, the grammar produces a transcript that can be treated as a language or as a sequence of symbols, each of which can be linked to one or more symbols in a lower level. Thus, the bush grows by one more level of abstraction.

A Cognitive Mode level is inferred from sequences of Cognitive Process symbols. Shifts in Cognitive Mode are strongly suggested when the user shifts from one system mode to another. But the two are not always the same. For example, when a user working in network mode shifts from building small conceptual structures to linking them into a larger hierarchical structure, this may indicate a shift in Cognitive Mode. On the other hand, when the user is building a large hierarchical structure in tree mode and returns to network mode to copy a structural component into the tree, that shift in system mode may not indicate a shift in Cognitive Mode. At present, this portion of the grammar is incomplete. We currently infer shifts in cognitive mode largely from shifts in system mode operations, but we will add rules to infer shifts from context sensitive sequences of cognitive process symbols. Again, the resulting transcript can be viewed as a language, but, again, the symbols at this level are linked to sequences of lower level symbols. Thus, the Mode Level extends the bush one more level.

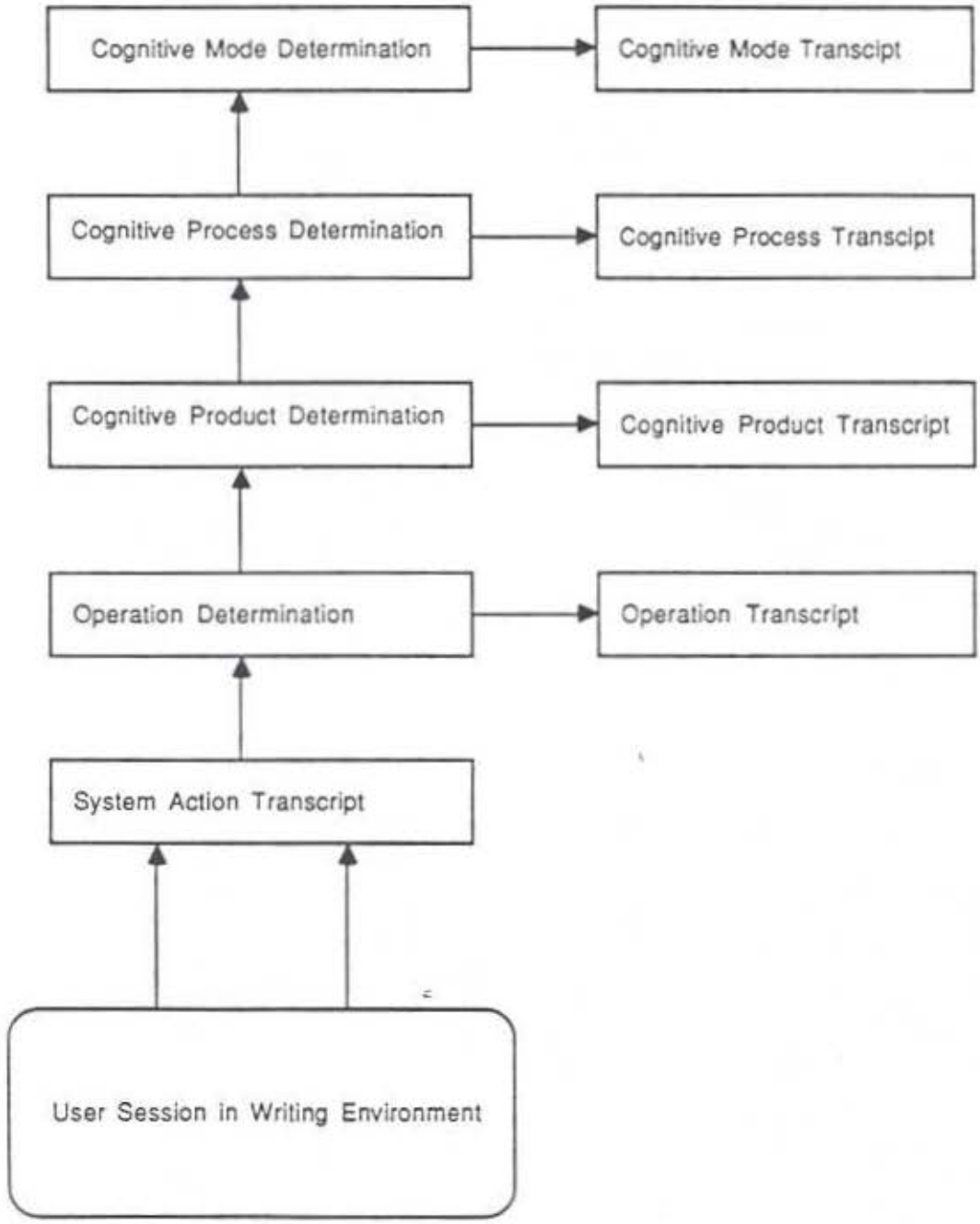
The Cognitive Mode Level symbols can be linked, finally, to a symbol representing the session as a whole. Thus, the separate branches can be joined to form a tree. It depicts the session as a hierarchy of cognitive and system activities in which the user's strategy is manifest as a series of mode shifts, each of which includes a series of shifts in cognitive processes, used to produce changes in the set of intellectual products being constructed, as indicated by short sequences of operations, each of which consists of several system actions.

The grammar is implemented as an OPS-83 program. Figure 6 provides a schematic view of its dataflow. OPS is an expert system shell that provides a programming language based on production rules. Thus, from another perspective, the grammar can be considered as an expert system that simulates the actions of an expert interpreter of the Action Level transcript. At present, the program is run as a "batch" program on the complete transcript after the session has been concluded. A topic for future research is running the grammar concurrently with the session and thereby building a real-time parse structure that represents the user's strategy as his or her strategy evolves. This capability could serve as the basis for an intelligent tutor, more powerful intelligent functions within WE, and functions that might search an external database for material relevant to the structure of ideas being built. But these possibilities must await future versions of the grammar.

In the remainder of Part 2, we first describe, briefly, the general form of the transcription language used at all five levels of the grammar. Following that are descriptions of the grammar for each level. Each discussion includes an introduction, a description of the transcription language symbols for that level, the grammar that maps symbol sequencies from lower levels to symbols at that level, and a sample annotated transcript. The one exception is the Action Level section: it does not include a grammar since it constitutes the level of primitive or terminal symbols.

Figure 6:

Cognitive Grammar Data Flow



2.2 Protocol Language Overview

At each of the five levels of abstraction, our system of analysis provides a description - by way of a transcript - of what occurred during a given session of the writing environment. At each level, the transcript of a session consists of an eight-line header followed by a blank line, followed by a series of one-line event records. Since these transcripts share a common format, this format is outlined below.

Header

The header lists the following items, in the following order:

1) The version of the Writing Environment (WE) which was being used when the transcript was recorded.
2) The date of the transcript.
3) The time at which the session began.
4) The name of the user.
5) The clock at the beginning of the session.
6) The name of the database being used.
7) A boolean indicating whether the database began with an empty workspace.

Event Records

Each line of a transcript following the blank line which follows the header is an event record. At each of the five posited levels, each event record contains two mandatory fields, followed by zero or more additional fields (depending on the level and event-type). Every event record of every transcript conforms to the following template:

Time Event-token {Event parameters}

The first field - Time - indicates the amount of time (in milliseconds) elapsed since the beginning of the session until the beginning of the event being recorded. The second field - Event-token - indicates what event the analysis has posited as having occurred. Finally, certain event-types are always followed by additional fields which are required to complete records which contain event-tokens of the type in question. These "event parameters" are fully documented at each level with their associated "event-types."

2.3 Action Level

Introduction

The Action Level consists of symbols produced by the tracking function in WE 1.0. Each action represents an act performed by the user, such as designating a location in the system mode window, selecting a menu item, typing a label, etc. These symbols also carry additional information in the form of attributes specific to the particular act, such as the string entered as a label or the location in the background of a window selected with the mouse. All symbols carry the time at which the action was enacted. Thus, the sequence of symbols constitutes a machine-recorded protocol for a user during a particular session. Since actions are the primitive or terminal symbols for the grammar, this section does not include a grammar component. It does, however, include a complete description of all menu items included in the WE 1.0 interface as well as every action that causes a line to be recorded in the Action Level Transcript.

2.3.1 Action Level Transcript Language

An action level transcript file consists of a header, followed by a blank line, followed by a sequence of event records. The transcript is a essentially a record of all the actions performed by a user of the Writing Environment (WE) during a session. Each action of the user is represented by a line (or event record) in the transcript. Additional lines may appear in the transcript, which do not correspond to user actions, but which are necessary for replaying the program from the transcript. The possible actions are listed below, with whatever collateral information is collected with the action (i.e. the action's parameters).

Header

The header lists the following items, in the following order:

1) The version of the Writing Environment (WE) which was being used when the transcript was recorded.
2) The date of the transcript.
3) The time at which the session began.
4) The name of the user.
5) The clock at the beginning of the session.
6) The name of the database being used.
7) A boolean indicating whether the database began with an empty workspace.

Action Records

Each line of the transcript is an event record. The event records are grouped into "actions ." We distinguish two types of actions: actions which the user initiates and actions the user takes in response to prompts. Each action begins with an action record possibly followed by one or more support records. The support records either supply additional parameters (Parameter records) or provide information for the benefit of the replay mechanism (Replay Records).

The format for the action records is consistently followed through the five levels of abstraction currently recognized by our method of analysis. That format is simply:

Time Symbol Attributes

where:

Time = The time (in milliseconds) from the beginning of the user session until the beginning of the action (operation, etc.) which that particular line of the transcript purports to record.

Symbol = The symbol which represents the type of action (operation, etc.) which is recorded on the transcript.

Attributes = A list of zero or more attributes which may include 1) the object of the particular action (operation, etc.), 2) the logical relationship of the object of the action to other objects, and/or 3) additional spatio-temporal information about the object.

The various kinds of actions are detailed below. Each action record is presented with all its fields; any additional Parameter or Replay records for this kind of action are also identified if possible. In some cases the support records needed depend on the context; these cases are indicated.

Of prime import to subsequent analysis is the menu action. Most user-initiated actions are begun by highlighting a menu item. In a menu action record, we record whatever menu item was highlighted, along with whatever object is the current agent and whatever mode is current. The possible menu items which may be highlighted are detailed following the discussion of action record types.

A sample annotated transcript concludes the section (terminals are in quotes).

Actions the user initiates include:

-Moving into a new mode; the mode action
time **"mode"** globalX globalY localX localY mode
-Moving the cursor over an agent; the agent action
time **"agent"** globalX globalY localX localY object

-Using a menu to get help for an option - the helpMenu action
time **"helpMenu"** globalX globalY localX localY menuItem

-Raising a menu, whether an option is chosen or not - the menu action

time **menuItem** globalX globalY localX localY mode:object

-Adding text to a node - the encode action
time **"encode"** time globalX globalY localX localY object

-Initiating a roam operation - the roamStart action
time **"roamStart"** globalX globalY

Actions the user takes in response to prompts:

-Entering a string into a prompt box - the string action
time **"string"** globalX globalY string

-Answering a yes/no prompt - the boolean action
time **"boolean"** globalX globalY boolean

-Selecting a point when requested (e.g. when moving a node) - the point action
time **"point"** globalX globalY localX localY

-Selecting the terminal node for a link; the link action
time **"link"** globalX globalY node

-Selecting the position below the selected sibling, to which to paste a node or subtree in tree mode - the below action
time **"below"** globalX globalY localX localY node

-Selecting the position above the selected sibling, to which to paste a node or subtree in tree mode - the above action
time **"above"** globalX globalY localX localY node

-Selecting the position as a child of the selected parent, to which to paste a node or subtree in tree mode - the child action
time **"child"** globalX globalY localX localY node

-Changing the viewing space by resizing or moving the roaming box - the viewing action
time **"view"** globalX globalY left top right bottom

In addition to these actions, there are support records which supply additional information to the parser:

-Indicates that a new node or link is created;
time **"create"** globalX globalY localX localY object

-Indicates that a roam operation was completed
time **"roam"** globalX globalY

-Indicates that an action failed (for whatever reaon)
time **"cancel"** globalX globalY localX localY [object]

Explanation of the parameters:

1) time: The amount of time in milliseconds since the beginning of the session.

