A Database for Analyzing Sequential Behavioral Data and Their Associated Cognitive Models

Bonnie E. John

15 May 1994
CMU-CS-94-127

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

This report is a revised version of "Applying Cognitive Theory to the Evaluation and Design of Human-Computer Interfaces," Final Report to USWest Advanced Technologies Sponsored Research Program, 14 Dec 1990.

Also appears as Human-Computer Interaction Institute Technical Report CMU-HCII-94-101

Abstract

Sequential behavioral data, be it verbal protocols, automatically-recorded keystrokes, or complete videotape protocols, can be analyzed at different levels of detail and from different viewpoints. If raw behavioral data is stored in a powerful database, rather than a simple text file, many domains will allow some automatic interpretation of that data. In addition, the raw data can be compared with traces of an associated computational cognitive model to assess how well the model accounts for the data and, conversely, how much support the behavioral data provides for the components of the model. This report describes a prototype database and user interface, called Trace&Transcription, that is designed to facilitate protocol analysis and cognitive modeling.

This research was supported in part by a grant from U S West Advanced Technologies Sponsored Research Program. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of U S West.

Keywords: Verbal protocols, protocol analysis, cognitive modeling, exploratory sequential data analysis

1. Introduction

Sequential behavioral data, be it verbal protocols, automatically-recorded keystrokes, or complete videotape protocols, can be analyzed at different levels of detail and from different viewpoints. Behavior is often first described as observation of overt action: verbal utterances, keystrokes, mouse movements, screen changes, etc. This level of description is pure observation, with no interpretation added by the analyst, and is often compiled either automatically (e.g., keystroke collection programs) or by clerical personnel skilled in transcription. This description produces vast amounts of data. Sheer quantity makes handling these data a difficult process. A long standing goal of researchers who use this type of data has been to create a methodology for handling its quantity and complexity, but as yet, no single method has emerged as a standard in the community. Our approach is to set up a database that allows us to relate several different descriptions of the same behavior in an attempt to meet this goal.

If raw behavioral data is stored in a powerful database, rather than a simple text file, many domains will allow some automatic interpretation of that data. For instance, in studying a programming environment with a programming window, an execution window and a help window (e.g., Figures 1 and 2), a powerful database would be able to identify activity in specific windows and segment the behavior into broad episodes. That is, mouse activity in the help window, probably indicates a browsing episode where the user is looking for syntax or semantics of a command; mouse or keyboard activity in the programming window probably indicates a code-editing episode. These episodes might be further categorized by their length, or by what episodes they follow or precede. For instance, long segments of behavior with verbal utterances but no mouse or keyboard activity might indicate either a program-planning episode or a reading episode, depending on whether they are preceded and followed by programming episodes (for planning) or browsing episodes (for reading). This second level of analysis, automatically generated by the database itself, could also be stored in the database and allow the analyst to easily identify the behavior for more detailed study.

Protocols can also be coded for critical incidents, events where the user either makes a mistake, has trouble understanding or using the interface, or makes some explicit remark either praising or condemning the system. These critical incidents could also appear in the database.

Another frequently used method of analyzing protocol data, is to code the behavior according to a general model of an activity. For instance, in our work with browsing, we have developed a model that decomposes a browsing activity into three sub-activities: defining criteria for search and evaluation, performing the actual search, and evaluating the results of the search. The protocol data automatically identified as browsing, could then be examined and hand-coded for these sub-activities. This information could then be added to the database.

Another way of looking at the data might be to perform a GOMS analysis of segments of behavior (Card, Moran & Newell, 1983). The raw data is arranged chronologically and the GOMS hierarchy of goals could also be associated with segments of behavior in the database. This would allow identification of when a goal is begun, suspended, re-activated, and completed. Patterns in goal-switching could be identified and quantified. Both chronological and goal-hierarchic displays of behavior could be produced.

Finally, the most detailed level of analysis of protocol data usually undertaken is a computer simulation of the behavior. For instance, we have a cognitive model of browsing that produces much of the verbal utterances and mouse events for several segments of browsing behavior (John, Newell & Card, 1990; Peck & John, 1992). A powerful database would allow traces of a computer simulation to be associated with individual behavioral events, and facilitate examination of the goodness of fit of the computer simulation to the behavior. Also, correspondence between the different analysis viewpoints could be obtained, and comparison of their effectiveness could be performed.

2. The Data

For a prototype database, we chose to included the raw videotape protocol data and the most detailed level of analysis, a trace of a cognitive simulation model. The prototype demonstrates how these two levels of analysis, one the pure description of observed behavior and the other a generalized model of that behavior, could be related usefully in a single database system.

2.1 Protocol Data

We focused on the browser associated with the on-line reference manual for the cT programming language.[1] cT provides a highly interactive programming environment where, typically, code is entered into one window (the programming window) and the results of the code are displayed in a second window (the execution window) (Figure 1). Changes in the code produce changes in the display and, conversely, a graphic editor allows changes to be made directly in the execution window which are immediately reflected in the code. In addition to the programming and execution windows, a Help window can be brought up to provide access to the on-line reference manual (an exact duplicate of the hard-copy reference manual). This Help window, shown in Figure 2, is the browser of interest in our research.

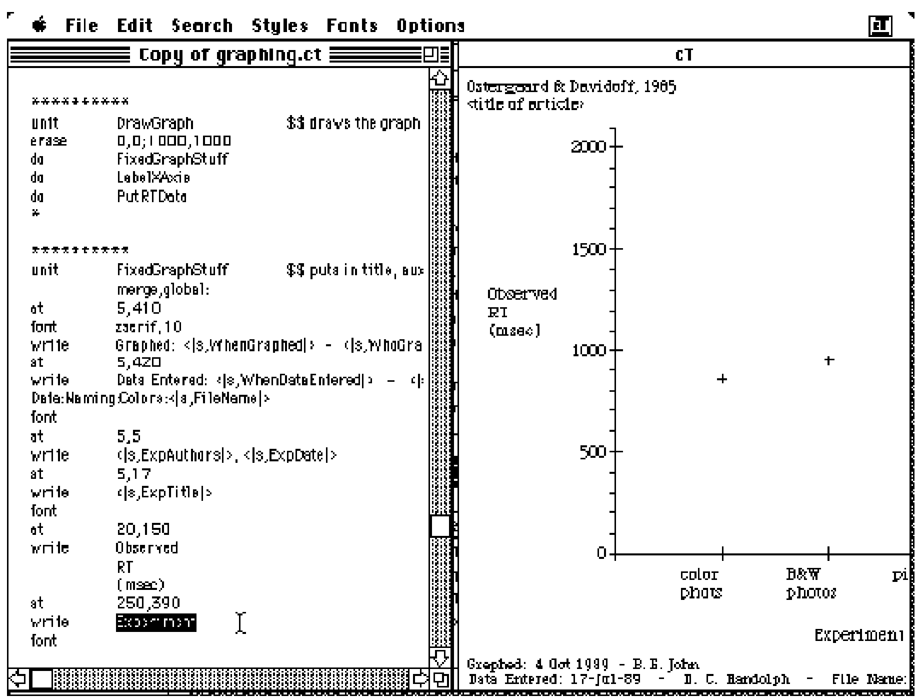

Figure 1. The typical cT programming environment on the Apple Macintosh. The window on the left is the programming window where code is entered. The window on the right is the execution

---

[1]cT is a product of Falcon Software, Inc.

window where the results of the code are displayed. Changes to the code produce changes in the execution window and changes made with a graphic editor in the execution window produce immediate changes in the code.

