Chapter 7

Performance demonstration I: Analyzing the Browser-Soar model faster and more deeply

Browser-Soar (Peck & John, 1992) is a model of a user using an on-line help system. Ten episodes totalling approximately ten minutes of a single subject's behavior have been used to test it. This chapter examines Browser-Soar in detail, duplicating and extending the previous set of analyses. By choosing to duplicate and extend an existing analysis, it includes a set of analyses known to be useful, and provides a reference point for measuring its speed and discrimination.

Soar/MT allows the sequential predictions of Browser-Soar to be tested more quickly and at a finer level than can be performed by hand using sheets of paper and a plain spreadsheet (Excel) to hold the correspondences. Displays showing the fit of the data to the predictions can be easily created, and provide additional insight for characterizing the model and how to improve its predictions.

Browser-Soar provides predictions of when structures enter working memory. This allows testing the sequentiality assumption of verbal protocol theory, that mental structures are reported on in the order that they appear in working memory. This assumption is found to hold for verbal utterances. Sequentiality can also be tested for mouse actions and they too are performed in order. However the verbal utterance and mouse action information streams do not initially appear to be sequential with respect to each other. The most likely cause for this discrepancy is that an approximation in the interpretation and alignment process was used. The data in the two information streams should be considered sequential. When this is done, the overt actions provide fixed reference points for computing the lag of the verbal utterances.

With a measure of the model's performance and fit in hand, a small modification of Browser-Soar suggested by the measures of fit is attempted, removing some problem spaces that might be redundant. This change does not drastically improve the fit, and this is shown clearly in the analytic displays. The resulting model, however, is more parsimonious with the effects of learning.

In nearly every case the results reported here duplicate what Peck and John already know about Browser-Soar, but they come at less expense and can be shown more compellingly with Soar/MT's displays.

7.1 Description of Browser-Soar and its data

While Browser-Soar and its data are explained fully elsewhere (Peck & John, 1992), an overview is presented here with particular emphasis on the aspects of the data and model that receive attention in this reanalysis. Because Peck and John have generously allowed me access to their original data, I am able to include additional descriptions of the data here.

Description of the data. The Browser-Soar data used to test Browser-Soar was gathered from a single user interacting with the cT programming environment on the Macintosh computer (Sherwood & Sherwood, 1984; Sherwood & Sherwood, 1992) to perform her own task arising out of her work, creating a graphing program for her own use. She was a non-professional but experienced computer programmer who had never used cT before the experiment. The episodes of interest occurred when she used the on-line help system to learn about cT. The data that Browser-Soar is tested with represents only a portion of the 85 episodes using the help browser that occurred during the three and one-half hours of behavior that was videotaped. A portion of the remaining data was used informally to help create the model.

The first four episodes were chosen to cover a wide variety of browsing behavior, and the remaining six were chosen randomly from all the browsing episodes that were videotaped. Each episode represents a different environment and goals.

As noted in Table 7-30, the ten episodes averaged 56 s in length and included a total of 58 verbal and non-verbal segments. Each episode included a mean of 126 words. This was not reported in the original analysis because computing the number of words per episode is something that is not easy to do, at least given a journeyman's familiarity with Excel.[8] The subject's behavior provided a high density of data, on average, over an action or utterance every second, and a relatively high verbalizations rate of 146 words per minute.

Three information streams from the subject were transcribed by hand into Excel spreadsheets, the verbal utterances, the mouse movements, and the mouse clicks. Verbal utterances were broken into separate segments when pauses of more than 100 ms occurred.

There are three special features of the Browser-Soar data worth noting. The first is that the types of mental information reported is small. The user generally only mentions search criteria, evaluation criteria, and the words that she is reading. Second, this is not a hard problem. In contrast to many tasks that have been modeled, using the computer interface is routine behavior for the subject and their internal representation of the task is not changing. Finally, there is what will turn out to be a useful mix of overt, necessary task actions (mouse actions) with verbal statements. The overt, motor actions will help disambiguate the verbal, and vice-versa.

Description of the Browser-Soar model. Figure 7-33 depicts the problem spaces in Browser-Soar and their relation to each other as drawn with the SX graphic display. Figure 7-34 presents the problem spaces as depicted by Peck and John (1992). All the problem spaces are related by operator no-change impasses except the *Selection* problem space, which is used to resolve operator ties in the *Find-criterion* space. Browser-Soar does not reuse any problem spaces, so the maximum goal depth will be six, not counting the top-goal.

The Soar learning mechanism is not turned on in the Browser-Soar model. Peck and John (1992) argued that there will be little learning observable in this set of tasks. The user is either performing as an expert, that is, will not be learning how to move the mouse, or is learning items that will not transfer between trials, such as how to print out a variable's value.

Based on a sample run (the "Write" episode, the subject was seeking information about writing information onto the cT screen) and assigning the productions to problem spaces graphically, the problem space level statistics function in the SX graphic display reports that there are a total of 18 problem spaces and at least 31 operators. The statistics based on a complete run are shown in Table 7-29.

Browser-Soar is actually a short progression of models based on testing and modify it with the ten episodes. During this progression Browser-Soar remained rather stable. Between the first and the tenth episode, two operators were added to Browser-Soar, and four operators application conditions were changed. Because there are so few adjustments, in this analysis Browser-Soar can be treated as a single program.

Comparing the listing of problem spaces in Table 7-29 with Peck and John's (1992) listing, it appears that either they do not include all the Macintosh method problem spaces, or the organization of Browser-Soar has changed since it was reported. Table 7-29 includes an additional operator more than reported by Peck and John (1992) (probably several more, because four of the *Mac-method-** problem spaces also would have operators). This missing operator could be the *Browsing-task* operator itself, or it could be an operator used in two spaces, which the SX graphic display would count twice. The difference in object counts is compounded by Peck and John's treatment of the model. They knew that they did not have an adequate model of reading, and did not attempt to match the model's behavior below reading the whole screen.

[8]Macros can, however, be created to perform this task (Schroeder, 1992).