2) globalX, globalY:
The x and y coordinates of the event on the screen.

3) localX, localY:
The x and y coordinates of the event within the particular mode screen in which the event transpired.

4) mode: One of several of the possible system modes. The modes are abbreviated as follows:

Edit Mode:	ParaMode1
Control Panel Mode:	CP1Mode
Network Mode:	NetMode1
Tree Mode:	TreeMode1
Text Mode:	ReviseMode1

5) object: Either a node, or an arc (identified by a unique number), or one of the modes listed above.

6) menuItem: A menuItem may be any one of the possible menu items which may be highlighted from any of the several modes. A complete list of these items appears at the end of this section.

7) boolean: Either true or false.

8) string: Used to indicate the first few letters of text used for node names, link names, etc.

9) left, top, right, bottom:
The upper left and lower right coordinates of the current mode.

10) node: A unique node identifier.

Replay Event Records

There are some events which are only present for the benefit of the Replay Manager. Event records which begin with the following words can be ignored:

openSession
firstmode
closeSession
notify
wbase
menuStart
helpMenuStart
menuEnd

MenuItems

Menu	In Transcript
System Control Panel Menus:	
Writing Environment menu:	
Redraw All:	redrawOpt
Default Layout:	defaultOpt
Quit:	finished
Workspace menu:	
Save WS:	saveWS
Change WS:	changeWS
Rename WS:	renameWS
Reset WS:	resetWS
Garbage Collect:	collectGarbage
Delete WS:	deleteWS
Holding Areas menu:	
Show Nodes:	showNodes
Show Trees:	showTrees
Holding Areas submenu:	
Move to Top:	moveToTop
Delete:	throwOut
System menu:	
Screendump:	screendump
Snapshot:	snapshot
Prompt before saving text:	editPromptToggle
Change WS Directory:	changeWSdir
Printing Options:	printMenu
Create WS Directory:	createWSdir
To Smalltalk:	toSmalltalk
Network Mode Menus:	
Control Panel Menus:	
View menu:	
Redraw:	redrawOpt
Resize:	resizeOpt
Switch size:	switchSize
Roam:	roam
Display/print menu:	
Show in text window:	toTextMode
Send to line printer:	toLinePtr
Write as a TeX file:	asTeX

Menu	In Transcript
Background menu:	
Create Node:	createNode
Paste Node:	pasteNode
Paste Tree:	pasteStructure
Show/Hide Links:	toggleLinks
Show/Hide Link Names:	toggleLinkNames
Subtract Tree:	deletePastedStructure
Node menu:	
Move:	moveNode
Link:	addLink
EditName:	editNodeNameS
Edit Text:	editNodeText
Copy Node:	copyNode
Copy Tree:	copyTree
Delete Node:	deleteNode
Link menu:	
Edit Link Name:	rename
Delete Link:	delete
Tree mode menus:	
Cantrol Panel menus:	same as for Network mode
Background menu:	
Paste Node:	pasteNode
Paste Tree:	pasteTree
Create Tree:	createTree
Copy Node:	copyNode
Copy Tree:	copyTree
View Context:	viewContext
Set Node Size:	setDepth
Node menu:	
Add Node: above:	addAbove
Add Node: below:	addBelow
Add Node: child:	addChild
Add Node: Parent:	addParent
Edit Name:	rename
Edit Text:	textEdit
Copy Node:	copyNode
Delete Node:	deleteNode
Subtree menu:	
Move Subtree:	moveSubtree
Show Text for Subtree:	printOutSubtree
Display Subtree:	newCurrRoot
Copy Subtree:	copyOutSubtree
Delete Subtree:	deleteSubtree

Printing Options Menu:

Laser Writer:	specLaserWriter
Line Printer	specLaserWriter
file name:	outfile
auto send	autoPrint

2.3.2 Sample Annotated Transcript

In the following transcript, lines which are comments rather than actual lines of the transcript are preceded by two asterisks.

```
'Writing Environment Transcript version 2.0'
version:     2.0
date:        23 September 1988
time:        1:23:26 pm
user:        jenkins
clock:       27947
database: emptyWS
empty:       true

      40  openSession  492  240    0    0
      80  mode           0    0 -701 -561      ParaMode1
  205420  mode          10  153    9   92      NetMode1

** The subject moves the cursor to network mode

  205480  agent         10  153    9   92          NetMode1
  206180  menuStart    105  184                    ToolHelpedMenu
  206860  createNode   108  187  104  123          NetMode1 Net
  206880  create       108  187  104  123          Nodes   23482710
  210480  string       108  187                    bird

** Node # 2348270 is created in network mode and given the label "bird."
** The node creaton began 206180 milliseconds after the beginning of
** the session.

  211160  agent        122  191  121  130          Nodes   23482710
  211340  agent        290  153  289   92          NetMode1
  211660  menuStart    302  147                    ToolHelpedMenu
  212120  createNode   305  150  301   86          NetMode1 Net
  212140  create       305  150  301   86          Nodes   24006711
   21406  string       305  150                    cat

** A second node with the label "cat" is created.

  215000  menuStart    208  273                    ToolHelpedMenu
  215420  createNode   215  274  207  212          NetMode1 Net
  215440  create       215  274  207  212          Nodes   24336712
  217920  string       216  275                    fish
  219300  menuStart    250  211                    ToolHelpedMenu
  219740  createNode   253  214  249  150          NetMode1 Net
  219780  create       253  214  249  150          Nodes   24770713
  223740  string       253  214                    dog
  224780  agent        305  169  304  108          Nodes   24006711
  225000  agent        399  161  398  100          NetMode1
```

```
225260  menuStart   417  151                 ToolHelpedMenu
225820  createNode  420  154  416   90       NetMode1 Net
225860  create      420  154  416   90       Nodes    25378714
227740  string      420  154                 mouse
228720  agent       379  151  378   90       Nodes    24006711
229480  agent       408  161  407  100       NetMode1
229580  agent       418  163  417  102       Nodes    25378714
230100  agent       397  156  396   95       NetMode1
230140  agent       377  152  376   91       Nodes    24006711
231300  agent       295  197  294  136       NetMode1
231480  agent       283  213  282  152       Nodes    24770713
231800  menuStart   277  221                 ToolHelpedMenu
232580  addLink     281  240  282  152       Nodes    24770713   Net
233840  link        322  170                 Nodes    24006711
234780  create      322  170  282  152       Arcs     26264715
```

** Node # 24770713 is linked to Node # 2400611 in network mode. The arc
** is given the # 26264715 by the system.

```
235460  agent       323  162  322  101       Nodes    24006711
236120  menuStart   331  162                 ToolHelpedMenu
236940  addLink     334  181  322  101       Nodes    24006711   Net
238260  link        438  170                 Nodes    25378714
238340  create      438  170  322  101       Arcs     26622716
239360  agent       417  246  416  185       NetMode1
240400  agent       435  185  434  124       Nodes    25378714
241200  agent       438  191  437  130       NetMode1
242320  menuStart   438  192                 ToolHelpedMenu
242520  createNode  441  195  437  131       NetMode1 Net
242540  create      441  195  437  131       Nodes    27048717
245260  string      441  195                 elephant
245420  agent       428  182  427  121       Nodes    25378714
246280  agent       452  202  451  141       Nodes    27048717
246460  agent       478  228  477  167       NetMode1
246520  agent       478  226  477  165       Nodes    27048717
246920  menuStart   472  212                 ToolHelpedMenu
248900  moveNode    485  213  477  165       Nodes    27048717   Net
250060  point       453  159  452   98
```

** Node # 27048717 is moved in network mode to coordinates 453,159,452,98

```
251100  agent       388  195  387  134       NetMode1
251560  agent       314  193  313  132       Arcs     26264715
251980  agent       410  171  409  110       NetMode1
252080  agent       424  169  423  108       Nodes    25378714
252780  agent       501  162  500  101       Nodes    27048717
253560  agent       390  232  389  171       NetMode1
254020  agent       324  223  323  162       Nodes    24770713
254300  agent       316  199  315  138       Arcs     26264715
254740  agent       326  205  325  144       NetMode1
255600  agent       401  168  400  107       Arcs     26622716
256260  agent       373  152  372   91       Nodes    24006711
256440  agent       369  214  368  153       NetMode1
256620  agent       321  230  320  169       Nodes    24770713
```

```
256820  agent       287  256  286  195        NetMode1
257020  agent       271  278  270  217        Nodes   24336712
257760  agent       95  249   94  188         NetMode1

** The subject moves the cursor around touching on arcs, nodes, and
** network mode itself in the preceding 16 "agent" records.

258360  menuStart    55  277                 ToolHelpedMenu
258940  createNode   58  280   54  216       NetMode1 Net
258980  create       58  280   54  216       Nodes   28688718
260840  string       58  280                 eagle
261780  agent       180  212  179  151        Nodes   23482710
262180  menuStart   158  200                 ToolHelpedMenu
263300  addLink     163  225  179  151        Nodes   23482710   Net
264220  link         75  291                 Nodes   28688718
264300  create       75  291  179  151       Arcs    29218719
265140  agent       180  274  179  213        NetMode1
265260  menuStart   180  274                 ToolHelpedMenu
265820  createNode  183  277  179  213        NetMode1 Net
265840  create      183  277  179  213        Nodes   29376720
267940  string      183  277                 ostrich
268880  agent       149  205  148  144        Nodes   23482710
269340  menuStart   145  193                 ToolHelpedMenu
270020  addLink     148  212  148  144        Nodes   23482710   Net
270960  link        222  298                 Nodes   29376720
271040  create      222  298  148  144        Arcs    29892721
272160  agent       221  298  220  237        Nodes   29376720
272480  menuStart   221  297                 ToolHelpedMenu
273800  moveNode    228  310  220  237        Nodes   29376720   Net
277700  point       196  384  195  323
279180  agent       170  419  169  358        NetMode1
279860  agent        96  291   95  230        Nodes   28688718
280060  agent       100  253   99  192        NetMode1
280340  agent       142  211  141  150        Nodes   23482710
280520  agent       156  179  155  118        NetMode1
281540  menuStart   125   85                 ToolHelpedMenu
281980  createNode  128   88  124   24       NetMode1 Net
282000  create      128   88  124   24       Nodes   30992722
284620  string      128   88                 Animals
285760  agent       134   92  133   31       Nodes   30992722
286440  menuStart   158   94                 ToolHelpedMenu
287520  addLink     177  119  133   31       Nodes   30992722   Net
288820  link        133  193                 Nodes   23482710
288880  create      133  193  133   31       Arcs    31678723
290240  menuStart   169   97                 ToolHelpedMenu
291060  addLink     172  116  133   31       Nodes   30992722   Net
292440  link        226  286                 Nodes   24336712
292520  create      226  286  133   31       Arcs    32040724
294080  menuStart   154  105                 ToolHelpedMenu
295820  addLink     159  126  133   31       Nodes   30992722   Net
301720  link        277  235                 Nodes   30857223   Net
302170  create      277  238  145  123        Arcs    32968725
303560  agent       233  167  232  106        NetMode1
303740  agent       232  167  231  106        Arcs    32968725
```

```
304140  menuStart    232  167                    ToolHelpedMenu
304860  rename       245  177  231  106          Arcs     32968725   Net
306620  string       245  177                    none

** A link is renamed to the null string

306940  agent        245  193  244  132          NetMode1
307720  agent        228  164  227  103          Arcs     32968725
308180  menuStart    228  164                    ToolHelpedMenu
309080  delete       232  183  227  103          Arcs     32968725   Net

** Arc # 32968725 is deleted

310340  agent        229  185  228  124          NetMode1
310620  agent        169  115  168   54          Nodes    30992722
310980  menuStart    161  101                    ToolHelpedMenu
312460  copyTree     184  186  168   54          Nodes    30992722   Net

** The tree with root at node # 30992722 is copied into the holding area

312740  agent        183  188  182  127          Nodes    23482710
312920  agent        161  368  160  307          NetMode1
313260  mode         137  574  136   73          TreeMode1

** The subject moves to tree mode

313300  agent        137  574  136   73          TreeMode1
314100  menuStart    137  574                    ToolHelpedMenu
315440  pasteNode    158  589  136   73          TreeMode1  Tree
317120  notify       158  589

** An attempt is made to paste a node from the holding to tree mode,
** but the copy fails (probably because no node is in the holding
**  area).

318320  pasteTree    170  608  157   88          TreeMode1  Tree

** A tree is successfully pasted into tree mode

328420  menuStart    169  607                    ToolHelpedMenu
329760  agent        324  716  323  215          Subtree  23482710
330020  agent        174  612  173  111          TreeMode1
330220  menuStart    174  612                    ToolHelpedMenu
334420  agent        263  659  262  158          Nodes    28688718
334600  agent        179  715  178  214          Nodes    24336712
335260  agent        100  727   99  226          TreeMode1
335720  agent         43  703   42  202          Nodes    30992722
336700  agent        174  677  173  176          Nodes    23482710
337760  agent        110  711  109  210          Nodes    30992722
337940  agent          4  707    3  206          TreeMode1
338120  agent         16  703   15  202          Nodes    30992722
338400  agent         38  713   37  212          Nodes    30992722
338940  agent         74  685   73  184          TreeMode1
339360  agent         78  686   77  185          Nodes    30992722
```

```
340180  menuStart    71  700                    ToolMenu
342800  textEdit     87  780   77  185          Nodes   30992722  Tree
345480  mode         926 548   225 -13          ParaMode1

** The subject enters into Edit mode with the contents of Node # 30992722
** in the Edit mode buffer.

353880  encode       725 542   210  12          Nodes   30992722
421220  encode       577 581   81   38          Nodes   30992722

** The subject enters text into the Edit mode buffer.

421260  mode         425 565   424  64          TreeMode1
421300  agent        425 565   424  64          TreeMode1
421700  agent        351 658   350  157         Nodes   28688718
422120  menuStart    291 656                    ToolMenu
424920  addBelow     327 682   350  157         Nodes   28688718  Tree
424960  create       327 682   350  157         Nodes   46668738
427900  string       327 682                    eaglet

** A sibling node to node # 28688718 is created and placed below node #
** 28688718 on the screen (in tree mode).

428560  agent        305 683   304  182         Nodes   46668738
429400  menuStart    300 683                    ToolMenu
430220  addBelow     303 702   304  182         Nodes   46668738  Tree
430240  create       303 702   304  182         Nodes   47198740
436240  string       305 701                    parrot
436960  agent        305 699   304  198         Nodes   47198740
437500  agent        281 669   280  168         Nodes   46668738
437920  menuStart    279 667                    ToolMenu
439500  moveNode     322 792   280  168         Nodes   46668738  Tree
441540  child        330 627   280  168         Nodes   28688718

** Node # 46668738 is moved to become a child of node # 28688718 in tree
** mode.

443020  agent        343 683   342  182         Nodes   47198740
443180  agent        243 715   242  214         Subtree 23482710
443520  agent        235 751   234  250         TreeMode1
443840  agent        177 723   176  222         Nodes   24336712
444140  agent        167 679   166  178         Nodes   23482710
444320  menuStart    167 677                    ToolMenu
445880  cancel       66  807   166  178         Nodes   23482710  Tree
445980  agent        92  795   91   294         TreeMode1
446340  agent        134 727   133  226         Nodes   23449280
447830  addAbove     203 721   133  226         Nodes   24336712  Tree
449300  create       203 721   133  226         Nodes   49104743
453780  string       203 721                    Reptile
454860  agent        184 699   183  198         Nodes   49104743
455960  agent        70  708   69   207         Nodes   30992722
456800  menuStart    52  702                    ToolMenu
458280  addChild     84  739   69   207         Nodes   30992722  Tree
458320  create       84  739   69   207         Nodes   50004745
```

```
462120  string      84  739                     Mammal
463640  agent       116  751  115  250          Subtree 30992722
464080  agent       170  747  169  246          Nodes   50004745
464560  menuStart   173  745                   ToolMenu
468220  moveNode    202  872  169  246          Nodes   50004745  Tree
469900  above       166  673  169  246          Nodes   49104743
472220  menuStart   182  677                   ToolMenu
474560  copyNode    194  781  183  182          Nodes   50004745  Tree
```