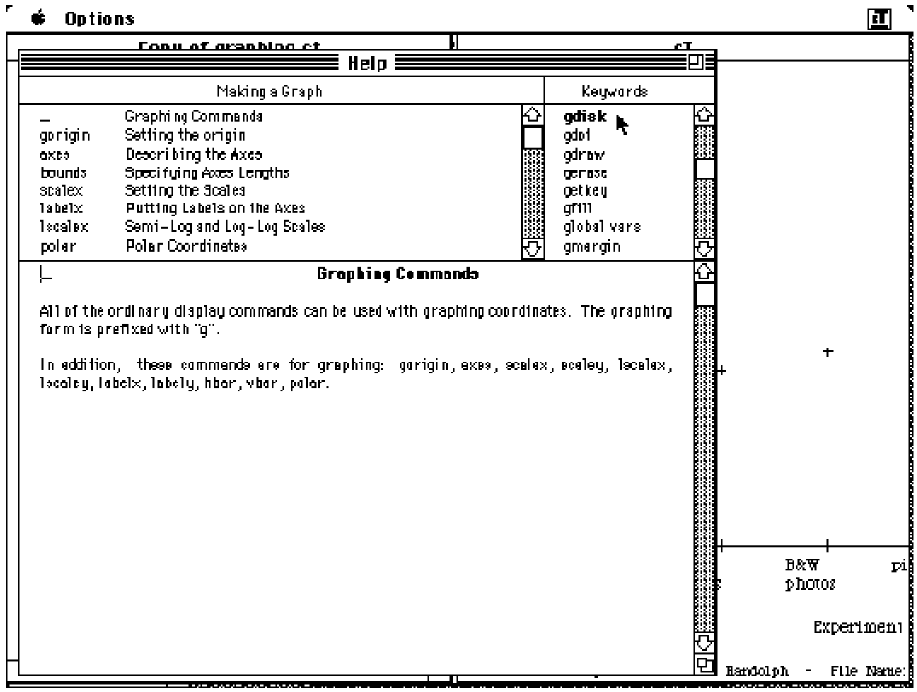

Figure 2. The cT Help window. The window on the top left is a hierarchical menu similar to a table of contents in a hard-copy reference manual. The window on the top right is an alphabetical list of all cT commands. The bottom window displays the reference material for the commands.

There are three sub-windows in the help window: a hierarchical menu (top left), an alphabetic list of cT commands (top right), and the help text (bottom). The two top windows provide two access paths to the data, which is displayed in the bottom window. When a user needs information about a cT command, he or she brings up the help window and searches for that information either by selecting increasingly more specific items from the hierarchical menu or selecting a particular command the alphabetical list. When the user selects items from the hierarchical menu, the menu changes to increasingly more specific items until specific, related commands are displayed in the menu. When the user selects a specific command in the hierarchical menu, that command is highlighted in bold print and the reference text for that command appears in the help text window. Selections in the hierarchical menu do not change the alphabetic list. However, when a user selects a command in the alphabetic list, several things happen. First, the selected command is highlighted in bold and the alphabetic list is scrolled up so the selected command is at the top of the list. Next the reference material for that command is displayed in the help text window. Last, the hierarchical menu is changed so that the selected command (highlighted in bold), and commands related to it, are displayed in the hierarchical menu.

A stated purpose of cT is to allow non-professional programmers to easily create highly interactive graphic applications. A volunteer, non-professional but experienced programmer (conversant in LISP, Pascal, FORTRAN, Basic, and assembly language) needed to create a program to read psychological data from a spreadsheet file and plot it on x-y coordinates. This person (hereafter called U1) had never used cT, but was willing to spend an afternoon attempting to create her application with cT, while being videotaped. She was asked to use only the on-line reference manual when she needed information about cT in the course of her programming. Three and a half hours of programming were videotaped, during which time

U1 completed the graphing part of her application. An additional four hours, not videotaped, was needed to complete the file input portion of the application.

We identified three types of behavior in the videotaped segment: the initial reading for general information, coding, and browsing for specific information needed in the course of coding. Of the 3.5 hours, 0.45 hours were spent on the initial reading, 1.80 hours were spent coding, and 1.25 hours were spent browsing. These 1.25 hours of browsing represent 80 browsing sequences averaging 56 seconds apiece with a range of 1 to 270 seconds.

To date, all of the browsing sequences have been transcribed so that every verbal utterance, every mouse-movement and button click, and every keystroke is recorded in a format automatically readable by the eventual database system.

2.2 Trace Data

A single, 67 second browsing sequence was chosen for detailed examination and preliminary modeling. Before this browsing sequence, U1 had produced code that displayed labeled axes on the screen and she now needed to place specific data points on the graph. She was looking for a cT command that would produce a "filled circle". The transcription of this browsing segment appears in Figure 3. We used this transcription as a guide for building a preliminary model of this browsing incident.

The cognitive model was built on the Soar architecture of cognition. As succinctly described in Lewis, et. al. (1990) and described in more detail elsewhere (Newell, 1990) the Soar architecture formulates all tasks in *problem spaces*, in which *operators* are selectively applied to the *current state* to attain *desired states*. Problem solving proceeds in a sequence of *decision cycles* that select problem spaces, states and operators. Each decision cycle accumulates knowledge from a long term *recognition memory* (realized as a production system). This memory continually matches against *working memory,* elaborating the current state and retrieving preferences that encode knowledge about the next step to take. Access of recognition memory is involuntary, parallel, and rapid (assumed to take on the order of 10 milliseconds). The decision cycle accesses recognition memory repeatedly to quiescence, so each decision cycle takes on the order of 100 milliseconds.

If Soar does not know how to proceed in a problem space, an *impasse* occur. Soars responds to an impasse by creating a subgoal in which a new problem space can be used to acquire the needed knowledge. If lack of knowledge prevents progress in a new space, another subgoal is created and so on, creating a goal-subgoal hierarchy. Once an impasse is resolved by problem solving, the *chunking* mechanism adds new productions to recognition memory encoding the results of the problem solving, so the impasse is avoided in the future. All incoming perception and outgoing motor commands flow through the state in the top problem space.

Figure 4 shows the structure of Soar when it is functioning in a highly interactive environment. The decision cycle is unchanged; all available information is accumulated about the acceptability and desirability of problem spaces, states, and operators for the total current context, and the best alternative is chosen among those that are acceptable. However, in a highly interactive environment, the outside world (in this case, the cT display on the CRT screen) is an important source of the information being collected in working memory. This influx of information influences the decision cycle both by directly depositing relevant information into working memory and by triggering long-term memory to deposit other general knowledge into working memory. The outcome of each decision cycle is to select a problem space, state, or operator for application, or generate an impasse. In a highly

Time (sec)	Verbal Behavior	Mouse Behavior	Screen Changes
05	OK		
06	so		
08	so I just want to		
13	make a mark		
15	I want to draw something		
16	I want to circle something		
18	guess I want to put a little circle in		
19		M to help screen	
20	so, let's see	C on help screen	help window comes up
21	I want a filled circle		
22	let's find...	M to command menu	
		D on up scroll arrow	command menu scrolls up
23			gcircle appears in the scrolling
		command menu	
24	"gcircle"	U+M to down arrow	gcircle scrolls off the top
			command menu stops scrolling
25		C on down arrow	command menu pages down
			gcircle is not yet in the window
		C on down arrow	command menu pages down
			gcircle is on 2nd line of menu
26	"gcircle"	M to gcircle + C	gcircle scrolls to top of menu, turns bold
			gcircle help text comes up
			hierarchical menu changes
			gdisk is visible below gcircle
29	I think it's "gdisk"		
30	actually	M to gdisk	
31	"gdisk"	C on gdisk	gdisk turns bold, help text comes up
			(same as gcircle)
			hierarchical menu changes
32	gonna give me the same thing		
33	OK-so I really want to look at...		
		M to above elevator	
34	I wonder what "dot" is		
35		C above elevator	command menu pages up
36		C above elevator	command menu pages up
37	what's "dot"?	M to dot + C	dot scrolls to top of menu, turns bold
			dot help text comes up
			hierarchical menu changes
39	making dots (start reading)		
40	single dot		
41	umm		
42	OK		
45	(reading) the dot command		
46	with one point is		
47	equivalent to a draw command		
48	with a single position (end reading)		
49	oh those are very little teeny		
50	I don't want those-I want disk		
51		M around command menu	
52	where's disk?	M to hierarchical menu	
53	dot circle		
54	draw disk		
55	filled in circle	M to disk + C	disk turns bold, help text comes up
56	disk command		
57	filled in circle		
58	umm		
60	disks, OK		

Figure 3. Transcription of the videotape protocol segment used to construct the preliminary Soar model of browsing. In the Mouse Behavior column, M = move mouse so cursor moves, D = press down on the mouse button, U = release the mouse button, C = click the mouse button (rapid down and up). Behavior in this protocol is referred to by its timestamp, e.g., t55 refers to U1 saying "filled in circle", moving the mouse to put the cursor ion the word "disk" and clicking the mouse.

interactive environment, the application of operators often leads to changes in the external environment (in this case, moving the mouse and pressing its button).