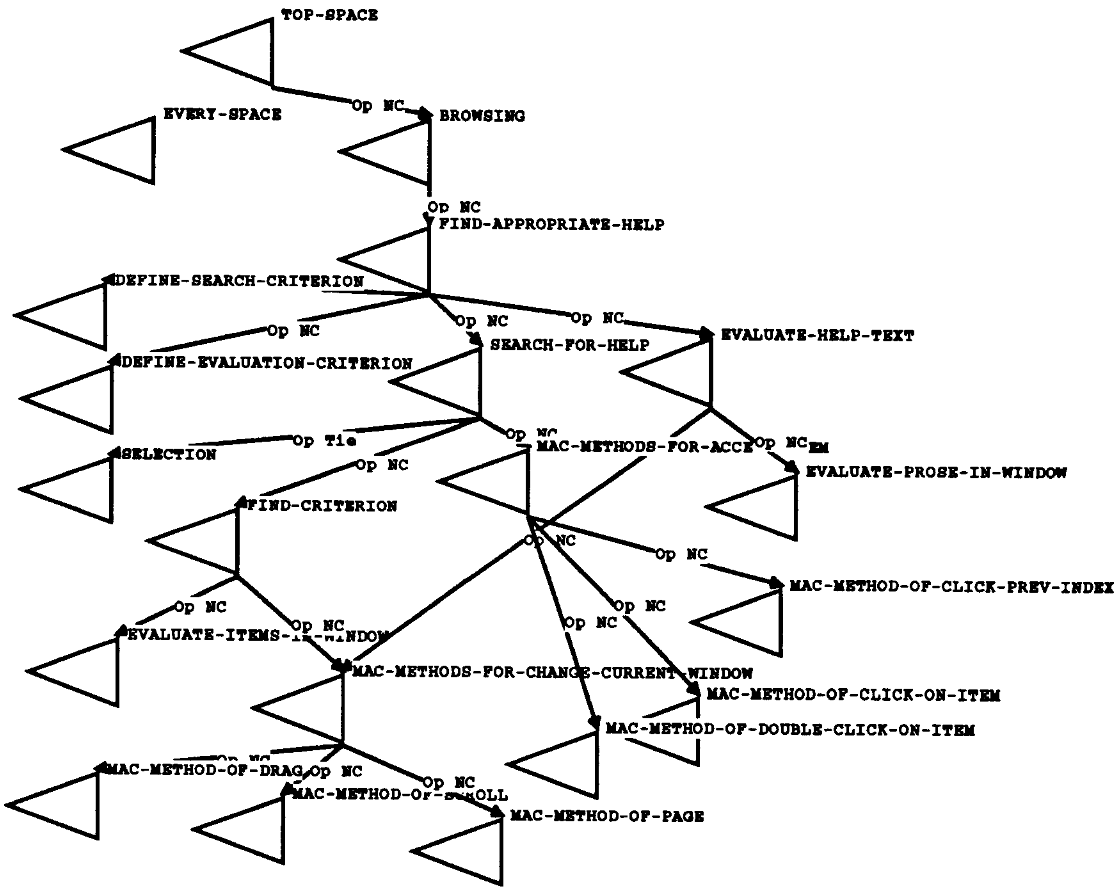


Figure 7-33: The problem space organization of the 19 problem spaces in Browser-Soar generated with the SX graphic display.

The static structure depicted in Figure 7-33 shows the normal dynamic selection and use of the problem spaces. It was created by loading Browser-Soar and running an episode. The problem spaces that were created were then rearranged from their location on a grid to the tree structure shown in the figure, connected together by hand, and annotated. Their organization was written out so that this structure could be used again.

Figure 7-35 shows the goal stack in Browser-Soar at decision cycle 17 of the Write episode. The selection and use of problem spaces moves roughly from top to bottom and left to right. At the start of the browsing episode, the Browsing space is selected and the *Find-appropriate-help* operator is applied. This cannot be directly implemented, so the *Find-appropriate-help* problem space is selected. Within this problem space, the operators *Define-search-criterion, Define-evaluation-criterion* are called to initialize the search. Both of these operators cannot be directly applied, and similarly named problem spaces are used to implement them.

The *Search-for-help* operator is applied once the search and evaluation criteria are defined. This operator also cannot be directly implemented, and the *Search-for-help* problem space is selected. Within this problem space, operators (and corresponding problem space to implement them) are applied to search the help screen (*Find-criterion*), and to select an interesting item to read about if it is found (*Mac-methods-for-accessing-item*). When searching for interesting items, the *Find-criterion* problem space uses two operators, one to evaluate items in the window, and one to scroll the screen

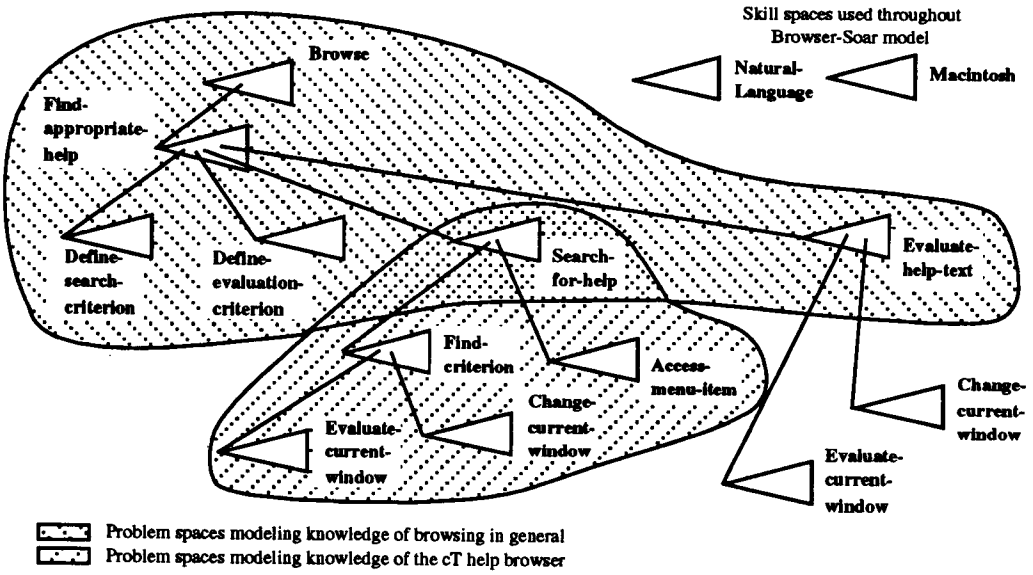


Figure 7-34: The problem space organization of Browser-Soar taken from Peck and John (1992).

when the end of the screen of items is reached (*Mac-methods-for-changing-window*). Both operators are implemented in their own space.