** Node # 50004745 is copied from tree mode onto the holding stack.

```
474640  agent       194  781  193  280          TreeMode1
474960  agent       199  742  198  241          Nodes   24336712
477000  mode        463  239  462  178          NetMode1
477060  agent       463  239  462  178          NetMode1
477080  menuStart   463  239                   ToolHelpedMenu
478980  pasteNode   497  259  462  178          NetMode1 Net
479460  agent       497  259  496  198          Nodes   50004745
480500  agent       468  336  467  275          NetMode1
481760  mode        100  686   99  185          TreeMode1
481820  agent       100  686   99  185          Nodes   30992722
482160  agent       127  674  126  173          Subtree 30992722
482500  agent       165  670  164  169          Nodes   50004745
482680  menuStart   169  676                   ToolMenu
484340  deleteNode  196  810  164  169          Nodes   50004745  Tree
```

2.4 Operation Level

Introduction

The Operation Level represents the kinds of operations one would normally designate in specifying a user interface -- such as create_node. Each such operation would then be implemented by several user actions -- such as, select position with mouse, select menu option, type label, type carriage return. The grammar for this level maps sequences of Action Level symbols -- normally, three or four -- into individual Operation Level symbols. In the sections that follow, we first discuss the transcription language for this level, then the grammar, and, finally, a sample annotated transcript.

2.4.1 Operation Level Transcript Language

An operation level transcript consists of a header followed by a series of one or more operation records. Any operation a user wishes to perform in WE must be initiated by highlighting a menu item (with one exception). An operation is a grouping of the sequence of actions initiated by the user in highlighting a menu item. Each operation consists of two or more actions. For example the "create node" operation entails opening a menu, highlighting the appropriate menu item, and giving the new node a name. A user may perform a system mode shift operation by simply moving the cursor from one system mode to another (and this is the exception). Each operation record occupies one line of the transcript file.

Header

The header lists the following items, in the following order:

1) The version of the Writing Environment (WE) which was being used when the transcript was recorded.
2) The date of the transcript.
3) The time at which the session began.
4) The name of the user.
5) The clock at the beginning of the session.
6) The name of the database being used.
7) A boolean indicating whether the database began with an empty workspace.

Operation Record

The format for the action records is consistently followed through the five levels of abstraction currently recognized by our method of analysis. That format is simply:

Time Symbol Attributes

where:

Time = The time (in milliseconds) from the beginning of the user session until the beginning of the action (operation, etc.) which that particular line of the transcript purports to record.

Symbol = The symbol which represents the type of action (operation, etc.) which is recorded on the transcript.

Attributes = A list of zero or more attributes which may include 1) the object of the particular action (operation, etc.), 2) the logical relationship of the object of the action to other objects, and/or 3) additional spatio-temporal information about the object.

Thus, preceding the operation identifier of each record, is the time elapsed since the beginning of the session until the time at the end of the action event which triggered the creation of the operation record (labeled "Time"). The operation identifier follows "Time" in the operation record. For certain operations, a list of attributes follows the operation identifier in the transcript.

Below is a list of all possible operation types in operation level transcripts. A short description of the type precedes the template (for the transcript) of that operation. A sample annotated transcript concludes the section.

-Aborting an operation (e.g. when a user attempts to paste a node when the holding area is empty) - the abort operation

Time abort

-Changing the viewing space of a mode by using a roaming operation or changing the viewing space to view the entire tree or a subtree - the context operation

Time context view [tree | view rectangle]

-Copying a node or tree (in tree mode) into the holding area - the copy operation

Time copy object

-Copying a tree into the holding area from network mode - the cptree operation

Time cptree root node

-Deleting an object from the workspace - the delete operation

Time delete object

-Associating text with a node in edit mode - the encode operation (Note: There is no comparable operation for revise mode. This is a design flaw.)

Time encode node

-Hiding the link names - the hidelinknames operation

Time hidelinknames

-Hiding the links- the hidelinks operation

Time hidelinks

-Changing the viewing space of a mode by resizing or switching the size of the mode - the layout operation

Time layout screen rectangle

-Changing modes - the mode operation
Time mode mode

-Moving a node to a new position - the move operation
Time move node position

-Creating a new link - the newLink operation
Time newLink link From node To node

-Creating a new node - the newNode operation
Time newNode node position string

-Pasting a node from the holding area into a mode - the paste operation
Time paste object position

-Renaming a node or link - the rename operation
Time rename object string

-Reading a saved file into the workspace - the session operation
Time session

-Displaying the link names - the showlinknames operation
Time showlinknames

-Displaying the links - the showlinks operation
Time showlinks

-Performing one of several possible system operations (e.g. showing Tex output, saving the workspace, etc.) - the system operation
Time system

-Viewing the text associated with a node in edit mode, or associated with a tree or subtree in text mode - the view operation
Time view object mode

The attributes of the various operations are:

mode	Either CP (control panel), Tree (tree), Net (network), Para (edit), or Revise (text) modes.
node	A node followed by a number (in parenthesis) which identifies it.
link	A link followed by a number (in parenthesis) which identifies it.
object	Either a node, arc, or tree. If the object is a node or link, it is identified by a number in parenthesis. If the object is a tree, it is identified by the number of the node which is

	its root.
tree	A tree identified by its root node.
position	Either x, y coordinates (a point) if in network mode, or tree positon if in tree mode. If tree position, then either "parent", "above", "below", or "child" will be given, followed by a node and its identifying number. Thus if node 8 of a tree is moved, a possible line of the transcript might be: "1020399 756 move node(8) (above, node(6))." "Above" and "below" indicate the node becomes the sibling (above or below it on the screen) of the given node, "child" that it becomes the child, and "parent" that it becomes the parent.
screen rectangle	The mode coordinates of the new view port after a roam operation.
view rectangle	The screen coordinates of the new mode location after a resize or shift size operation.
From node, To node	The nodes which a link links. The To node is the node to which the link points.
string	A string.

2.4.2 Operation Level Grammar

The following attribute grammar clarifies the origin in the action level transcript of every item of information output in the operation level transcript. Following each operation record type is a parenthesized list of attributes which will appear with that operation type in the operation level transcript. Within the parenthesized list may appear literals (which are enclosed in quotes) of the operation level record. Following this parenthesized list (i.e. after the '::=' symbol), is a list of the action level record type(s) (with their associated attributes) which cause the operation record type in question to be generated.