The preliminary Soar model of browsing for the cT coding task, Browser-Soar, uses the Soar problem space structure to provide the goals, operators, methods and selection rules found in traditional GOMS analyses. Browser-Soar consists of a set of problem spaces (Figure 5) that provide the capability to search deliberately through the help windows, while allowing recognition of new items to trigger knowledge at any time that may change the search strategy. The top problem space in Browser-Soar (**Browse**) is entered when an impasse arises in the task space for programming in the cT language. A browsing episode involves bringing up the help window, finding the appropriate help, and applying the newly found information to the problem at hand. Each of these activities corresponds to an operator in the **Browse** space. Currently, Browser-Soar implements the **find-appropriate-help** operator. Applying this operator results in an impasse because the operator cannot be implemented by recognition. Soar responds by setting up another problem space, with operators that define the search criteria (e.g., what labels to look for in the help windows), define the evaluation criteria (how to decide that some piece of information will actually help resolve the impasse in the task space), carry out the search, and evaluate the search results. Each of these operators is also implemented in a problem space; for example, carrying out the search for the defined criteria is accomplished in a space with operators that select among search methods and execute them. At the bottom of the problem space hierarchy are motor operators that control mouse and keyboard actions, and cognitive operators that can be

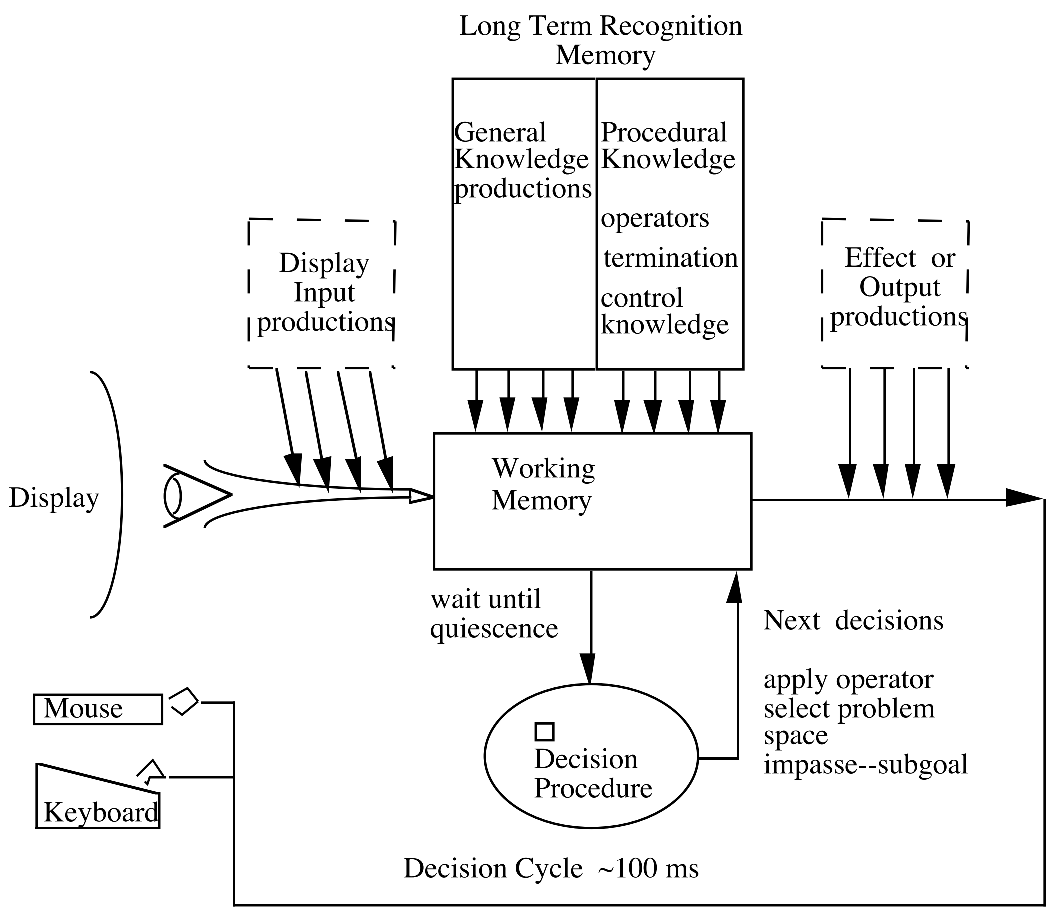

Figure 4. Soar in a highly interactive environment.

applied with directly available knowledge. The operators in Browser-Soar can be viewed as deliberate goals, and this organization is useful for modeling the goal-oriented component of browsing as well as the mechanics of manipulating the windows used for browsing. Data-driven, opportunistic behavior emerges because an additional operator, **evaluate-new-items**, is proposed whenever new information is brought within the scope of attention **evaluate-new-items** is available in every Browser-Soar problem space). The current problem solving is thus *interrupted* so the new items may be considered, possibly suggesting a more relevant path to pursue.

When given the necessary knowledge about the user's goals and the information visible on the screen in the protocol of Figure 3 and the problem space structure in Figure 5, Browser-Soar produces the trace in Figure 6.

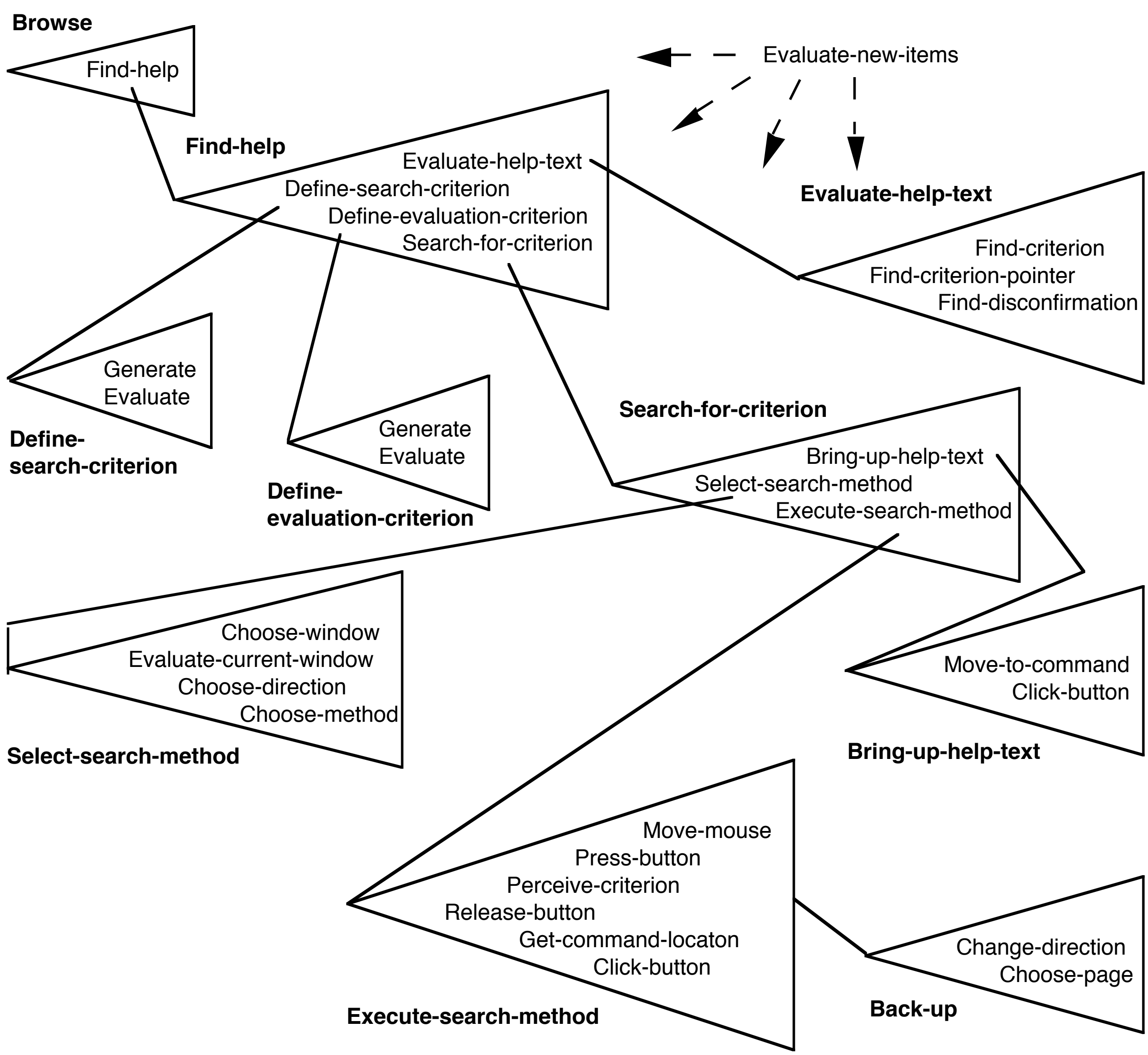

Figure 5. The problem space structure of Browser-Soar.