Once an item has been found and selected by clicking on it, the *Evaluate-help-text* operator is selected in the *Find-appropriate-help* problem space. This operator is implemented in its own space using two operators. The first operator selected will be to evaluate the help text by reading it. The other operator is the same scroll used to scroll the window of items to select from.

When they did their analyses, Peck and John grouped two of the Soar operators in *Evaluate-items-in-window* problem space that implement reading the computer screen into a higher level operator, *Evaluate-current-window*, not shown in the automatically derived figures. In Peck and John's operator support displays, the low-level operators, such as *Read-input*, do not appear, for all the coding was based on the higher level operator. This coding scheme was duplicated in the later analyses that are reported here, except that the lower level operators and problem spaces do appear in the automatic display and aggregate model statistics.

Figure 7-36 shows the number of productions used to implement each problem space. Approximately 430 productions are generated from the 193 TAQL constructs used to create these problem spaces. The exact numbers varied slightly between episodes. When the productions were sorted into problem spaces, a problem space was found for most productions. Productions and TAQL constructs that are included as part of Soar's default knowledge are not included in the counts or the display.

Productions without a problem space name directly in their condition were assigned to *Every-space*, 29 in all. *Every-space* is used to display productions that could fire in every space. Examining these with the graphic display indicated that 15 of them are for proposing new problem spaces based solely on the goal and its superstate, 12 are internal TAQL productions, one is used to note that all search-for-help operators are equivalent, and one prints out the search criteria whenever it changes.

The number of productions associated with each problem space is an approximate measure of the amount of knowledge in each problem space. One of the reasons this measure is approximate is because TAQL uses a production for each of its state edits. Only the user's productions are included in this display, so the lack of productions associated with the *Selection* space means that it only uses the

Table 7-29:
Problem space level statistics for the "Write" episode. The top block presents the problem spaces and operators represented in the graphic display. The selection counts for each goal, problem space, state, and opeator are presented in their hierarchical calling order.

```
PSCM Level statistics on November 27, 1992

18 problem spaces, with a total of 31 operators.
Ops  Problem space
 1   top-space
 1   browsing
 5   find-appropriate-help
 2   define-search-criterion
 2   define-evaluation-criterion
 2   search-for-help
 3   find-criterion
 2   evaluate-items-in-window
 1   mac-methods-for-change-current-window
 3   mac-method-of-scroll
 1   mac-methods-for-access-item
 2   mac-method-of-click-on-item
 3   evaluate-help-text
 3   evaluate-prose-in-window
 0   mac-method-of-drag
 0   mac-method-of-page
 0   mac-method-of-click-prev-index
 0   mac-method-of-double-click-on-item


The actual selection counts and calling orders:
 1 G: g1 (g1)
 1 .P: top-space (top-space) (3 chunks)
 1 . S: s5 (no name)
 1 . O: browse (browse)
 1 .  G: (operator no-change) (g19)
 1 .  .P: browsing (browsing) (16 chunks)
 1 .  . S: s39 (no name)
 1 .  .  G: (state no-change) (g3145)
 1 .  .  .G: (goal no-change) (g3152)
 1 .  .  . G: (goal no-change) (g3159)
 1 .  .  .  G: (goal no-change) (g3166)
 1 .  .  .  .G: (goal no-change) (g3173)
 1 .  .  .  . G: (goal no-change) (g3180)
 1 .  . O: find-appropriate-help (find-appropriate-help)
 1 .  .  G: (operator no-change) (g43)
 1 .  .  .P: find-appropriate-help (find-appropriate-help) (55 chunks)
 1 .  .  . S: s59 (no name)
 1 .  .  . O: define-search-criterion (define-search-criterion)
 1 .  .  .  G: (operator no-change) (g65)
 1 .  .  .  .P: define-search-criterion (define-search-criterion) (30 chunks)
 1 .  .  .  . S: s79 (no name)
 1 .  .  .  . O: generate-search-criterion((write)) (generate-search-criterion)
 1 .  .  .  . O: evaluate-search-criterion (evaluate-search-criterion)

(continued on next page)
```

default productions provided with Soar. It appears that it takes a minimum of three user productions to create a usable problem space.

Browser-Soar interacts with a simulation of the cT help browser. The simulation provides Browser-Soar with the contents of each window in the browser. The simulation does not take into account the length of time a mouse is held down; on each mouse click it scrolls to the same place the subject scrolled to in the same situation. If the model were to scroll in the wrong direction (which it no longer does, and perhaps never did), it would be up to the analyst to catch this.

Description of original Browser-Soar analyses. Peck and John's (1992) originally performed the alignment by hand, aggregating the correspondences into summary measures for each episode and for each operator. They used limited graphic displays of the alignment, relying mostly on a tabular representation. Their analysis also included a picture of the Browser-Soar problem spaces drawn by hand in MacDraw (their Figure 3).