Operation Transcript ::= Operation Record+

Operation Record ::=
- Abort Record |
- Context Record |
- Copy Record |
- Copy Tree Record |
- Delete Record |
- Encode Record |
- Hide LinksRecord |
- Hide Link Names Record |
- Layout Record |
- Mode Record |
- Move Record |
- NewLink Record |
- NewNode Record |
- Paste Record |
- Rename Record |
- Session Record |
- Show Links Record |
- Show Link Names Record |
- System Record |
- View Record |

Abort Record (time) ::=

addLink(time) [cancel | link] |

addAbove(time) [cancel | ø] |

addBelow(time) [cancel | ø] |

addChild(time) [cancel | ø] |

addParent(time) notify |

cancel (time) |

changeWS (time) [boolean (false) | string] |

```
                copyTree (time)  |
                createTree (time) |
                createWSdir (time) string |
                deletePastedStructure (time) |
                finished (time)boolean (false) |
                moveNode (time) cancel |
                moveSubtree (time) cancel |
                pasteNode (time) [cancel | ø] |
                pasteTree (time) [cancel | ø] |
                printMenu (time) cancel
Context Record (time, "tree(", subtree,")" ) ::=
                newCurrRoot (time, subtree)
Context Record (time, "tree (0)" ) ::=
                viewContext (time)
Context Record (time, rectangle) ::=
                roam (time) view (rectangle)
Copy Record (time, "node(", node,")") ::=
                copyNode (time, node)
Copy Record (time, "tree(", node,")") ::=
                copyOutSubtree (time, node)
Copy Tree Record (time, node) ::=
                copyTree (time, node)
Delete Record (time, node) ::=
                deletePastedStructure (time, node) |
                deleteNode (time, node)
Delete Record (time, arc) ::=
                delete (time, arc)
```

Operation Level Grammar

Delete Record (time, subtree) ::=

deleteSubtree (time, subtree)

Encode Record (time, node) ::=

encode (time, node)

HideLinks Record (time) ::=

hideLinks (time)

HideLinkNames Record (time) ::=

hideLinkNames (time)

Layout Record (time, rectangle) ::=

resizeOpt (time) view (rectangle) |

switchSize (time) view (rectangle)

Mode Record (time, modetype) ::=

mode (time, modetype)

Move Record (time, node, localx, localy) ::=

moveNode (time, node) point (localx, localy) |

Move Record (time, $node_1$, "(above", $node_2$,")") ::=

moveNode (time, $node_1$) above ($node_2$)

Move Record (time, $node_1$, "(below", $node_2$,")") ::=

moveNode (time, $node_1$) below ($node_2$)

Move Record (time, $node_1$, "(parent", $node_2$,")") ::=

moveNode (time, $node_1$) child ($node_2$)

Move Record (time, "tree(",$node_1$, ")", "(above", $node_2$,")") ::=

moveSubtree (time, $node_1$) above ($node_2$)

Move Record (time, "tree(", $node_1$, ")", "(below", $node_2$,")") ::=

moveSubtree (time, $node_1$) below ($node_2$)

NewLink Record (time, arc, $node_1$, $node_2$) ::=

addLink (time, $node_1$) link ($node_2$) create (arc)

NewNode Record (time, $node_2$, "(above", $node_1$,")" , string) ::=

addAbove (time, $node_1$) create ($node_2$) string (string-token)

NewNode Record (time, $node_2$, "(below", $node_1$,")" , string) ::=

addBelow (time, $node_1$) create ($node_2$) string (string-token)

NewNode Record (time, $node_2$, "(child", $node_1$,")" , string) ::=

addChild (time, $node_1$) create ($node_2$) string (string-token)

NewNode Record (time, $node_2$, "(parent", $node_1$,")" , string) ::=

addParent (time, $node_1$) create ($node_2$) string (string-token)

NewNode Record (time, node, "(root)", string) ::=

createtree (time) create (node) string (string-token)

NewNode Record (time, node, localx, localy, string) ::=

createNode (time, localx, localy) create (node) string (string-token)

Paste Record (time, node, localx, localy) ::=

pasteNode (time, localx, localy, node)

Paste Record (time, "node(0)", "(above ",node,")") ::=

pasteNode (time) above (node)

Paste Record (time, "node(0)", "(below ",node,")") ::=

pasteNode (time) below (node)

Paste Record (time, "node(0)", "(parent ",node,")") ::=

pasteNode (time) child (node)

Paste Record (time, "node(0), root") ::=

pasteNode (time)

Paste Record (time, "tree(0)", "(above ",node,")") ::=

pasteTree (time) above (node)

Paste Record (time, "tree(0)", "(below ",node,")") ::=

pasteTree (time) below (node)

```
Paste Record (time, "node(0)",  "(parent ",node,")") ::=
        pasteTree (time) child (node)
Paste Record (time, "tree(0), root") ::=
        pastetree (time)
Rename Record (time, node, string) ::=
        editNodeName  (time, node) string (string-token)
Rename Record (time, node, string) ::=
        rename  (time,  node) string (string-token)
Rename Record (time,  arc, string) ::=
        rename  (time,  arc) string (string-token)
Session Record (time) ::=
        finished  (time) boolean (true) |
        changeWS  (time)  |
        changeWS (time)  boolean (true) string  |
        changeWSdir (time)  boolean (true) |
        resetWS (time) boolean (true) |
        toSmalltalk (time) boolean (true)

Show Links Record (time) ::=
        showLinks (time)
Show Link NamesRecord (time) ::=
        showLinkNames (time)
System Record (time)::=
        asTeX  (time) |
        suspend (time) |
        collectGarbage (time)  |
        deleteWS (time) |
        printMenu (time) specLaserWriter |
```

printMenu (time) specLinePrinter |

printMenu (time) outFile string |

printMenu (time) toggleAutoPrint |

redrawOpt (time) |

renameWS (time) string |

saveWS (time) string |

screenDump (time) |

throwOut (time)

View Record (time, node, mode) ::=

editNodetext (time, node, mode) |

printOutSubtree (time, node, mode) |

printOutLine (time, node, mode) |

textEdit (time, node, mode) |

toTextMode (time, node, mode)

2.4.3 Sample Annotated Transcript - Operation Level

The following is an annotated operation level transcript. Annotations follow lines (or groups of lines) which they annotate, and are preceded by two asterisks.

```
'Writing Environment Transcript version 2.0'
version:     2.0
date:        23 September 1988
time:        1:23:26 pm
user:        jenkins
clock:       27947
database: emptyWS
empty:       true

80          40          mode       Para
205420      205340      mode       Net

** The subject moves to net mode.

206860      680         newNode    node(1)     (104,123) "bird"

** A first node is created and given the label "bird." The
** creation began 206860 milliseconds after the beginning of the session,
** and occured at x,y coordinates 104, 123.

212120      460         newNode    node(2)     (301,86)   "cat"
215420      420         newNode    node(3)     (207,212)  "fish"
219740      440         newNode    node(4)     (249,150)  "dog"
225820      560         newNode    node(5)     (416,90)   "mouse"
232580      780         newLink    link(1)     node(4)    node(2)

** A new link (link #1) is created linking nodes # 4 and 2.

236940      820         newLink    link(2)     node(2)    node(5)
242520      200         newNode    node(6)     (437,131)  "elephant"
248900      1980        move       node(6)     (452,98)

** Node 6 is moved to new x,y coordinates 452, 98.

258940      580         newNode    node(7)     (54,216)   "eagle"
263300      1120        newLink    link(3)     node(1)    node(7)
265820      560         newNode    node(8)     (179,213)  "ostrich"
270020      680         newLink    link(4)     node(1)    node(8)
273800      1320        move       node(8)     (195,323)
281980      440         newNode    node(9)     (124,24)   "Animals"
287520      1080        newLink    link(5)     node(9)    node(1)
291060      820         newLink    link(6)     node(9)    node(3)
295820      1740        newLink    link(7)     node(9)    node(4)
304860      720         rename     link(7)     "none"
309080      900         delete     link(7)
```

```
** Link # 7 is deleted.

312460      1480      cptree      node(9)

** The tree with root # 9 is copied from network node into the
** holding area.

313260      340       mode        Tree

** The subject shifts to tree mode.

315440      1340      abort

** Some action is aborted.

318480      1200      paste       tree(0)     (root)

** The tree on top of the holding stack area is pasted into tree mode.

342960      2620      view        node(9)   Tree
345640      2680      mode        Para

** The subject moves into Edit mode with the contents of node # 9 in the
buffer.

354040      8400      encode      node(9)
421380      67340     encode      node(9)

** Text is associated with node # 9.

421420      40        mode        Tree
425080      2800      newNode     node(10)    (below,node(7))    "eaglet"

** A new node (node # 10) is created as a sibling node to node # 7,
** and placed below node # 7 on the screen.

430380      820       newNode     node(11)    (below,node(10))   "parrot"
439660      1580      move        node(10)    (parent,node(7))
446040      1560      abort
449440      2480      newNode     node(12)    (above,node(3))    "Reptile"
458440      1480      newNode     node(13)    (child,node(9))    "Mammal"
468380      3660      move        node(13)    (above,node(12))
474720      2340      copy        node(13)

** Node # 13 is copied into the holding area from tree mode.

477160      2040      mode        Net
479140      1900      paste       node(0)     (462,178)
481920      1260      mode        Tree
484500      1660      delete      node(13)
```

2.5 Δ-Product Level

Introduction

The Δ-Product Level shifts the perspective to the mind of the user. In this level, we begin to infer the user's cognitive behavior based on his or her interaction with the system. This level of the grammar actually contains two steps. In the first, each operation symbol is mapped to a correspondiong symbol that interprets the effects of the system operation on the set of intellectual products the user is constructing. These products are well-defined, consisting of structural forms that are important for writing, such as an isolated concept, a cluster of ideas, a relation between two ideas, a primitive subordinate/superordinate relationship, a block of text, etc. The effects of individual changes in the set of cognitive products are then aggregated. Thus, each Δ-Product symbol represents the changes produced by one operation or a sequence of operations on one of these conceptual forms. In the sections that follow, we first discuss the transcription language for this level, then the grammar, and, finally, a sample annotated transcript.

2.5.1 Δ-Product Level Transcript Language

A Δ-product level transcript is a record of transformations in the population of the abstract entities recognized as cognitively significant. These abstract entities are of five types: 1) nodes 2) links 3) trees (defined as any noncyclic linked structure with at least two links and a root in network mode or any structure in tree mode) 4) networks (defined as any linked structure with at least two links and not a tree) and 5) clusters (defined as any group of at least two nodes within some constant distance of one another). A Δ-product level transcript is preceded by a header giving information identical to that at earlier levels.

Not every cognitively significant action (or operation) results in the creation of a Δ-product record; only those actions (or operations) which affect the population of abstract entities cause a Δ-product record to be created. Thus, although we may (and do) regard mode shifts as cognitively significant, the cognitive significance of the shift will be registered at some other level.

Header

The header lists the following items, in the following order:

1) The version of the Writing Environment (WE) which was being used when the transcript was recorded.
2) The date of the transcript.
3) The time at which the session began.
4) The name of the user.
5) The clock at the beginning of the session.
6) The name of the database being used.
7) A boolean indicating whether the database began with an empty workspace.

Δ-Product Record

The format for the Δ-product records is consistent with that followed at other levels. That format is simply:

Time Symbol Attributes

where:

Time = The time (in milliseconds) from the beginning of the user session until the beginning of the action (operation, etc.) which that particular line of the transcript purports to record.

Symbol = The symbol which represents the type of action (operation, etc.) which is recorded on the transcript.

Attributes = A parenthesized list headed by the mode of the transformation and followed by the entities affected by the transformation.

Thus, preceding the Δ-product identifier of each record, is the time elapsed since the beginning of the session until the time at the end of the action event which triggered the creation of the operation record (labeled "Time"). The Δ-product identifier follows "Time" in the operation record. For certain operations, a list of attributes follows the operation identifier in the transcript.

The attribute list following the Δ-product symbol is parenthesized. Every Δ-product record has as an attribute the mode in which the transformation occurred. Following the mode in the attribute list, is a list of products altered in the transformation, both before and after the transformation. The list of altered products before the transformation is separated from the list of altered products after the transformation by the symbol '->'. If the list of products before or after the transformation is nil, then 'nil' is written on the transcript. Thus a sample line from a Δ-product transcript might be:

```
1233211     M3          (Net:  N:3  -> C4)
```

indicating that 1233211 milliseconds after the beginning of the session, the Δ-product transformation 'M3' began in network mode altering nodes 3 and cluster 4. The record

```
4325433     C1          (Net:  nil -> N:4)
```

indicates that 4325433 milliseconds after the beginning of the session, the Δ-product transformation 'C1' (a singleton node creation) began in network mode with no products involved prior to the transformation and with Node 4 involved after the transformation.

Frequently a product might appear both before and after a transformation. For example if a node is moved within a cluster, the cluster undergoes a transformation in the position of its elements. In such a case, the cluster will only be cited in the after transformation list. On the other hand, moved and deleted nodes always appear on the list before transformation, while created nodes always appear after transformation. The Δ-product transcript should be read with the Δ-product rule documentation at hand.

Listed below are the Δ-product symbols with a short description of each.