Figure 6. Browser-Soar trace (continues for 3 pages)

```
0    G: g1
1    P: p4 (top-space)
2    S: s5
3    O: o12 (browsing-task)
4    ==>G: g36 (operator no-change)
5       P: p42 (browsing)
6       S: s49
7       O: o51 (find-appropriate-help)
8       ==>G: g53 (operator no-change)
9          P: p59 (find-appropriate-help)
10         S: s72
11         O: o74 (define-search-criterion)
12         ==>G: g78 (operator no-change)
13            P: p84 (define-search-criterion)
14            S: s94
15            O: o100 (generate-search-criterion)
16            O: o112 (evaluate-search-criterion)
17            O: o118 (generate-search-criterion)
18            O: o128 (evaluate-search-criterion)
19            O: o134 (generate-search-criterion)
20            O: o142 (evaluate-search-criterion)
21            O: o93 (final-state)
22         O: o76 (define-evaluation-criterion)
23         ==>G: g148 (operator no-change)
24            P: p154 (define-evaluation-criterion)
25            S: s165
26            O: o169 (generate-evaluation-criterion)
27            O: o177 (evaluate-evaluation-criterion)
28            O: o164 (final-state)
29         O: o185 (search-for-criterion)
30         ==>G: g187 (operator no-change)
31            P: p193 (search-for-criterion)
32            S: s212
33            O: o214 (select-search-method)
34            ==>G: g216 (operator no-change)
35               P: p222 (select-search-method)
36               S: s236
37               O: o238 (choose-window)
38               O: o244 (focus-on-current-window)
39               O: o255 (evaluate-current-window)
40               O: o289 (choose-method)
41               O: o295 (choose-direction)
42               O: o235 (final-state)
43            O: o311 (execute-search-method)
44            ==>G: g313 (operator no-change)
45               P: p319 (execute-search-method)
46               S: s335
47               O: o337 (move-mouse-to-up-arrow)
48               O: o342 (press-mouse-button-scroll-up)
49               O: o351 (perceive-search-criterion)
50               O: o349 (release-mouse-button)
51               O: o371 (evaluate-new-items-nn)
52               O: o362 (get-location-of-command)
53               ==>G: g375 (operator no-change)
54                  P: p381 (back-up)
```

```
         S: s394
         O: o396 (change-direction)
         O: o398 (change-method)
         O: o393 (final-state)
      O: o416 (move-mouse-to-below-elevator)
      O: o421 (click-mouse-button)
      O: o438 (evaluate-new-items-nn)
      O: o421 (click-mouse-button)
      O: o453 (perceive-search-criterion)
      O: o459 (evaluate-new-items-nn)
      O: o463 (get-location-of-command)
      O: o334 (final-state)
   O: o480 (bring-up-help-text)
   ==>G: g482 (operator no-change)
      P: p488 (bring-up-help-text)
      S: s499
      O: o501 (move-mouse-to-item-location)
      O: o506 (click-mouse-button)
      O: o498 (final-state)
   O: o544 (evaluate-new-items)
   O: o558 (bring-up-help-text)
   ==>G: g560 (operator no-change)
      P: p566 (bring-up-help-text)
      S: s577
      O: o579 (move-mouse-to-item-location)
      O: o584 (click-mouse-button)
      O: o576 (final-state)
   O: o606 (evaluate-new-items-nn)
   O: o211 (final-state)
O: o615 (evaluate-help-text)
==>G: g617 (operator no-change)
   P: p623 (evaluate-help-text)
   S: s632
   O: o634 (focus-on-help-text)
   O: o638 (find-pointer-to-criterion)
   O: o631 (final-state)
O: o655 (search-for-criterion)
==>G: g657 (operator no-change)
   P: p663 (search-for-criterion)
   S: s682
   O: o684 (select-search-method)
   ==>G: g686 (operator no-change)
      P: p692 (select-search-method)
      S: s706
      O: o708 (choose-window)
      O: o716 (focus-on-current-window)
      O: o721 (evaluate-current-window)
      O: o755 (choose-method)
      O: o761 (choose-direction)
      O: o705 (final-state)
   O: o777 (execute-search-method)
   ==>G: g779 (operator no-change)
      P: p785 (execute-search-method)
      S: s801
      O: o803 (move-mouse-to-above-elevator)
      O: o808 (click-mouse-button)
      O: o825 (evaluate-new-items-nn)
      O: o808 (click-mouse-button)
```

```
            O: o840 (perceive-search-criterion)
            O: o846 (evaluate-new-items-nn)
            O: o850 (get-location-of-command)
            O: o800 (final-state)
         O: o867 (bring-up-help-text)
         ==>G: g869 (operator no-change)
            P: p875 (bring-up-help-text)
            S: s886
            O: o888 (move-mouse-to-item-location)
            O: o893 (click-mouse-button)
            O: o885 (final-state)
         O: o915 (evaluate-new-items-nn)
         O: o681 (final-state)
      O: o924 (evaluate-help-text)
      ==>G: g926 (operator no-change)
         P: p932 (evaluate-help-text)
         S: s941
         O: o943 (focus-on-help-text)
         O: o949 (find-disconfirmation)
         O: o940 (final-state)
      O: o972 (search-for-criterion)
      ==>G: g974 (operator no-change)
         P: p980 (search-for-criterion)
         S: s999
         O: o1001 (select-search-method)
         ==>G: g1003 (operator no-change)
            P: p1009 (select-search-method)
            S: s1023
            O: o1025 (choose-window)
            O: o1031 (focus-on-current-window)
            O: o1036 (evaluate-current-window)
            O: o1022 (final-state)
         O: o1126 (bring-up-help-text)
         ==>G: g1038 (operator no-change)
            P: p1128 (bring-up-help-text)
            S: s1045
            O: o1139 (move-mouse-to-item-location)
            O: o1144 (click-mouse-button)
            O: o1137 (final-state)
         O: o1166 (evaluate-new-items-nn)
         O: o998 (final-state)
      O: o1175 (evaluate-help-text)
      ==>G: g1050 (operator no-change)
         P: p1177 (evaluate-help-text)
         S: s1052
         O: o1186 (focus-on-help-text)
         O: o1190 (find-criterion)
         O: o1184 (final-state)
      O: o71 (final-state)
   O: o48 (final-state)
O: o6 (halt)
```

(end of Figure 6. Browser-Soar trace)

3. The Database

As stated previously, the goals of the database are to relate several levels of protocol analysis, from raw protocol transcription data through the most detailed cognitive simulation traces, and to allow complex database manipulation to perform automatic classification of the data. The prototype database relates a raw protocol transcription of a segment of browsing behavior (Figure 3) to a trace of the Soar model of that behavior (Figure 6).

The prototype database application, called Trace&Transcription, was built in the Oracle Relational Database Management System.[2] We chose Oracle because it seemed to have the best mix of power and performance with ease-of-use and standardization of the relational databases. Oracle can be used either as a stand-alone application on a personal workstation (the prototype works on an Apple Macintosh), or as a multi-user database on a central processor with individual access through networked workstations. It provides the power of the industry standard query language, SQL. On the Macintosh, Oracle provides a HyperCard[3] interface for loading data in many formats (SQL*Loader Utility), constructing queries (Query Tool), and creating other HyperCard application interfaces (Application Generator) (see Oracle for Macintosh User's Guide, Version 1.2, Oracle Corp, 1990). Trace&Transcription was built using HyperCard and Oracle's HyperCard interface to the database.

3.1 Organization of the Database

The database has one table for the raw protocol transcription data and one additional table for each level of analysis performed on the protocol data; in the case of the prototype database, there is only one additional table for Soar traces. These analysis tables can be created by hand by the analyst, or they can be created automatically by sophisticated query techniques provided by Oracle. This prototype uses only hand-created analysis tables.