Table 7-29: Problem space level statistics for the "Write" episode (concl.).

```
 1 . . . O: define-evaluation-criterion (define-evaluation-criterion)
 1 . . .  G: (operator no-change) (g103)
 1 . . .  .P: define-evaluation-criterion (define-evaluation-criterion) (17 chunks)
 1 . . .  . S: s117 (no name)
 1 . . .  . O: generate-evaluation-criterion((value-of-something)) (generate-evaluation-criterion)
 1 . . .  . O: evaluate-evaluation-criterion (evaluate-evaluation-criterion)
 2 . . . O: search-for-help (search-for-help)
 2 . . .  G: (operator no-change) (g1025)
 2 . . .  .P: search-for-help (search-for-help) (17 chunks)
 2 . . .  . S: s1043 (no name)
 2 . . .  . O: find-criterion(keyword) (find-criterion)
 2 . . .  .  G: (operator no-change) (g1049)
 2 . . .  .  .P: find-criterion (find-criterion) (27 chunks)
 2 . . .  .  . S: s1066 (no name)
 2 . . .  .  . O: focus-on-current-window (focus-on-current-window)
 9 . . .  .  . O: evaluate-current-window (evaluate-current-window)
 9 . . .  .  .  G: (operator no-change) (g2415)
 9 . . .  .  .  .P: evaluate-items-in-window (evaluate-items-in-window) (85 chunks)
 9 . . .  .  .  . S: s2432 (no name)
65 . . .  .  .  . O: read-input (read-input)
65 . . .  .  .  . O: attempt-match(i2504) (attempt-match)
 7 . . .  .  . O: change-current-window (change-current-window)
 7 . . .  .  .  G: (operator no-change) (g2339)
11 . . .  .  .  .P: mac-methods-for-change-current-window (mac-methods-for-change-current-window) (34 chunks)
11 . . .  .  .  . S: s3042 (no name)
11 . . .  .  .  . O: scroll(help-text) (scroll)
11 . . .  .  .  .  G: (operator no-change) (g3054)
11 . . .  .  .  .  .P: mac-method-of-scroll (mac-method-of-scroll) (21 chunks)
11 . . .  .  .  .  . S: s3070 (no name)
 4 . . .  .  .  .  . O: move-mouse(help-text down) (move-mouse)
11 . . .  .  .  .  . O: press-button (press-button)
11 . . .  .  .  .  . O: release-button (release-button)
 2 . . .  . O: access-item(keyword) (access-item)
 2 . . .  .  G: (operator no-change) (g2527)
 2 . . .  .  .P: mac-methods-for-access-item (mac-methods-for-access-item) (4 chunks)
 2 . . .  .  . S: s2542 (no name)
 2 . . .  .  . O: click-on-item(i2537) (click-on-item)
 2 . . .  .  .  G: (operator no-change) (g2548)
 2 . . .  .  .  .P: mac-method-of-click-on-item (mac-method-of-click-on-item) (5 chunks)
 2 . . .  .  .  . S: s2562 (no name)
 2 . . .  .  .  . O: move-mouse(keyword unspecified) (move-mouse)
 2 . . .  .  .  . O: click-button (click-button)
 2 . . . O: evaluate-help-text (evaluate-help-text)
 2 . . .  G: (operator no-change) (g2576)
 2 . . .  .P: evaluate-help-text (evaluate-help-text) (26 chunks)
 2 . . .  . S: s2592 (no name)
 2 . . .  . O: focus-on-help-text (focus-on-help-text)
 6 . . .  . O: evaluate-current-window (evaluate-current-window)
 6 . . .  .  G: (operator no-change) (g3104)
 6 . . .  .  .P: evaluate-prose-in-window (evaluate-prose-in-window) (69 chunks)
 6 . . .  .  . S: s3122 (no name)
 6 . . .  .  . O: read-input (read-input)
 6 . . .  .  . O: comprehend (comprehend)
 6 . . .  .  . O: compare-to-criteria (compare-to-criteria)
 4 . . .  . O: change-current-window (change-current-window)
 4 . . .  .  G: (operator no-change) (g3027)
11 . . .  .  .P: mac-methods-for-change-current-window (mac-methods-for-change-current-window) (34 chunks)
11 . . .  .  . S: s3042 (no name)
11 . . .  .  . O: scroll(help-text) (scroll)
11 . . .  .  .  G: (operator no-change) (g3054)
11 . . .  .  .  .P: mac-method-of-scroll (mac-method-of-scroll) (21 chunks)
11 . . .  .  .  . S: s3070 (no name)
 4 . . .  .  .  . O: move-mouse(help-text down) (move-mouse)
11 . . .  .  .  . O: press-button (press-button)
11 . . .  .  .  . O: release-button (release-button)
 1 . . . O: change-search-criterion (change-search-criterion)
```

The model trace and protocol were first printed out and interpreted and aligned by hand, with the correspondences and annotations entered into an Excel spreadsheet. Over the course of testing Browser-Soar with the ten episodes, few changes to the model were required. The first episode was used to create the initial model, and during testing of the next three episodes four additional operators were added and two were modified. During the analyses of the last six, the only changes required of the model were modifying two of the operators.