ΔProduct-Type	Description
M1	Move singleton node to empty space
M2	Move node within cluster
M3	Move node from cluster to empty space
M4	Move node empty space to cluster
M5	Move node outside of cluster creating new cluster
M6	Move node destroying cluster into existing cluster
M7	Move node from existing cluster to existing cluster
M8	Move node from cluster destroying cluster
M9	Move node from empty space creating cluster
M10	Move node destroying cluster creating another
C1	Node creation in empty space
D1	Node deletion in empty space
C2	Node creation creating cluster
D2	Node deletion destroying cluster
C3	Node creation within existing cluster
D3	Node deletion with existing cluster
C4	Node creation destroying a cluster but creating another
D4	Node deletion destroying a cluster but creating others
L1	Link node to another node
UL1	Unlink node
DL1	Delete linked node
DL2	Delete linked node within cluster
DLC1	Delete linked node destroying cluster
TC1	Node creation tree mode creating tree
TD1	Node deletion destroying tree
TC2	Node creation tree mode new tree root
TD2	Node deletion net mode destroying tree root (but not tree)
TC3	Node creation tree mode new subtree
TD3	Node deletion tree mode shrinking subtree
TC4	Node creation tree mode new leaf
TD4	Node deletion tree mode of leaf
STC1	Subtree deletion in tree mode
STD1	Subtree addition in tree mode
TPC1	Tree creation by pasting in tree mode
TM1	Node movement in tree mode
TL1	Link node (net mode) creating tree
TL2	Unlink node (net mode) destroying tree
TL3	Link node (net mode) two trees become one
TL4	Unlink node destroying one tree creating two others
TL5	Link node (net mode) new root to tree
TL6	Unlink node net mode shrinking tree by deleting old root
TL7	New Link which destroys a tree (by (e.g.) creating a cycle)
EncN	Associating text with a node in edit mode

2.5.2 Δ-Product Level Grammar

The grammar of the Δ-product level differs markedly from the operation level grammar. The operation level grammar was a string grammar which formally described the context-free parse of action level strings. In the Δ-product grammar, we take into account certain "cognitive entities" or "cognitive products" as well operation level event records in determining whether a certain Δ-product rule type applies.

Hence the domain of a Δ-product level parse is the operation level transcript as well as a well-defined conceptual space kept resident in the computer's memory, which purports to represent the conceptual space (to some degree) of a user of WE at a particular point in time.

Currently five types of entities are recognized as cognitively significant. The creation, modification, and deletion of these entities serves as the basis for grammatical analysis at the cognitive product level, which in turn serves as input for the analysis at the cognitive process and cognitive mode levels. The five types of entities and their informal definition are as follows:

1) **singleton node**: An atomic rectangular entity of network and tree modes which is the building block of the system. Text may be associated with nodes in either Edit or Revise mode.

2) **relation**: A directed arc linking two nodes. Relations are linked structures not considered networks or trees.

3) **cluster**: Any group of two or more nodes in network mode in which the top left corner of each node of the group is separated by no more than some constant distance from the top left corner of some other node within the group. Currently this constant distance is set at 145 pixels. Thus (e.g.) in the current implementation, two nodes separated by 145 pixels (or less) form a cluster, as will twenty nodes side-by-side in which every node is within 145 pixels of some other node of the group. Clusters may only be created in network mode.

4) **tree**: Any linked structure of two or more links in network mode which contains a unique root node (a node with no parent), and in which every other node of the structure has one and only one parent. In tree mode, every structure (including a structure with only one node) is defined as a tree of one or more nodes.

5) **network**: Any linked structure of two or more links in network mode which is not a tree. Networks exist only in network mode.

More formally we define the cognitive products in the following manner:
Cluster:
1) Partition the node set, N, with the transitive closure of the dist ≤ minimum-cluster-distance relation.
2) Any such partition, c, such that |c| ≥ minimum_cluster_distance is a cluster.

Singleton Node:
Any node not in a cluster is referred to as a singleton node.

Tree, Network, Relation:
1) Partition the node set using the transitive closure of the is_linked_to relation L ($n_1 L n_2$ <-> there is a link L from n_1 to n_2).
2) Define a path of length K as some node sequence $n_1, n_2, \ldots, n_k$ such that $n_i L n_{n+1}$ $1 \leq n \leq k$ and a cycle as a path of length > 1 such that $n_1 = n_k$.
3) Any partition s such that i) $|s| \geq$ minimum_tree_size and ii) S contains no cycles is a tree.
4) Any partition s such that i) $|s| \geq$ minimum_network_size and ii) S is not a tree is a network.
5) All other partitions are relations.
6) Nodes with no links are not classified.

With the current state of these cognitive products (their number and composition) in memory, after each operation the parser attempts to determine whether a transformation of a well-defined type has occurred in the "conceptual space" of the system.

For each rule of the Δ-product grammar, 4 items are given:

1) A description of what the rule is supposed to capture;
2) A description of the change(s) in the cognitive product population effected by the transformation in question;
3) A horn-clause like formal rule describing the transformation, and
4) a graphic image of the transormation.

For 2) (the population description), a number of abbreviations are used. They are:

C: cluster
N: singleton node
L: link
W: network
S: structure
-->: the transformation in question

Every population description is prefixed with either a "C" (for cluster) or "S" (for structure), depending on whether the transformation affects the cluster population or the structure population. Of course certain transformation may affect both the structure and cluster population; for these transformations, the population description will have 2 entries: one entry for the transformation in cluster population and one for transformation in structure population.

A sample population description is: "(C: <C> --> <C,N>)." From this description we can tell that as regards the cluster population, the transformation is from a cluster to a cluster and a singleton node. Or we might see: "(S: <N,T> --> <T>)." From this description we can deduce that the structure population has been modified in the following manner: Prior to the transformation we had a singleton node and a tree; following the transformation we only had a tree. This example indicates the limitation of this sort of description various operations could lead to this transformation in the cognitive product population.

The horn-clause like ("like" since (e.g.) skolem functions have not been used) grammar description has the following form:

operation-type(variable)(Predicates 1 .. N -> Predicates 1..M)

where the "operation-type" refers to one of the operation event-types at the operation level (e.g. "newNode", or "move"), the variable ranges over the node, link, or tree which the operation acted on, the "->" is the transformation symbol, and the "Predicates" are one or more of the following:

Node(x)	: x is a Node
Link(x)	: x is a Link
Relation(x)	: x is a Relation
Tree(x)	: x is a Tree
Cluster(x)	: x is a Cluster
Network(x)	: x is a Network
IsIn(x,y)	: x is in y
InCluster(x)	: x is in a Cluster
Root(x,y)	: x is the root of y
Interior(x,y)	: x is in the interior of y
Leaf(x,y)	: x is a leaf of y
Source(x,y)	: x is the source of y
Destination(x,y)	: x is the destination of y
Create(x)	: x is created
Delete(x)	: x is deleted

For example the rule for M1 is: move(x)(Node(x), ~(InCluster(x)) ->
~(InCluster(x))

From this rule we learn that if the operation was a "move" operation, and a node was moved, and prior to the operation the node was not in a cluster, and after the operation the node was still not in a cluster, then we have an M1. Or if we see the rule for TD4:

delete(x)(Node(x), Tree(y), IsIn(x,y), Leaf(x,y) -> Delete(x))

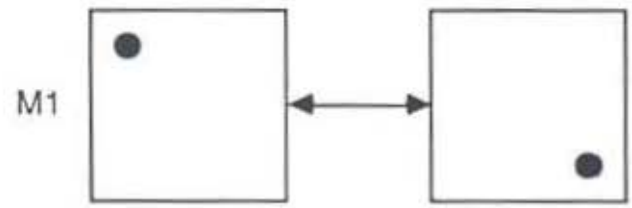
we know that if we have a "delete" of some node x, and x is the leaf of some tree y prior to the transformation, and the transformation entails the deletion of node x, then we have an instance of rule TD4.

Rule M1

Description: A move of a node in empty space, affecting neither clusters nor networks.

Population: (C: <N> --> <N>)

M1 :- move(x)(Node(x),~(InCluster(x)) -> ~(InCluster(x)))

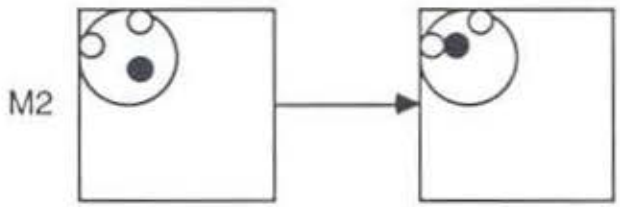


Rule M2

Description: A move of a node within a cluster.

Population: (C: <C>--> <C>)

M2 :- move(x)(Node(x),Cluster(y),IsIn(x,y) -> Cluster(z), IsIn(x,z), (z=y))



Rules M3 & M4

Rule M3

Description: A move of a node from within a cluster to empty space.

Population: (C: <C> --> <C,N>)

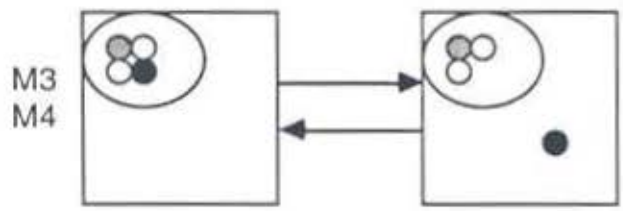
M3 :- move(x)(Node(x), Cluster(y), IsIn(x,y) -> ~(IsIn(x,y)))

rule M4

Description: A move of a node from empty space to a cluster

Population: (C: <C,N> --> <C>)

M4 :- move(x)(Node(x), Cluster(y), ~(InCluster(x)) -> IsIn(x,y))



Rules M5 & M6

Rule M5

Description: A move of a node from a cluster creating a new cluster

Population: (C: <C1,N> --> <C1,C2>)

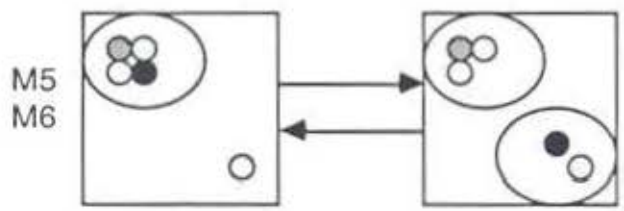
M5 :- move(x)(Cluster(y), Node(x), IsIn(x,y) -> Cluster(z), ~(z=y),
IsIn(x,z), Create(z))

rule M6

Description: A move of a node from a cluster to another cluster
destroying the first cluster

Population: (<C1,C2> --> <C1,N>)

M6 :- move(x)(Cluster(y), Node(x), IsIn(x,y) -> Cluster(z), IsIn(x,z),
~(z=y), Delete(y))

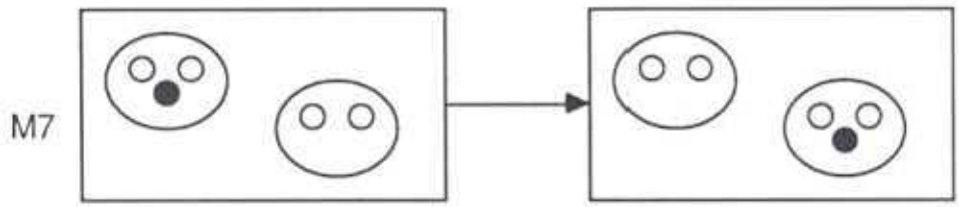


Rule M7

Description: A move from one cluster to another cluster; cluster status unchanged

Population: (C: <C1,C2> --> <C1,C2>)

M7 :- move(x)(Cluster(y), Node(x), IsIn(x,y) -> Cluster(z), IsIn(x,z), ~(y=z))



Rules M8 &M9

Rule M8

Description: A move of a node from empty space to another singleton creating a cluster

Population: (C: <C> --> <N1,N2>)

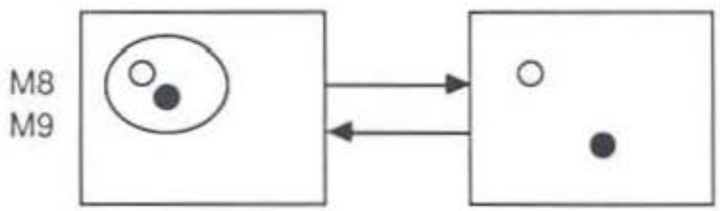
M8 :- move(x)(Cluster(y), Node(x), IsIn(x,y) -> ~(InCluster(x), Delete(y))

Rule M9

Description: A move of a node from a cluster destroying the cluster leaving nodes in empty space

Population: (C: <N1,N2> -> <C>)

M9 :- move(x)(Node(x), ~(InCluster(x)) -> Cluster(y), IsIn(x,y), Create(y))