Each record in the protocol transcription table represents a one second time period, during which a verbal utterance has occurred, a mouse action has occurred (move, click, press-button or release-button), neither, or both have occurred. Each record in the Soar trace table represents a single decision cycle for Browser-Soar which terminated in the selection of a goal, problem space, state, or operator. A useful relationship between the two tables is to link a Soar decision cycle to the behavior that is direct evidence for the action produced by that decision cycle, e.g., an observed mouse-button click would be direct evidence for the selection of the Soar operator **click-mouse-button**. This is a symmetric relationship, where the observed behavior is evidence for application of Soar operators, and the selection of Soar operators provide a simulation of human behavior. Our database application, Trace&Transcription, allows a user to explicitly create these links and the database records them by creating a third table for pointers between the two user-provided tables.

---

[2]Trademark of the Oracle Corporation.
[3]HyperCard is a trademark of Apple Computer, Inc.

The TRANSCRIPTION table has six columns:

BEHAVIOR_NUMBER
- Type: CHAR
- Size: 10
- Required? no
- Description: This is the unique number of the single behavior in one line of the transcription of this segment of behavior. It ends with the character "v" if it is a verbal utterance, or "m" if it is motor action like mouse movement or button clicking.

TAPE_TIME
- Type: NUMBER
- Size: 12
- Required? yes
- Description: This is the time in milliseconds since the beginning of the tape. This number appears in the bottom right-hand corner of the videotape. Each new record starts one second (1000 msec) after the record preceding it.

VERBAL
- Type: CHAR
- Size: 100
- Required? no
- Description: This is the verbal utterance that happened during the one second (1000 msec) after the TAPE_TIME of this record, if any.

POINTING
- Type: CHAR
- Size: 100
- Required? no
- Description: This is the motor action that happened during the one second (1000 msec) after the TAPE_TIME of this record, if any.

TRANSCRIPTION_ID
- Type: CHAR
- Size: 10
- Required? yes
- Description: This is the number of the transcription segment. In the prototype database, this number is always 1 and the loading program provides this constant.

TAPE_ID
- Type: CHAR
- Size: 10
- Required? yes
- Description: This is the number of the videotape. In the prototype database, this number is always 1 and the loading program provides this constant.

The SOAR table has four columns:

LINE_NUMBER
- Type: NUMBER
- Size: 10
- Required? yes
- Description: This is the unique number for each line in this Soar trace. It is provided by the loading program.

DECISION_CYCLE
- Type: NUMBER
- Size: 10
- Required? yes
- Description: This is the decision cycle of the Soar trace.

TEXT
- Type: CHAR
- Size: 200
- Required? no
- Description: This is the text of the Soar trace, e.g., "O: o342 (press-mouse-button)"

SOAR_ID
- Type: CHAR
- Size: 10
- Required? yes
- Description: This is the number of the Soar trace. If several variants of a Soar model are being evaluated, there may be multiple traces that map to the same protocol data. In the prototype database, this number is always 1 and the loading program provides this constant.

The POINTER table has seven columns, automatically created in the process of using Trace&Transcription to explicitly provide links between transcription and trace:

TAPE_ID - taken from the transcription record (required)
TRANSCRIPTION_ID - taken from the transcription record (required)
TAPE_TIME - taken from the transcription record (required)
BEHAVIOR_NUMBER - taken from the transcription record (required)
(together the above four items point to a unique transcription record)

SOAR_ID - taken from the Soar trace record (required)
LINE_NUMBER - taken from the Soar trace record (required)
(together the above two items point to a unique Soar trace record)

FORCED_ALIGNMENT
- Type: CHAR
- Size: 1
- Required? yes
- Description: This is either "Y" or "N", where "Y" means that the unique transcription record should be spatially aligned with the unique Soar trace record when it is displayed in the database application, and "N" means that the two are related but are not to be spatially aligned in the display.

3.2 Operation of the Database Application Relating Transcriptions to Traces

This section describes the operation of the Trace&Transcription database application written for Oracle;[4] it is not intended as an instruction manual for the operation of the Oracle Relational Database Management System itself. Please refer to the Oracle for Macintosh User's Guide Version 1.2 where referenced in this section. We assume that the reader is familiar with the installation, logon procedure and use of Oracle and HyperCard.

3.2.1 Loading Protocol and Soar Trace Data

The protocol transcription data and Soar traces must be loaded into their respective tables before the Trace&Transcription application can be used. Before loading data, the tables described above must be created in the Oracle database on which Trace&Transcription is to be run. Refer to Oracle User's Guide Chapter 2, Managing Tables and Views, pages 2-9 through 2-15, for instructions for creating tables. Then refer to Chapter 2, Loading Data with SQL*Loader, pages 2-30 through 2-36, for instructions for loading the data. The following four files must be resident on the host computer to load the transcription and trace data:

transcription.ctl, a file used by Oracle to control the loading of the transcription data. The content of transcription.ctl is shown in Figure 7. The INFILE line must be edited to conform to the names of the disk and file structure of the host computer.

transcription.dat, a text file with the transcription data in the format described by transcription.ctl. An example of the protocol transcription data is shown in Figure 8.

soar.ctl, a file used by Oracle to control the loading of the Soar trace data. The content of soar.ctl is shown in Figure 9. The INFILE line must be edited to conform to the names of the disk and file structure of the host computer.

soar.dat, a text file with the Soar trace in the format described by soar.ctl. An example of the Soar trace data is shown in Figure 10.

```
LOAD DATA
INFILE "<disk name>:<folder>:<subfolder>:<filename>.dat"  STREAM
INTO TABLE transcription replace
FIELDS TERMINATED BY ","
OPTIONALLY ENCLOSED BY '"'
( behavior_number          CHAR,
  tape_time                INTEGER EXTERNAL,
  verbal                   CHAR,
  pointing                 CHAR,
  transcription_id         CONSTANT 1,   -- Constant only in prototype
  tape_id                  CONSTANT 1
)
```

Figure 7. The file that allows Oracle to load protocol transcription data, transcription.ctl

[4] At the initial writing of this report in the winter of 1990, Trace&Transcription was available as a HyperCard stack. This prototype is no longer available, but a subsequent system has been built upon several of the interface design ideas (Ritter, 1992; Ritter & Larkin, in press).

```
V1,15705000,OK,""
V2,15706000,so,""
,15707000,,""
V3,15708000,so I just want to,""
,15709000,,""
,15710000,,""
,15711000,,""
,15712000,,""
V4,15713000,make a mark,""
,15714000,,""
V5,15715000,I want to draw something,""
V6,15716000,I want to circle something,""
,15717000,,""
V7,15718000,guess I want to put a little circle in,""
M8,15719000,,"M to help screen"
```

Figure 8. First 10 lines of the protocol transcription data file; example of the format

```
LOAD DATA
INFILE "<disk name>:<folder>:<subfolder>:<filename>.dat"
INTO TABLE soar replace
FIELDS TERMINATED BY ","
OPTIONALLY ENCLOSED BY '"'
(Line_Number          recnum,
 Decision_Cycle       INTEGER EXTERNAL,
 Text                 CHAR(200),
 Soar_id              CONSTANT 1  -- Constant only in prototype
)
```

Figure 9. The file that allows Oracle to load Soar trace data, soar.ctl

```
0,  "G: g1 "
1,  "P: p4 (top-space)"
2,  "S: s5 "
3,  "O: o12 (browsing-task)"
4,  "==>G: g36 (operator no-change)"
5,  "   P: p42 (browsing)"
6,  "   S: s49 "
7,  "   O: o51 (find-appropriate-help)"
8,  "   ==>G: g53 (operator no-change)"
9,  "      P: p59 (find-appropriate-help)"
10, "      S: s72 "
```

Figure 10. First 10 lines of the Soar trace data file; an example of the format

3.2.2 Running the Trace&Transcription Application

After the tables are created, the data are loaded, and Oracle is running, Trace&Transcription can be run. To start Trace&Transcription from Oracle's Log On card, click on the **Home** icon, and enter Trace&Transcription through the dialog box of the **mystacks** icon. Oracle maintains security of the database by forcing the user to log on each time you enter the database through any application, so you must log onto Oracle again at this point. The Trace&Transcription card then appears (Figure 11).