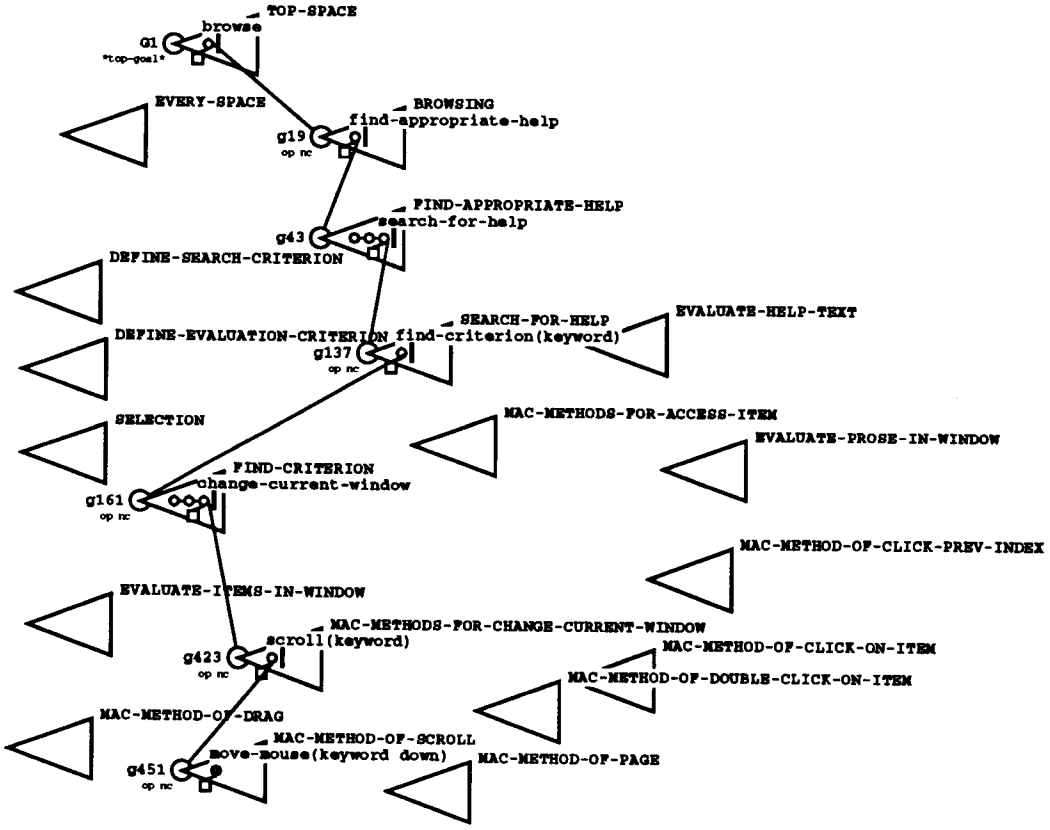


Figure 7-35: Browser-Soar during a run.

All the interpretation was done with respect to operator applications. This included overt, task actions necessary to perform the task, and internal mental actions necessary for deciding what to do and for understanding. The verbal utterances were interpreted with respect to internal operators or their results. Mouse clicks were always interpreted with respect to overt task actions. Mouse movements could correspond to either. When the model predicted that they were required to perform the task and they were used in a task specific way, they were interpreted as overt task actions. When the model did not predict their use, and the mouse pointer could be interpreted as over some part of the display currently being used or read, they were treated like eye-movements and interpreted with respect to an internal operator.

Peck and John's major analyses were to aggregate how many of the subject's behaviors were predicted by the model's actions, aggregating separate measures for directly observable operations, such as mouse clicks, and mental operators that are only observable through verbal protocols or movements of the mouse over words on the screen. Over 90% of the subject's actions and utterances were accounted for by the model's predictions, and the fit between data and predictions was judged to be very tight. These computations were computed by hand for each episode.

The percentage of operator predictions supported by the data were also computed. At 15% this initially appears to be a low rate. One must keep in mind that the trace of the Browser-Soar model provides more predictions than can be tested, even given the rich verbal and non-verbal data streams used to test it. Across all episodes they found indirect evidence for 57% of the operators that could not be directly observed.

An operator support display drawn by hand in MacDraw (their Figure 4, our figure 2-7) illustrated the

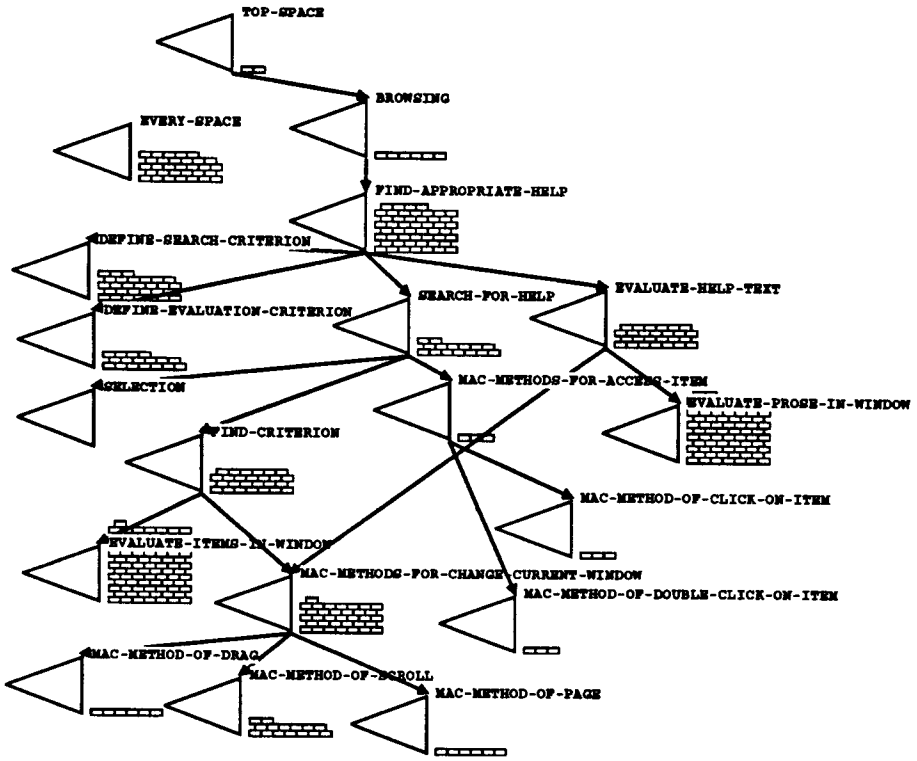


Figure 7-36: Browser-Soar problem space organization with productions shown by their problem space.

tightness of fit, but this was not done for all episodes because it took approximately a day to produce (Peck, 1992).

7.2 Producing richer analyses more quickly

This demonstration of the Soar/MT environment must show that it is possible to duplicate the previous tests of Browser-Soar, and that new, more useful analyses can be performed as fast or faster than have been done in the past. Supporting the analyses is the most important though, speed can come from faster machines or revisions of the software.

This demonstration will not include all the possible analyses that could be or were done by hand for Browser-Soar or for other process models, but it should be clear from this example that the analyses that were not performed are no harder, and are quite likely more easily performed with Soar/MT than by hand.