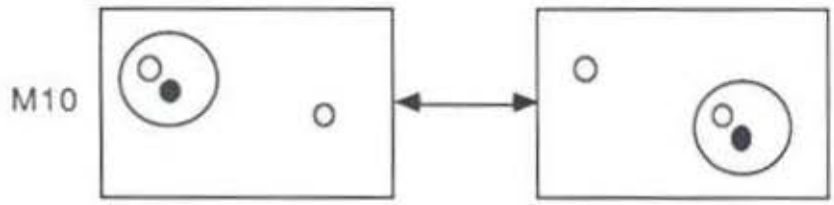


Rule M10

Description: Move of a node from one cluster (destroying it) to a singleton creating a cluster

Population: (C: <C1,N1> --> <C2,N2>)

M10 :- move(x)(Node(x), Cluster(y), IsIn(x,y) -> Cluster(z), IsIn(x,z), Create(z), Delete(y), ~(z=y))



Rule C1 & D1

Rule C1

Description: Creation of a singleton node

Population: (C: <> --> <N>)

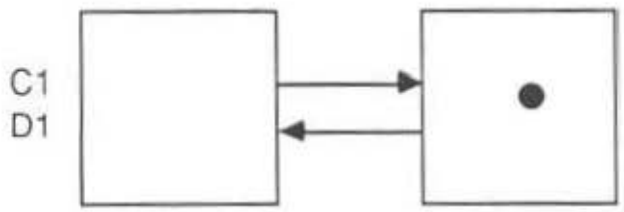
C1 :- newNode(x)(Node(x) -> Create(x),~(InCluster(x))) |
C1 :- pasteNode(x)(Node(x) -> Create(x), ~(InCluster(x)))

Rule D1

Description: Deletion of a singleton node

Population: (C: <N> --> <>)

D1 :- delete(x)(Node(x), ~(InCluster(x)) -> Delete(x))



Rules C2 & D2

Rule C2

Description: Creation of a node causing creation of a cluster

Population: (C: <N> --> <C>)

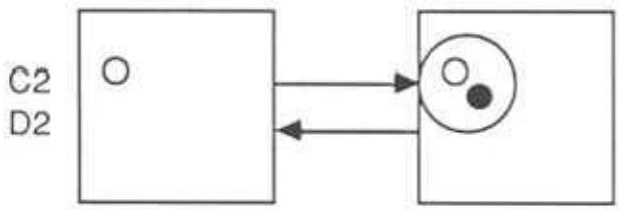
C2 :- newNode(x)(Node(x) -> Create(x), Cluster(y), Create(y), IsIn(x,y)) |
C2 :- pasteNode(x)(Node(x) -> Create(x), Cluster(y), Create(y), IsIn(x,y))

Rule D2

Description: Deletion of a node causing deletion of a cluster

Population: (C: <C> --> <N>)

D2 :- delete(x)(Node(x),Cluster(y), IsIn(x,y) -> Delete(x), Delete(y)))



Rules C3 & D3

Rule C3

Description: Creation of a node within a cluster

Population: (C: <C> --> <C>)

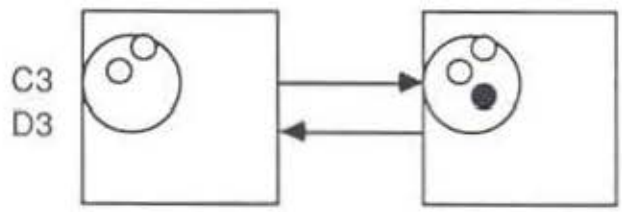
C3 :- newNode(x)(Node(x), Cluster(y) -> Create(x),Cluster(z), IsIn(x,z), (x=y)) |
C3 :- pasteNode(x)(Node(x), Cluster(y) -> Cluster(z), IsIn(x,z), (x=y))

Rule D3

Description: Deletion of a node within a cluster

Population: (C: <C> -> <C>)

D3 :- delete(x)(Node(x), Cluster(y), IsIn(x,y) -> Delete(x),Cluster(y), (x=y))



Rules C4 & D4

Rule C4

Description: Creation of a node creating a new cluster, and destroying other clusters.

Population: (C: <C1,C2> --> <C3>)

C4 :- newNode(x)(Node(x), Cluster(y), Cluster(z) -> Create(x),Cluster(w), IsIn(x,w), Create(w), Delete(y), Delete(z))
C4 :- pasteNode(x)(Node(x), Cluster(y), Cluster(z) -> Create(x),Cluster(w), IsIn(x,w), Create(w), Delete(y), Delete(z))

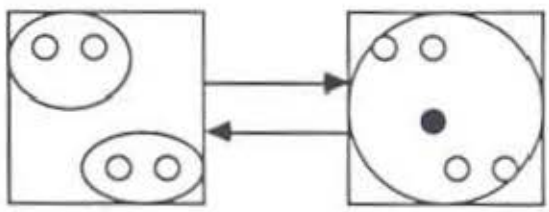
Rule D4

Description: Deletion of a node destroying one cluster but creating other clusters.

Population: (C: <C1> -> <C2,C3>)

D4 :- delete(x)(Node(x), Cluster(y),IsIn(x,y) -> Delete(x),Delete(y), Cluster(z), Cluster(w), Create(z), Create(w))

C4
D4



Rules L1 & UL1

Rule L1

Description: Creation of an isolated link

Population: (S: <N1,N2> --> <L>)

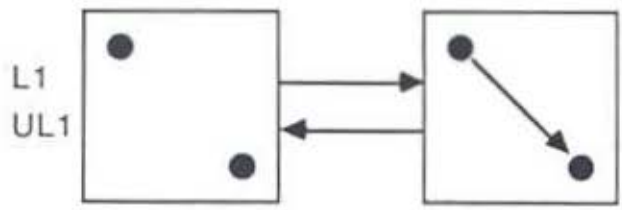
L1 :- newLink(x)(Link(x) -> Relation(y), IsIn(x,y), Create(x), Create(y))

Rule UL1

Description: Deletion of an isolated link

Population: (S: <L> --> <N1,N2>)

UL1 :- delete(x)(Link(x), Relation(y), IsIn(x,y)-> Delete(x),Delete(y))

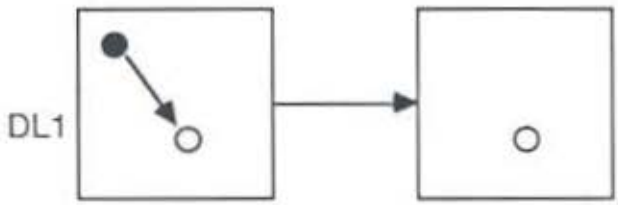


Rule DL1

Description: Deletion of an isolated node which is linked

Population: (S: <L> --> <N>)

DL1 :- delete(x)(Node(x), Relation(y), IsIn(x,y), ~(InCluster(x)) ->
Delete(x),Delete(y))

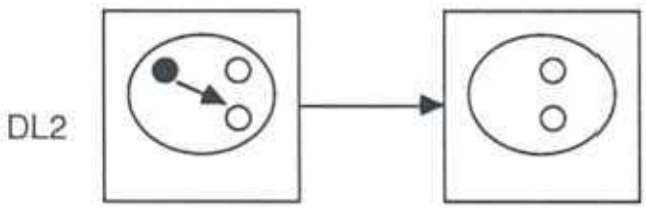


Rule DL2

Description: Deletion of a node which is within a cluster;
cluster status unchanged

Population: (S: <L> --> <N>)
Population: (C: <C> --> <C>)

DL2 :- delete(x)(Node(x), Relation(y), Cluster(z), IsIn(x,y), IsIn(x,z) ->
Delete(x), Delete(y))

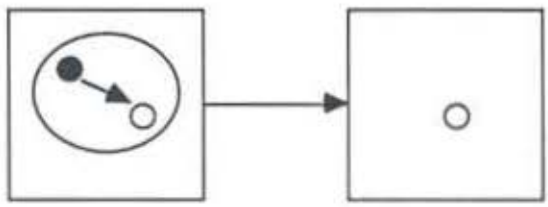


Rule DLC1

Description: Deletion of a linked node destroying a cluster
Population: (S: <L> --> <N>) (C: <C> -> <N1 .. NN>)

DLC1 :- delete(x)(Node(x), Link(w),Relation(y), Cluster(z), IsIn(x,y), IsIn(x,z) ->
Delete(w),Delete(x),Delete(y), Delete(z))

DLC1



Rules TC1 & TD1

Rule TC1

Description: Creation of a tree in tree mode

Population: (S: <T> --> <T>)

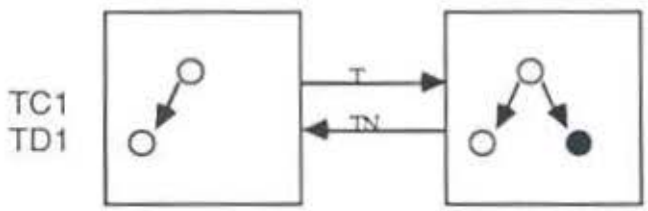
TC1 :- newNode(x)(Node(x) -> Tree(y), IsIn(x,y), Create(x), Create(y),
Root(x,y))

Rule TD1

Description: Deletion of a node destroying a tree

Population: (S: Net: <T> --> <L>)
(Possibly) (C: <C1..Cn> --><C1..Cm,N1..Nj>)

TD1 :- delete(x)(Node(x), Tree(y), IsIn(x,y), leaf(x,y) -> Delete(x), Delete(y))



Δ-Product Grammar

Rules TC2 & TD2

Rule TC2

Description: New root creation in tree mode

Population: (S: <T> --> <T>)

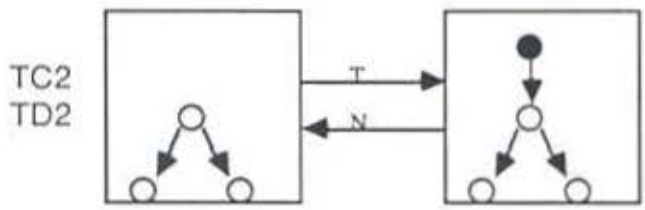
TC2 :- newNode(x)(Node(x), Tree(y) -> Tree(z), Create(x), IsIn(x,z),
Root(x,z), (y=z))

Rule TD2

Description: Deletion of a linked node which is the root of a
tree in net mode

Population: (S: <T> --> <T>)
(Possibly)(C: <C1..Cn> --> <C1..Cm,N1..Nj>)

TD2 :- delete(x)(Node(x), Tree(y), IsIn(x,y), Root(x,y) -> Delete(x))



Rules TC3 & TD3

Rule TC3

Description: Addition of an interior node in tree mode

Population: (S: <T> --> <T>)

TC3 :- newNode(x)(Node(x), Tree(y) -> Create(x), Tree(z), IsIn(x,z), (z=y), Interior(x,z))

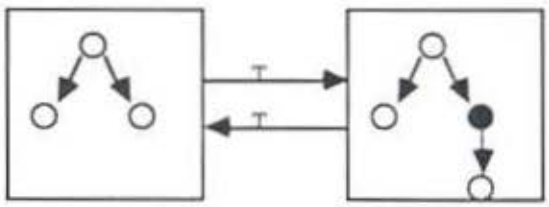
Rule TD3

Description: Deletion of an interior node in tree mode.

Population: (S: <T> --> <T>)

TD3 :- delete(x)(Node(x), Tree(y), IsIn(x,y), Interior(x,y) -> Delete(x))

TC3
TD3



Rules TC4 & TD4

Rule TC4

Description: Addition of a leaf node in tree mode

Population: (S: <T> --> <T>)

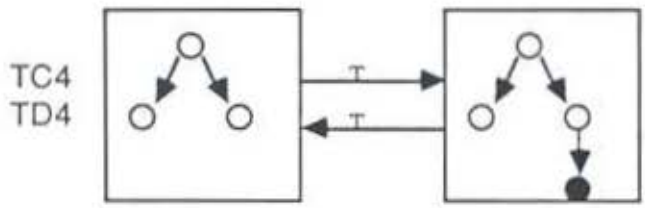
TC4 :- newNode(x)(Node(x), Tree(y) -> Create(x), Tree(z), IsIn(x,z), (y=z), Leaf(x,z))

Rule TD4

Description: Deletion of a leaf node in tree mode

Population: (S: <T> --> <T>)

TD4 :- delete(x)(Node(x), Tree(y), IsIn(x,y), Leaf(x,y) -> Delete(x))



Δ-Product Grammar

Rules STC1 & STD1

Rule STC1

Description: Addition of a subtree in tree mode

Population: (S: <T1,T2> --> <T1>)

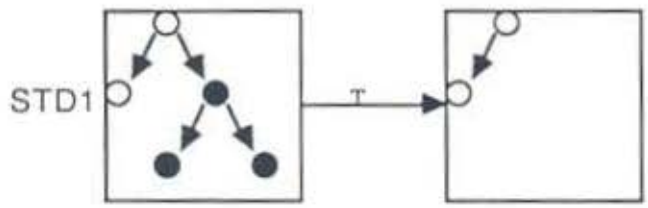
STC1 :- pasteTree(x)(Tree(x), Tree(y) -> Tree(z), IsIn(x,z), (y=z))

Rule STD1

Description: Deletion of a subtree in tree mode

Population: (S: <T> --> <T>)

STD1 :- deleteSubtree(x)(Tree, Tree(x), Tree(y), IsIn(x,y) -> Delete(x))



Rule TPC1

Description: Creation of a tree in tree mode by pasting.