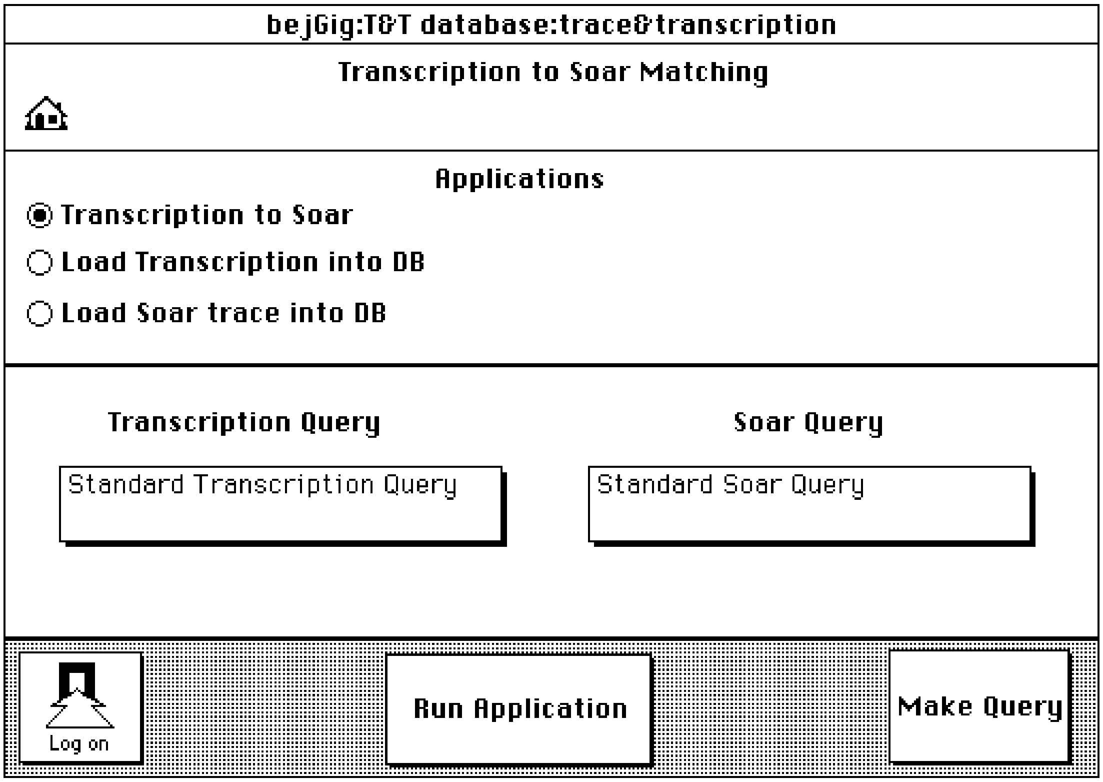

Figure 11. Trace&Transcription application card.

The top section of the Trace&Transcription card allows access to several related applications, selected by clicking on the radio buttons to the left of their names. The first, **Transcription to Soar Trace** is the application that allows the user to link observed behaviors to Soar decision cycles (and is the only one implemented in the prototype). The others, **Load Transcriptions into DB** and **Load Soar Trace in to DB**, are future applications that load data directly from the natural format of the data files rather than having to put them into a format acceptable to Oracle, as must be done with the prototype (section 3.2.1 of this report). That is, these applications will strip off standard file headers, provide correct indentation, and do other formatting customized for the transcription and Soar trace files.

The middle section of the Trace&Transcription card, when the **Transcription to Soar Trace** application is selected, allows access to pre-stored queries. Clicking on the boxes below the **Transcription Query** or **Soar Trace Query** headings, causes a dialog box to appear so that a different pre-stored query can be selected. Making and storing queries is discussed in the next section (section 3.2.3).

The bottom section of the Trace&Transcription card provides button to **Log On**, **Run Application** or **Make Query**. **Log On** allows another user to log on, if desired. **Make**

Query takes you to the card shown in Figure 16 and will be discussed in the next section. **Run Application** performs the queries previously selected and displays the results of those queries on the **Trace&Transcription Information** card, shown in Figure 12.

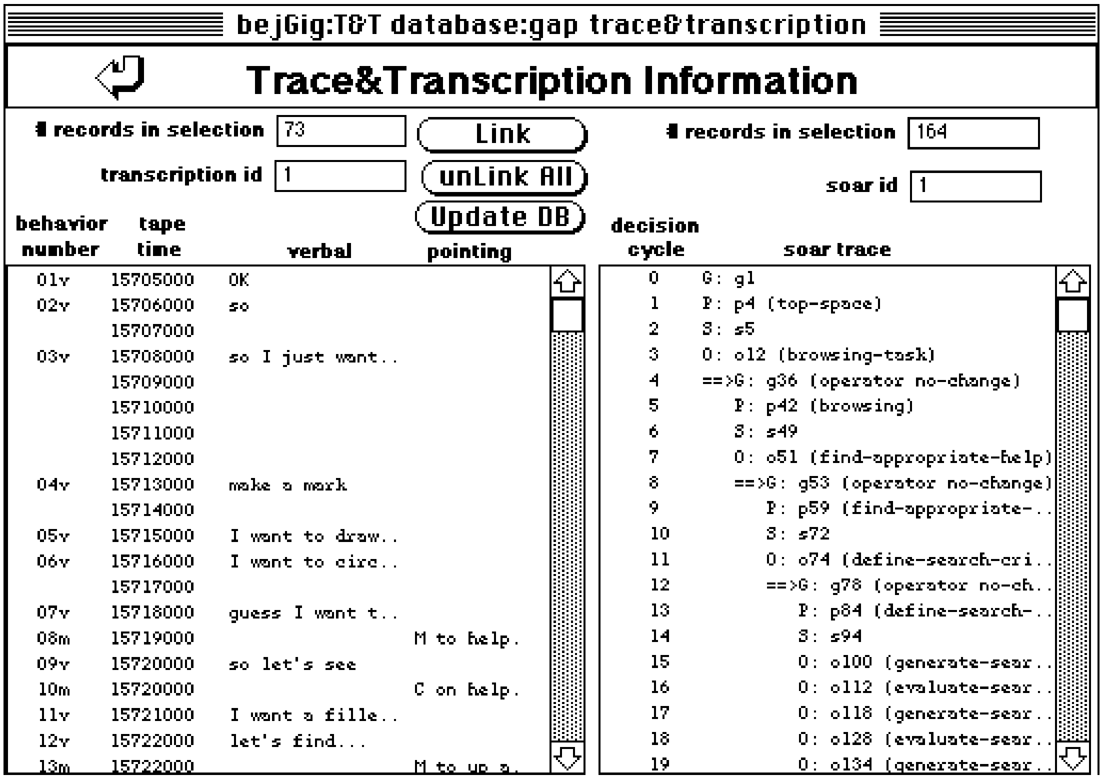

Figure 12. Trace&Transcription Information card.

The Trace&Transcription Information card displays the results of the selected queries with the protocol transcription data appearing on the left-hand side and the Soar trace data appearing on the right-hand side. Initially, there are no links between the transcription records and the trace records, so these windows are not yoked and they scroll separately.

The function of the Trace&Transcription application is to allow an analyst to create links between the records of the protocol transcription table representing individual observed behaviors and the records of the Soar trace table representing Soar decision cycles.

There are two types of links necessary to make the relationships between these tables apparent to the analyst. An observed behavior might have a one-to-one mapping to a Soar decision cycle, that is, the simulation of a particular behavior is performed by a single Soar operator and the empirical evidence for that Soar operator is that particular operator. When this relationship occurs, then it is often convenient to align the behavioral record with the Soar decision cycle, allowing the analyst to see the one-to-one mapping. This type of link, called a forced_alignment, is created by clicking the mouse on the desired behavioral record (a black circle will appear on the far left of the transcription line), then clicking on the desired Soar decision cycle (a black circle will appear on the far left of the Soar line), and then clicking on the **Link** button. The displays will align accordingly and arrowheads facing each other will appear in the center of the screen indicating the link (Figure 13). When a forced_alignment link is created, the two screens become yoked, and scrolling one screen will automatically scroll the other screen.