The emphasis of the analyses will be Grant's (1962) emphasis, finding out where to improve the Browser-Soar, not that all the improvements will be incorporated in this demonstration.

7.2.1 The interpretation of data with respect to the model trace done faster and tighter

Using the Soar/MT environment allowed the interpretation and alignment step to be performed more quickly. The first four episodes were used to debug the Soar/MT system. For later episodes, either analyst (FER or VAP) could go once through the TBPA loop, deriving the predictions, aligning them, and creating the global displays in 20 minutes to 2.5 hours, depending on the pre-existing degree of

alignment, length of the episode, and desired level of detail. The amount of time to analyze another episode is now much less than the initial ten hours needed to understand one.

In each episode the unambiguous data was aligned first. This took on average a minute to set up. During the 30 minutes it would take to rearrange the cells the analyst did not need to be present. The verbal protocols would then be partially interpreted, their locality would be bounded by the matched non-verbal actions near them. A complete listing of the analyses' results are shown in Table 7-30, and the visual, analytic measures created for each display are included as the Appendix to this chapter.

After two episodes of observing me work and working jointly, Virginia Peck (VAP), analyzed three episodes on her own except for creating the displays of model fit. She took approximately 100 minutes to perform these analyses from producing the trace to interpreting the data. Her time was a limited resource, and the software I was most interested in testing was the alignment capabilities, so I created the displays based on her alignments. She also attempted to analyze the last data set, Zcommand, but the unusual size (it is the largest episode by approximately a factor of two) disclosed some bugs in the Spa-mode.

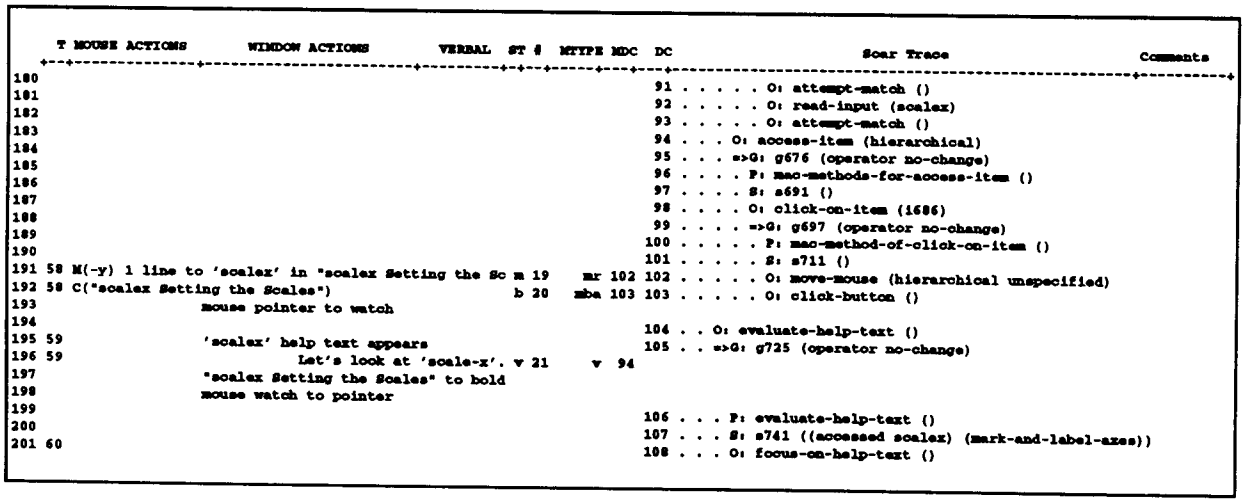
The Card2 algorithm worked admirably. Across the ten episodes it correctly aligned all of the 296 predicted unambiguous mouse actions (mouse clicks and mouse movements). The ability of the algorithm to adjust the edit-list to align a predicted action with the last action in a series of similar subject actions substantially contributed to this performance. Without that modification the results would have been less, around 90%. For each episode the edit list used to align the two meta-columns was generated in under a minute. The alignment of the data with the predictions then took approximately 30 minutes for the Write episode. This alignment process does not require intervention of the analyst. If the two information streams were partially aligned this took less time. A single trial with a single subject on the Write episode, an average sized episode, took approximately 45 minutes to align by hand with Excel. Longer episodes take more than proportionally longer in Excel (Peck, 1992), up to several hours.

After the Card2 auto-alignment algorithm was run, the analyst (FER, VAP, or both) would go through the Spa-mode spreadsheet interpreting the remaining data with respect to the model's predictions. Because both data streams were completely included in Spa-mode, this resulted in a tighter match between the two information streams. Each correspondence included a line of Soar trace (including the decision cycle, the context element selected, and any traced substructures), instead of a coded operator name. These alignments included in the display Soar actions not matched. Figure 7-37 provides a partial example, and the appendix to this chapter includes a complete example for the Write episode.

These alignments generated in Spa-mode provide a more telling comparison of the predictions with the data. Including both data streams in a tabular display shows gaps where the model performed more or less actions (and thus took more or less time) than the subject. When we viewed the first episode aligned this way, we were somewhat surprised by the amount of Soar trace not aligned with subject data. It is also easy to find mismatched actions in this display. False alarms, actions by the model not matched by the subject's actions, which are not representable when the model's predictions are not directly included, can also be represented in Spa-mode. It remains slightly difficult to compare and aggregate the comparisons between episodes with this spreadsheet representation because of the large number of sheets of paper and dispersion of information across them.

7.2.2 Operator support displays created automatically -- as a set they highlight periodicity in behavior

An operator support display for each episode was generated automatically from the alignment data in the spreadsheets. These displays are shown in the appendix to this chapter as Figure 46, along with the displays for a modified model and episode called Better-array, which is explained later in this chapter. These displays originally took approximately a day to produce, so only four were created in the initial analysis (Peck, 1992).