Population: (S: Nil --> <T>)

TPC1 :- pasteTree(x)(Tree,Tree(x) -> Create(x))

Δ-Product Grammar

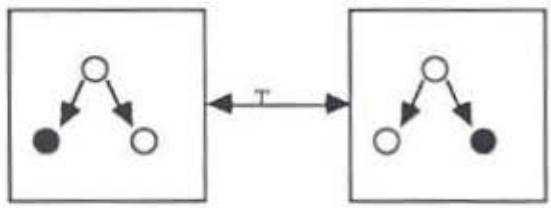
Rule TM1

Description: Movement of a node in a tree in tree mode

Population: (S: <T> -> <T>)

TM1 :- move(x)(Node(x), Tree(y), IsIn(x,y) -> Tree(z), IsIn(x,z), (y=z))

TM1



Δ-Product Grammar

Rules TL1 & TL2

Rule TL1

Description: Link creation causing creation of a tree in net mode

Population: (<L,N> --> <T>)

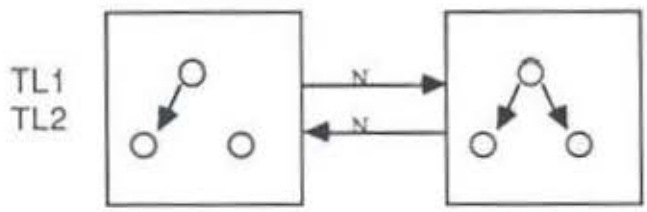
TL1 :- newLink(x)(Link(x), Relation(y) -> Tree(z), IsIn(x,z),
IsIn(y,z), Create(z), Create(x))

Rule TL2

Description: Deletion of a link causing a tree to be destroyed

Population: (S: <T> --> <L,N>)

TL2 :- delete(x)(Link(x), Tree(y), Root(y,Source(x)),
(Leaf(y,Destination(x)) ∨ (Interior(y,Destination(x)), IsIn(x,y) ->
Delete(x), Delete(y))



Rules TL3 & TL4

Rule TL3

Description: Addition of a link giving a tree a new root

Population: (<L,T> --> <T>)

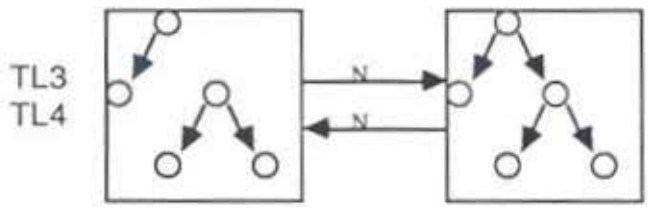
TL4 :- newLink(x)(Link(x), Tree(y) -> Create(x), IsIn(x,y))

Rule TL4

Description: Deletion of a link leaving a tree and a relation

Population: (S: <T> -> <L,T>)

TL4 :- delete(x)(Link(x), Tree(y), IsIn(x,y) -> Delete(y))



Rules TL5 & TL6

Rule TL5

Description: Addition of a link (from a singleton node) giving a tree a new root

Population: (S: <N,T> --> <T>)

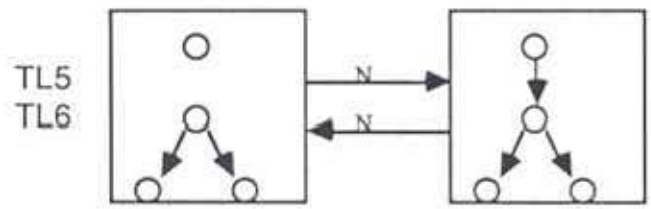
TL5 :- newLink(x)(Link(x), Tree(y) -> Create(x), Tree(z), IsIn(x,z), Root(z,Source(x)), Interior(z,Destination(x)), (z=y))

Rule TL6

Description: Deletion of a link leaving a tree and a singleton node

Population: (S: <T> --> <T,N>)

TL6 :- delete(x)(Link(x), Tree(y), IsIn(x,y), Root(y, Source(x)), Interior(y,Destination(x)) -> Delete(x), Tree(z), (z=y))

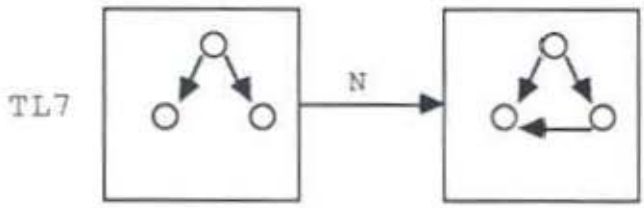


Rule TL7

Description: Creation of a link destroying a tree and creating a network.

Population:(S: <T> --> <W,L>)

TL7 :- newLink(x)(Link(x), Tree(y), Network(z), IsIn(x,y) ->
Create(x), Create(z), IsIn(x,z))



rule EncN

Description: Associating text with a node in edit mode

EncN :- encode(x)(Node(x) -> Encoded(x))

2.5.3 Sample Annotated Transcript - Δ–Product Level

The following is an annotated Δ-product level transcript. Annotations follow lines (or groups of lines) which they annotate, and are preceded by two asterisks.

```
'Writing Environment Transcript version 2.0'
version:    2.0
date:       23 September 1988
time:       1:23:26 pm
user:       jenkins
clock:      27947
database:   emptyWS
empty:      true

   206860   5260          C1          (Net: nil -> N:1 )

** Node # 1 is created in network mode; the node is created outside
** of any existing cluster.

   212120   3300          C1          (Net: nil -> N:2 )
   215420   10400         C2          (Net: nil -> N:3 C:1 )

** Node # 3 is created in network mode; the node creation causes a
** cluster to be created. The C2 indicates that a node is created
** creating a cluster.

   225820   6760          C3          (Net: nil -> N:5 C:3 )
   232580   4360          L1          (Net: nil -> L:1 )

** A link is created creating a relation between nodes.

   236940   5580          TL1         (Net: nil -> T:1 L:2 )

** A second link is created creating a tree.

   242520   6380          C3          (Net: nil -> N:6 C:3 )

** A node is created within cluster 3.

   248900   10040         M2          (Net: N:6 -> C:3 )

** A node is moved within cluster 3.

   258940   4360          C3          (Net: nil -> N:7 C:3 )
   263300   2520          L1          (Net: nil -> L:3 )
   265820   4200          C3          (Net: nil -> N:8 C:3 )
   270020   3780          TL1         (Net: nil -> T:2 L:4 )

** A link is creating creating a second tree.
```

273800	8180	M2	(Net: N:8 -> C:3)
281980	5540	C3	(Net: nil -> N:9 C:3)
287520	3540	TL5	(Net: T:2 -> T:3 L:5)

** A link is created linking two trees (two trees become one).

291060	62980	TL3	(Net: nil -> L:6)
354040	67340	EncN	(Edit: N:9 -> nil)
421380	3700	EncN	(Edit: N:9 -> nil)

** Text is associated with node # 9.

425080	5300	TC4	(Tree: N:10 -> nil)
430380	9280	TC4	(Tree: N:11 -> nil)
439660	9780	TM1	(Tree: N:10 -> nil)
449440	9000	TC4	(Tree: N:12 -> nil)
458440	9940	TC4	(Tree: N:13 -> nil)
468380	10760	TM1	(Tree: N:13 -> nil)
479140	5360	C3	(Net: nil -> N:14 C:3)
484500	37640	TD4	(Tree: N:13 -> nil)

2.6 Cognitive Process Level

Introduction

The Cognitive Process Level represents the cogntive process inferred to be active in the mind of the user in producing one or more changes to the set of cognitive products. Thus, for example, a sequence of additions to a group of nodes in close spatial proximity to one another is interpreted as an instance of sustained "Focused Recall" whereas, if the nodes are "far apart", the process is assumed to be unfocused or free recall. Currently, the grammar also inserts system mode shift operations into the Cognitive Process Transcript to facilitate the Cognitive Mode portion of the grammar, described below. However, we anticipate removing these symbols in the near future when that portion of the grammar is completed. In the sections that follow, we first discuss the transcription language for the Cognitive Process Level, then the grammar, and, finally, a sample annotated transcript.

2.6.1 Cognitive Process Transcript Language

A cognitive process transcript is a record of the cognitive processes posited as having occurred in the mind of a user of the Writing Environment during the course of a session. A cognitive process transcript is preceded by a header and followed by a list of cognitive process records (one per line).

Header

The header lists the following items, in the following order:

1) The version of the Writing Environment (WE) which was being used when the transcript was recorded.
2) The date of the transcript.
3) The time at which the session began.
4) The name of the user.
5) The clock at the beginning of the session.
6) The name of the database being used.
7) A boolean indicating whether the database began with an empty workspace.

Cognitive Process Record

The format for the cognitive process records is consistent with that followed at other levels. That format is simply:

Time Symbol Attributes

where:

Time = The time (in milliseconds) from the beginning of the user session until the beginning of the action (operation, etc.) which that particular line of the transcript purports to record.

Symbol = The symbol which represents the type of cognitive process which is recorded on the transcript.

Attributes = The object of the cognitive process where object=the entity which is the focus of the current cognitive process. Certain processes (e.g. "Shift Perspective Network" (ShiN)) have no object.

A cognitive process symbol is output only if either: a) the cognitive process is different from the previous cognitive product, or b) if the cognitive process is the same as the previous process, the entity which is the focus of the two processes is different. Thus if a "Loose Recall" is followed by a "Focused Recall", two records will always be created. If a "Focused Recall" is followed by a "Focused Recall", two records will be created only if two different clusters were the subject of the disparate "Focused Recalls".

Cognitive Process	Symbol	Modes	Description
Loose Recall	LOO	N	Node Creations in Empty Space
Focused Recall	FOC	N	Node Creations in Clusters
Cluster	CLU	N	Node Movements into Clusters
Decluster	DEC	N	Node Deletions/Movements out of Clusters
Refine Cluster	REC	N	Node Movement within Clusters
Relate	REL	N,T	Linking One Node to Another/No Tree
Derelate	DER	N,T	Unlinking an UnTree
Hierarchize	HIE	N,T	Creation of a Tree
Dehierarchize	DEH	N,T	Destruction of a Tree
Develop Hierarchy	DEV	N,T	Adding to a Tree
Shrink Hierarchy	SHK	N,T	Shrinking a Tree
Synthesize Hier.	SYN	N,T	Joining Two Trees
Desynth. Hier.	DES	N	Breaking one tree into two trees
Refine Hier.	REF	N,T	Rearranging the Nodes in a Tree
Shift Perspective Net	ShiN	N	Shift Perspective To Network Mode
Shift Perspective Tree	ShiT	T	Shift Perspective To Tree Mode
Shift Perspective Edit	ShiE	E	Shift Perspective To Edit Mode
Shift Perspective Revise	ShiR	R	Shift Perspective To Revise Mode
Shift View	ShiV	N,T,E,R	Shift the viewing space of a mode
Shift Context	ShiC	T	Shift the portion of a tree being viewed
Encode	ENC	E	Associating Text with a Node

2.6.2 Cognitive Process Grammar

The grammar for the cognitive process level is very simple. Every cognitive process rule is mapped to either one or more Δ-products, or one or more operations. These mappings are summarized here in two ways. First, two tables are given. Table 1 lists those cognitive processes which rewrite one or more Δ-product symbols. Table 2 lists those cognitive processes which rewrite one or more operation symbols.