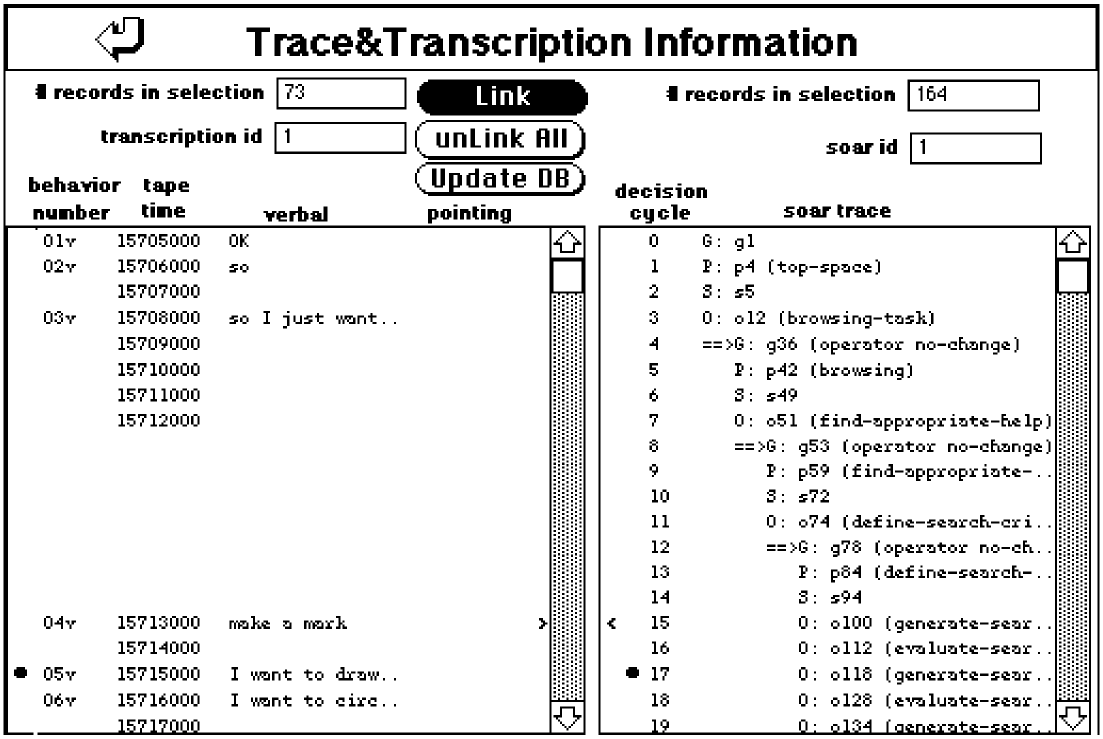

Figure 13. Example of a forced_alignment link (04v & dc15). 05v and dc17 show the black dots that indicate they will also be linked with a forced_alignment link after the Link process is complete.

The other type of link is needed when forced_alignment would cause conflict in aligning the data. This can occur in two cases: when linked behaviors and decision cycles occur out of order (i.e., a behavior linked to a later decision cycle occurs earlier in time than a behavior linked to an earlier decision cycle) and when more than one behavioral record should be linked to a single Soar decision cycle. To create a link that does not force alignment on the screen, hold the **shift** key down when clicking on the behavior and the decision cycle (dashes will appear next to the selected lines, rather than the black circles in the forced_alignment selections) and then click on the **Link** button. Numbers will appear in the center of the screen indicating the link (Figure 14). If more than one behavior in linked to a single Soar decision cycle, the same number will appear on each line of the multi-line behavior to indicate this many-to-one link (Figure 15).

These links are made in the Trace&Transcription card only until the analyst clicks on the **Update DB** button. Then these links are sent to the POINTERS table of the database, and will come back into Trace&Transcription whenever the linked behaviors or decision cycles are accessed. It is good practice to update the database after every few links are created.

The **UnLink All** button removes all the links currently in Trace&Transcription, including both the ones permanently in the database and those only in the card (not yet updated). This is a powerful button, useful in a prototype with only a few links. However, it is probably too powerful for routine use in a full-fledged database and a more local undo function should be added to this card.

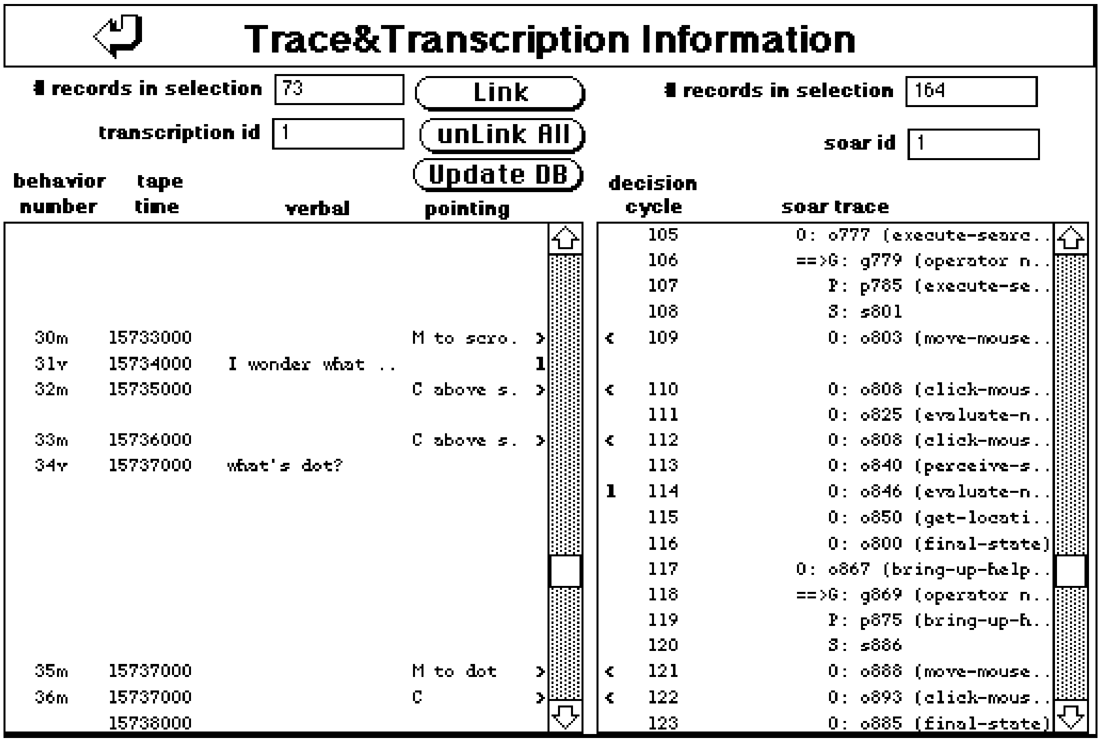

Figure 14. Example of a link without forced alignment, necessary because the linked behaviors and decision cycles are out of order with other links. In this example, decision cycle 114 in the Soar trace window is linked to behavior number 31v in the transcription window, with the intervening links for behavior 32m and dc110 and 33v and dc112 that would conflict if this were a forced_alignment link.

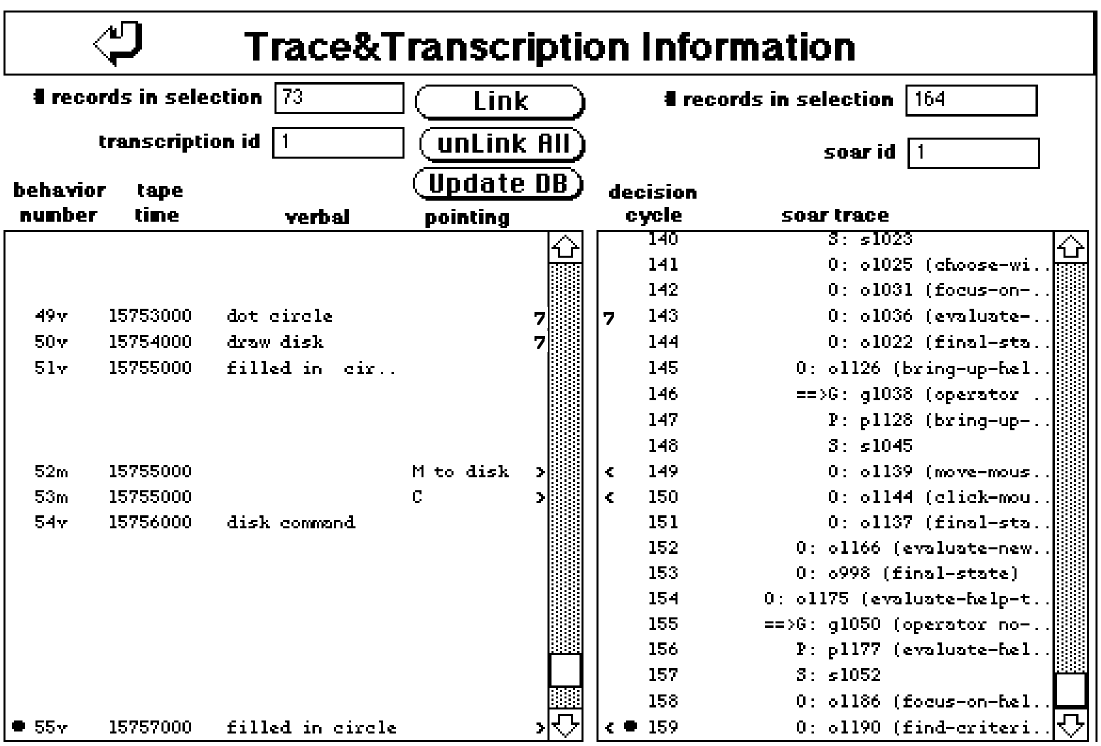

Figure 15. Example of a many-to-one link without forced alignment. Behaviors V49 and V50 are linked to decision cycle 143.