	T	MOUSE ACTIONS	WINDOW ACTIONS	VERBAL	ST	#	MTYPE	MDC	DC	Soar Trace	Comments
180									91	. . . . . O: attempt-match ()	
181									92	. . . . . O: read-input (scalex)	
182									93	. . . . . O: attempt-match ()	
183									94	. . . O: access-item (hierarchical)	
184									95	. . . =>O: g676 (operator no-change)	
185									96	. . . . P: mac-methods-for-access-item ()	
186									97	. . . . S: s691 ()	
187									98	. . . . O: click-on-item (1686)	
188									99	. . . . =>O: g697 (operator no-change)	
189									100	. . . . . P: mac-method-of-click-on-item ()	
190									101	. . . . . S: s711 ()	
191	58	M(-y) 1 line to 'scalex' in "scalex Setting the Sc			m	19	mr	102	102	. . . . . O: move-mouse (hierarchical unspecified)	
192	58	C("scalex Setting the Scales")			b	20	mba	103	103	. . . . . O: click-button ()	
193			mouse pointer to watch								
194											
195	59			'scalex' help text appears					104	. . O: evaluate-help-text ()	
196	59			Let's look at 'scale-x'.	v	21	v	94	105	. . =>O: g725 (operator no-change)	
197			"scalex Setting the Scales" to bold								
198			mouse watch to pointer								
199											
200									106	. . . P: evaluate-help-text ()	
201	60								107	. . . S: s741 ((accessed scalex) (mark-and-label-axes))	
									108	. . . O: focus-on-help-text ()	

Figure 7-37:
Portion of the alignment of the protocol and model trace from the Axis episode. On each row: T is time of subject's actions in seconds. MOUSE ACTIONS is any mouse action. WINDOW ACTIONS are any responses from the actual cT system that the subject saw. ST is the segment type. VERBAL is any verbal utterances by the subject. # is the segment number. MTYPE is type of match, MDC is the decision cycle matched, DC is corresponding Soar decision cycle. Soar Trace holds the model's predictions.

Each display provides a visual depiction of the operator applications for the episode modeled, along with the support each operator received, if any, from corresponding verbal utterances, move actions necessary to perform the task, and mouse movements over screen items that were read. The indentation of the operator names corresponds to their problem space level, and roughly to which problem space they belong to.

For each episode. Individually the operator support displays indicate for each episode the level of support for the model's operators in that episode. Figure 7-38 shows the operator support display for the Write episode. It shows that most of the subject's actions could be interpreted by the model's actions. The verbal utterances mostly match the *Evaluate-current-window* operator, as do the mouse movements that are not required to perform the task.

We also can start to see that the subject's performance shows a definite periodicity. The cycle of evaluating a window, changing it through scrolling by moving the mouse and then clicking on the scroll bar occurs 13 times, with some variations. On the third cycle of examining help topics, the subject sees something that changes her search criteria. On the ninth cycle, she finds a topic that matches the criteria she was looking for, and selects an item for examination. On the remaining cycles she examines the help window. So the main loop is based on menu interaction, and there may be a secondary loop of revision of the search criteria. Just this episode is not enough to tell.

Ohlsson (1980) noted that he could find regularities in protocols that covered a shorter period of time than Newell and Simon (1972) used (200 s versus 1,000 s). This display shows that regularities can occur over shorter time periods. The point is getting enough data, not time. In this domain, in addition to verbal utterances, the mouse movements and mouse button presses help provide the required data density.

A few of the subject's actions could not be interpreted, and they are shown on the bottom as corresponding to the NOT MATCHED operator. Just examining the surface of this display does provide any insight into why they were unmatched, although two of the mouse movements appear to come after a click button operator. When the points and their context are examined by clicking on them (or by finding them in the original spreadsheet, but this is more work), the first is found to be a random mouse movement to a position that is not over something being read or in anticipation of a later click or move, the second the subject laughing, the third another random mouse movement, and

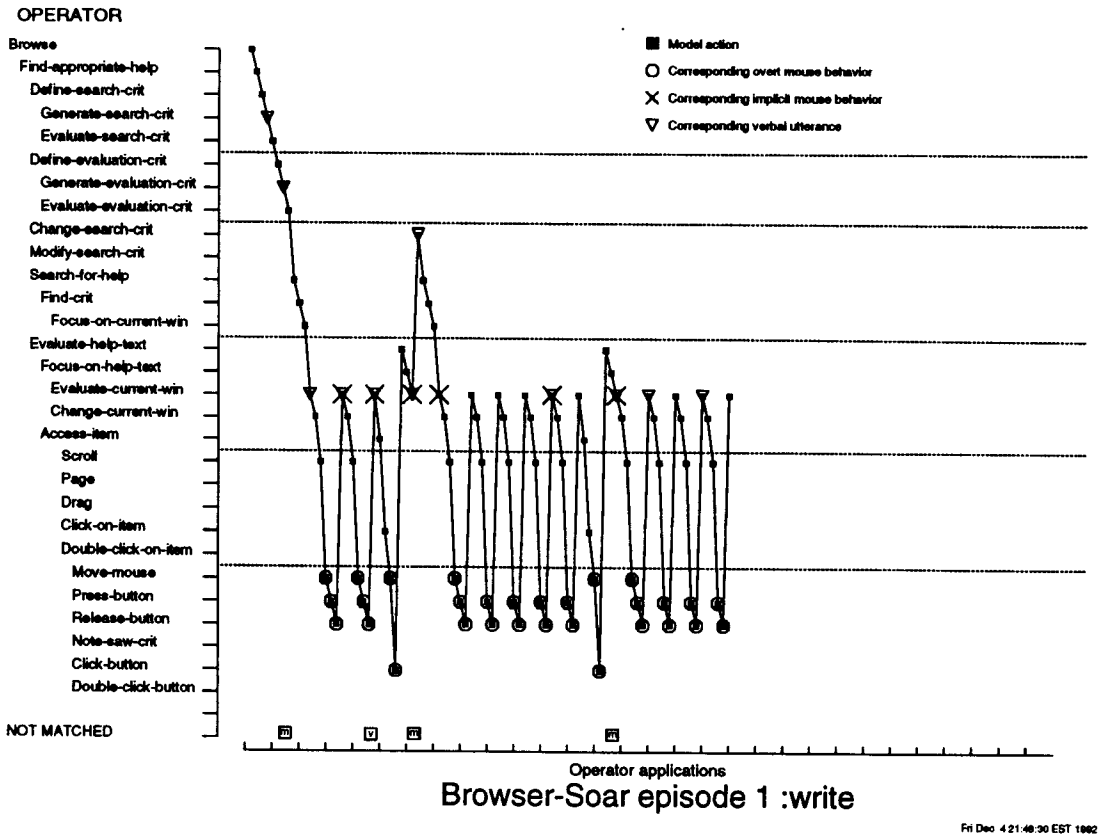


Figure 7-38: Operator support display for the Write episode.