Secondly, a string parsing grammar similar to the grammars found at the action and operation level is given formally describing the rewrite rules used in obtaining a mapping from previous levels to the cognitive process level.

Table 1:

Cognitive Process	Symbol	Δ-Product(s)
Loose Recall	LOO	C1
Focused Recall	FOC	C2,C3,C4
Cluster	CLU	M4,M5,M6,M7,M9,M10
Decluster	DEC	M3,M8,D2,DLC1
Refine Cluster	REC	M2,D3
Relate	REL	L1
Derelate	DER	UL1,DL1,DL2
Hierarchize	HIE	TC1,TL1
Dehierarchize	DEH	TD1,TL2,TL7
Develop Hierarchy	DEV	TC3,TC4
Shrink Hierarchy	SHK	TD3,TD4,STD1
Synthesize Hier.	SYN	TC2,TL3,TL5
Desyn. Hier.	DES	TD2,TL4,TL6
Refine Hier.	REF	TM1

Table 2:

Cognitive Process	Symbol	Operation(s)
Shift to Network	ShiN	mode "Net"
Shift to Tree	ShiT	mode "Tree"
Shift to Edit	ShiE	mode "Para"
Shift to Revise	ShiR	mode "Revise"
Shift View	ShiV	layout | view
Shift Context	ShiC	context
Encode	Enc	encode

The Grammar:

Loose Recall Record (Time) ::= C1(Time);

Focused Recall Record (Time) ::= C2 (Time) |
C3 (Time) |
C4 (Time);

```
Cluster Record (Time)                ::= M4 (Time) |
                                         M5 (Time) |
                                         M6 (Time) |
                                         M7 (Time) |
                                         M9 (Time) |
                                         M10 (Time);

Decluster Record (Time)              ::= M3 (Time) |
                                         M8 (Time) |
                                         D2 (Time) |
                                         DLC1 (Time);

Refine Cluster Record (Time)         ::= M2 (Time) |
                                         D3 (Time);

Relate Record (Time)                 ::= L1 (Time);

Derelate Record (Time)               ::= UL1 (Time) |
                                         DL1 (Time) |
                                         DL2 (Time);

Hierarchize Record (Time)            ::= TC1 (Time) |
                                         TL1 (Time);

Dehierarchize Record (Time)          ::= TD1 (Time) |
                                         TL2 (Time) |
                                         TL7 (Time);

Develop Hierarchy Record (Time)      ::= TC3 (Time) |
                                         TC4 (Time);

Shrink Hierarchy Record (Time)       ::= TD3 (Time) |
                                         TD4 (Time) |
                                         STD1 (Time);

Synthesize Hierarchy Record (Time)   ::= TC2 (Time) |
                                         TL3 (Time) |
                                         TL5 (Time);

Desynthesize Hierarchy Record (Time) ::= TD2 (Time) |
                                         TL4 (Time) |
                                         TL6 (Time);

Refine Hierarchy Record (Time)       ::= TM1 (Time);

** The right-hand sides of the following rules refer to records found at the operation
** level.

Encode Record                        ::= Encode (Time);

Shift Tree Mode Record (Time)        ::= Mode ("Tree", Time);

Shift Network Mode Record (Time)     ::= Mode ("Net", Time);
```

Shift Edit Mode Record (Time)	::=	Mode ("Para", Time);
Shift Revise Mode Record (Time)	::=	Mode ("Revise", Time);
Shift View Record (Time)	::=	Layout (Time) | View (Time);
Shift Context Record (Time)	::=	Context (Time).

2.6.3 Sample Annotated Transcript - Cognitive Process Level

The following is an annotated cognitive process level transcript. Annotations follow lines (or groups of lines) which they annotate, and are preceded by two asterisks.

```
'Writing Environment Transcript version 2.0'
version:    2.0
date:       23 September 1988
time:       1:23:26 pm
user:       jenkins
clock:      27947
database:   emptyWS
empty:      true

   80          205340      ShiE

** The subject shifts to edit mode.

   205420      1440        ShiN

** The subject shifts to network mode.

   206860      5260        Loo
   212120      3300        Loo

** Two instances of loose recall occur.

   215420      10400       Foc
   225820      6760        Foc

** Two instances of focused recall occur.

   232580      4360        Rel
   236940      5580        Hie

** The subject "hierarchizes."

   242520      6380        Foc
   248900      10040       Rcl

** The subject refines a cluster.

   258940      4360        Foc
   263300      2520        Rel
   265820      4200        Foc
   270020      3780        Hie
   273800      8180        Rcl
   281980      5540        Foc
   287520      3540        Hie
```

291060	22200	Syn

** The subject synthesized a hierarchy.

313260	29700	ShiT

** The subject shifts to tree mode.

342960	2680	ShiV

** The subject shifts view (by for example viewing a subtree).

345640	8400	ShiE
354040	67340	Enc

** The subject writes for a period of 67340 milliseconds in edit mode.

421380	40	Enc
421420	3660	ShiT
425080	5300	Dev
430380	9280	Dev

** The subject develops a hierarchy.

439660	9780	Reh

** The subject refines a hierarchy.

449440	9000	Dev
458440	9940	Dev
468380	8780	Reh
477160	1980	ShiN
479140	2780	Foc
481920	2580	ShiT
484500	7300	Shk

The subject shrinks a hierarchy.

2.7 Cognitive Mode Level

Introduction

The Cognitive Mode level represents the largest shifts in cognitive behavior modeled by the grammar. Shifts in Cognitive Mode are strongly suggested when the user shifts from one system mode to another. But the two are not always the same. For example, when a user working in network mode shifts from building small conceptual structures to linking them into a larger hierarchical structure, this may indicate a shift in Cognitive Mode. On the other hand, when the user is building a large hierarchical structure in tree mode and returns to network mode to copy a structural component into the tree, that shift in system mode may not indicate a shift in Cognitive Mode. At present, this portion of the grammar is incomplete. We currently infer shifts in cognitive mode largely from shifts in system mode operations inserted into the Cognitive Process Transcript, but we will add rules in the near future to infer shifts from context sensitive sequences of cognitive process symbols. In the sections that follows, we first discuss the transcription language for this level, then the grammar, and, finally, a sample annotated transcript.

2.7.1 Cognitive Mode Transcript Language

A cognitive mode level transcript consists of a header followed by a series of one or more cognitive mode records. Currently we posit four cognitive modes in which a user of WE may function: an exploration mode, an organization mode, a writing mode, and a revision mode. By design these four cognitive modes correspond closely to the four system modes (in order - network mode, tree mode, edit mode, and text mode), although the mapping is not one-to-one. The user may shift system modes without shifting cognitive mode or shift cognitive modes without shifting system mode. (See grammar.) Each cognitive mode record occupies one line of the transcript.

Header

The header lists the following items, in the following order:

1) The version of the Writing Environment (WE) which was being used when the transcript was recorded.
2) The date of the transcript.
3) The time at which the session began.
4) The name of the user.
5) The clock at the beginning of the session.
6) The name of the database being used.
7) A boolean indicating whether the database began with an empty workspace.

Cognitive Mode Record

Each line of the cognitive mode transcript contains three columns. The first column contains the absolute time since the beginning of the transcript at which the cognitive mode began. This value is obtained directly from the first column of the operation level transcript.

The second column contains the duration of the cognitive mode, and is obtained simply by subtracting the absolute time of the next cognitive mode from the absolute time of the current cognitive mode. The duration of the last cognitive mode of the transcript is obtained by subtracting the last absolute time listed on the operation transcript from the absolute time of the current cognitive mode.

The third column of the cognitive mode transcript lists a symbol which represents the system's posit as to what cognitive mode was underway during the period represented by columns one and two. Currently four such symbols appear in the transcript:

1. Explore: The symbol for exploration mode.
2. Organize: The symbol for organization mode.
3. Edit: The symbol for write mode.
4. Revise: The symbol for revise mode.

Thus the template for a line of the cognitive mode transcript is:

Time Duration Symbol

where "Symbol" is either "Explore," "Organize," "Edit," or "Revise."

2.7.2 Cognitive Mode Grammar

The cognitive mode grammar tries to capture from the operation level transcript along with certain additional information the current cognitive mode of a user of WE. The cognitive mode ascription is based primarily on the current system mode. However shifts in system mode do not necessarily signal shifts in cognitive mode. For example the operation record will record a shift in system mode in response to the cursor passing momentarily over (e.g.) the text mode area of the screen. The user may move the cursor from tree to network mode, but do nothing before returning to tree mode. We filter the operation record transcript to avoid positing cognitive mode shifts in such circumstances.

On the other hand, we posit that the user may shift cognitive modes while remaining in the same system mode. For example the user may be in network mode, with Exploration mode being posited as the current cognitive mode. The user may begin systematically building a large tree, during which we might be warranted in positing a shift in cognitive mode from Exploration mode to Organization mode. In the current implementation we do not capture this sort of cognitive shift. In later implementations the cognitive mode posits will be based upon the cognitive process transcript; we then will be able to capture this type of cognitive mode shift.

The operation level transcript lists all system mode changes which occur during a user session. Every mode change is one of five possible types:

1: Network mode: the network system mode;
2: Tree mode: the tree system mode;
3: Edit mode: the edit system mode;
4: Revise mode: the text system mode;
5: Control Panel mode: the mode in which a user has access to the system-wide menus. These menus are accessed through the bar at the top of the writing environment.

Every such mode change will result in the creation of an cognitive mode event record with three exceptions:
i: The mode "cp" does not result in the creation of an event record;
ii: If the duration of the mode change was less than one second, an event record is not created. The duration of the mode is relative to the next mode change; thus duration is calculated by subtracting the time at the beginning of the next mode from the time at the beginning of the current mode;
iii: If the mode change was either to network or tree mode, and the user does nothing in the mode before changing to a different mode, an event record is not created. Nothing is done in the mode if the operation (in the operation level transcript) following the mode change to tree or network mode is another mode change.

From points ii. and iii. it follows that an event record is created for the current mode change only after the next mode change. The last event record of the cognitive mode transcript is created (or not created) by using the last time given on the operation level transcript.

Thus (informally) the rules are:

Exploration Mode (Time1, (Time2-Time1)) :-
(Operation Record = Time1 "mode" "Net") &
(Time Until Next Mode Change > 1 second) &
(Next Operation Record ≠ Time "mode") &
((Next Record of a Mode Change = Time2 "mode") V
(Next Record = Time2 "closeSession")

Organization Mode (Time1, (Time2-Time1)) :-
(Operation Record = Time1 "mode" "Tree") &
(Time Until Next Mode Change > 1 second) &
(Next Operation Record ≠ Time "mode") &
((Next Record of a Mode Change = Time2 "mode") V
(Next Record = Time2 "closeSession")

Write Mode (Time1, (Time2-Time1)) :-
(Operation Record = Time1 "mode" "Edit") &
(Time Until Next Mode Change > 1 second)
((Next Record of a Mode Change = Time2 "mode") V
(Next Record = Time2 "closeSession")

Revise Mode (Time1, (Time2-Time1)) :-
(Operation Record = Time1 "mode" "Net") &
(Time Until Next Mode Change > 1 second) &
((Next Record of a Mode Change = Time2 "mode") V
(Next Record = Time2 "closeSession")

2.7.3 Sample Annotated Transcript - Cognitive Mode Level

The following is an annotated cognitive mode level transcript. Annotations follow lines (or groups of lines) which they annotate, and are preceded by two asterisks.

```
'Writing Environment Transcript version 2.0'
version:    2.0
date:       23 September 1988
time:       1:23:26 pm
user:       jenkins
clock:      27947
database:   emptyWS
empty:      true

    80          205340    Edit
```

** The subject begins in edit mode.

```
    205420      107840    Explore
```

** The subject moves into exploratory mode.

```
    313260      32380     Organize
```

** The subject shifts to organizational mode.

```
    345640      75780     Edit
    421420      55740     Organize
    477160      4760      Explore
    481920      9880      Organize
    491800      4920      Explore
```