3.2.3 Making Queries

Once the protocol transcription data are linked to the Soar trace data, an analyst may want to make complex queries of the database to explore these relationships. For instance, in evaluating the goodness of fit of a cognitive simulation like a Soar model, it is desirable to know how many individual overt behaviors are explicitly simulated, and how many aren't. More specific information about the goodness of fit can be had by looking at what percentage of verbal behaviors are, and are not, explicitly simulated, vs. the same statistics for the motor behaviors. More qualitative, but perhaps more useful in understanding the failings of a model is to display those behaviors not simulated and some of the behavioral context leading up to each unsimulated behavior. These questions can be answered with this Oracle database through the power of SQL queries.

To build a query and store it in the database for repeated use, the analyst clicks the **Make Query** button on the Trace&Transcription application card. This action brings the analyst to a card with 3 choices; Construct New Query, Import Saved Query from Oracle Query Tool, and View & Modify Query in DB.

To create a brand new query click on the button next to "Construct New Query". This brings up the Oracle's Query Tool (Chapter 3 in the Oracle User's Guide). Once the query is built using the tool, then store this query in an ascii file by clicking on the **Report** button, then on the write-to-disk icon on the Report card (second one down on the far left of the card).[5] Return to the **Make Query** card by going through HyperCard Home, and Trace&Transcription. The stored file can then be imported into the Trace&Transcription application by clicking on the button next to "Import Saved Query from Oracle Query Tool" (see the next paragraph).

The button next to "Import Saved Query from Oracle Query Tool" puts up a dialog box that requests the filename for a previously created ascii query (see previous paragraph). Once a file is selected, the query is extracted and is shown in the scrolling field (Figure 16). The name of the source file is displayed above the query. The query can be modified using all the standard Macintosh text manipulation techniques. All the changes to this query are local to this card, until the **Update DB** button is clicked. First, enter the name the query should be stored under in the **Save...As...** field above the scrolling field. Then the **Update DB** button puts the query in the scrolling field into the Trace&Transcription database, in a QUERY table ,under that name (it does not change the original ascii file). At any time, the query can be run in the Trace&Transcription application by clicking on the **Run Application** button.

Queries can include either constants or variables. It is often easier to create a query with a particular example in mind, putting constants into the query and testing it with this example. Then the constants can be replaced by variables. Then, at run time, the user is asked for a value for each variable in the query, making the query much more general. To make a constant into a variable, select the constant in the scrolling field and click on the **Make Variable** button. The selected constant will then be replaced with a variable of the form %%VARx%%, where x is a number between 1 and 10, and a dialog box will appear. The dialog box will ask for a name to label the variable, to use to ask for the value of the variable at run time. The original constant will become the default value of the variable. To make a variable back into a constant, select the variable and click the **Remove Variable** button. This replaces the variable with its default value and releases the variable

---

[5] Do not use the Save Query button to store queries because that button saves them in a format unreadable by Trace&Transcription.

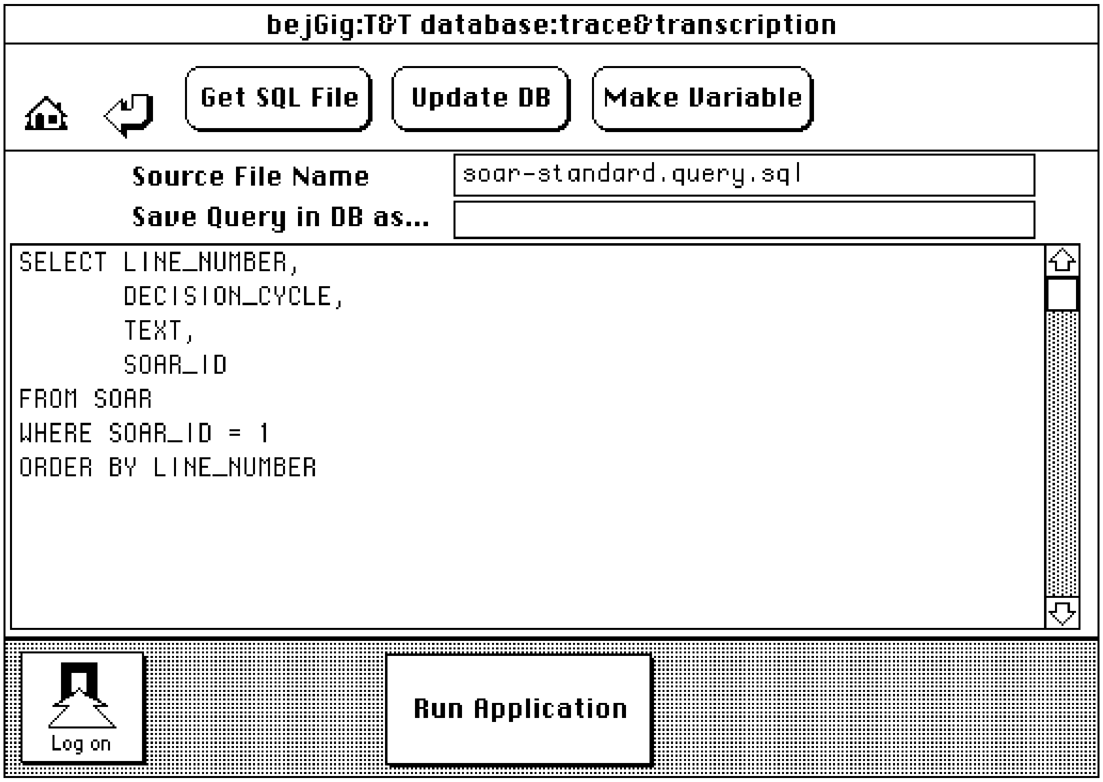

Figure 16. Import Saved Query from Oracle Query Tool card.

for use in other places in the query. (Trace&Transcription currently has a 10-variable limit per query, so releasing the variable may be important.)

The last choice of action on the Make Query card is "View & Modify Query in DB". This button presents the same interface to the user as the "Import Saved Query..." button. However, it allows access to queries already stored in the Trace&Transcription database, rather than queries externally created by the Oracle Query Tool. As before, the changes made to the query are only local to the card until **Update DB** is clicked. As before, the query can be run in the Trace&Transcription application directly from this card with the **Run Application** button.

Acknowledgments

Special thanks to Gary Pelton and Anna Zacherl for their efforts in designing and implementing this prototype database.

References

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

John, B. E., Newell, A., & Card, S. K. (1990) *Browser-Soar: A GOMS-like model of a highly interactive task*. Talk presented at the Human Computer Interaction Consortium Winter Workshop, San Diego, CA, February 12, 1990.

Lewis, R. L., Huffman, S. B., John, B. E., Laird, J. E., Lehman, J. F., Newell, A., Rosenbloom, P. S., Simon, T., & Tessler, S. T. (1990) "Soar as a unified theory of cognition: Spring 1990." *Proceedings of the Twelfth Annual Conference of Cognitive Science Society*, July, 1990.

Peck, V. A. & John, B. E. Browser-Soar: A cognitive model of a highly interactive task. In proceedings of CHI, 1992 (Monterey, California, May 3- May 7, 1992) ACM, New York, 1992. pp. 165-172.

Ritter, F. E. (1992). "TBPA: A methodology and software environment for testing process models' sequential predictions with protocols." Doctoral dissertation, Dept. of Psychology, Carnegie Mellon University. Also available as School of Computer Science Technical Report No. CMU-CS-93-101.

Ritter, F. E. & Larkin, J. H. (in press) "Developing process models as summaries of HCI action sequences, *Human-Computer Interaction*.

Newell, A. (1990) *Unified Theories of Cognition*. Harvard University Press, Cambridge, Mass.