finally, a movement that is interpreted as a mistake. The last mouse uncoded action is a mouse movement that falls short of a scroll bar, and is soon followed by a mouse movement to the position the model predicts.

Across episodes. As a group the ten operator support displays (included as Figure 46 in the appendix to this chapter) tell us even more, and the reader is encouraged to examine them before proceeding. The largest effect visible when viewing these en masse is the periodicity. The longest episode, Zwrite, looks like the display of an oscilloscope indeed.

When viewed together we also can start to characterize what operators are supported and with what types of evidence. We can see that the subject did not talk about every operation. (Many operators have no mark (V) indicating a corresponding verbal utterance.) This is predicted by Ericsson and Simon's (1984) theory of verbal protocol production, so this is as expected, and the rate, in quantitative terms is probably acceptable as well. However, it is slightly surprising to see what this looks like, see just how little is said and supported in each single episode. Based on these displays, Browser-Soar appears too small grained indeed, much more detail is provided than in Newell & Simon's models where nearly every production firing could be matched against a verbal segment.

Across the ten episodes, the subject talked about operators that she should have talked about, and did not describe operators that she should not have. Higher level operators, such as setting up the search criteria and evaluating the window contents were often talked about. These operators manipulate verbal representations, so they should appear in the verbal protocol stream. The motor operators for actually scrolling the windows were never mentioned in the verbal protocol, and this is appropriate

given our measurement theory (Ericsson & Simon, 1984), for they would include non-verbalizable operators or information.

What is not verbalized? The *Change-current-window* and its implementation operators *Scroll*, *Page*, *Drag*, and *Click-on-item*, were never supported by verbal utterances, nor could they be directly supported by mouse movements or mouse button actions for they are themselves implemented with lower level primitives such as *Click-button* and *Move-mouse*. In the future, they must be considered for removal, unless other evidence, perhaps timing evidence, can be provided for them.

The mouse clicks also appear not to be in working memory. In no episode did the subject report that they were using the mouse. Based on the Soar architecture we would believe that they are motor operations, so we would not expect them to be directly represented. The external motor actions need to be set up, however, and the operator that does this remains unsupported.

New questions these displays raise. In each episode at least one of the operators that set up the search in the cT help browser, the first seven operators below the *Browse* operator, is mentioned at the beginning of an episode. Never are they all mentioned, and eight different combinations appear across the ten episodes. It may be possible to combine or rearrange these operators to provide more consistent support for a single operator or set of operators.

During both the Zwrite and Vars episodes, there is a long period of behavior where nothing is said. Similar periods exist in other episodes but there are verbal utterances and mouse movements to support the intermediate operators of reading the topic lists. Characterizing these periods in some way remains an open problem.

Several problems remain with this display. The indentation of the operator names hints at their hierarchical relationship to each other. The implementation of their relationship remains poorly specified and awkward. Operators can come from different problem spaces, and still appear at the same level in the hierarchy.

7.3 Where the model and subject process at different rates shown clearly

Relative processing rate displays were created automatically from the alignment data for each episode. A complete set of these displays is included in the Appendix to this chapter as Figure 47.

7.3.1 Processing rate display based on decision cycles shows that the quality of fit is high

The relative processing rate display can provide hints about how to improve the model within a single episode. Across episodes it can provide additional hints, and measures of the architecture can start to be taken.

For each episode. As an example, consider Figure 7-39, which shows the relative processing rate display (developed in Chapter 5) for the Write episode of Browser-Soar. Each correspondence between the model's predictions and the subject's actions is noted with a connected symbol. Each correspondence shows the relative times when the model and the subject performed the same action. The time that the subject performed the matched actions is represented in seconds on the x axis, and the time that the prediction occurred in the model's behavior is represented in model cycles on the y axis.

The number and relative linearity of the line of correspondences indicates that the predictions generated by Browser-Soar are relatively well matched by the subject's behavior. The number of unmatched subject segments, placed on the bottom of the display at the time they occurred is a relatively low amount, and there are no overt task actions performed by the model that were not observed in the subject. If there were any, these would go near the y-axis.

The squiggles and sections with extremely high or low slopes show where the fit could be better.

When the line becomes more vertical, it means that the subject has started to perform faster than the model, and when the line becomes more horizontal, it means that the model is performing faster than the subject. In Figure 7-39 both occur.

A regression line is provided to help judge the rate of correspondences, and it is used to provide some additional information as well. Its slope is the relative processing rate for that episode in decision cycles per second. The correlation it computes may be a prediction of how well the model can predict the time course of processing in an episode, but it is likely to show a falsely high correlation. Note that the relationship of decision cycles to seconds (the slope) is well within the range (indicated by the dashed lines at 3 DC/s and 30 DC/s) predicted by the Soar theory.

In each episode the correlation between the subject actions and predictions (measured in decision cycles and operator applications) is fairly high. The values of the slope and r^2 value for each episode are shown in Table 7-30. These values for r^2 values are comparable to well developed single response models (e.g., $r^2 = 0.79$: Thibadeau, et al., 1982; $r^2 = 0.94$: Just & Carpenter, 1985). Browser-Soar is near to making engineering level predictions of human behavior, as has been called for by John (1988).

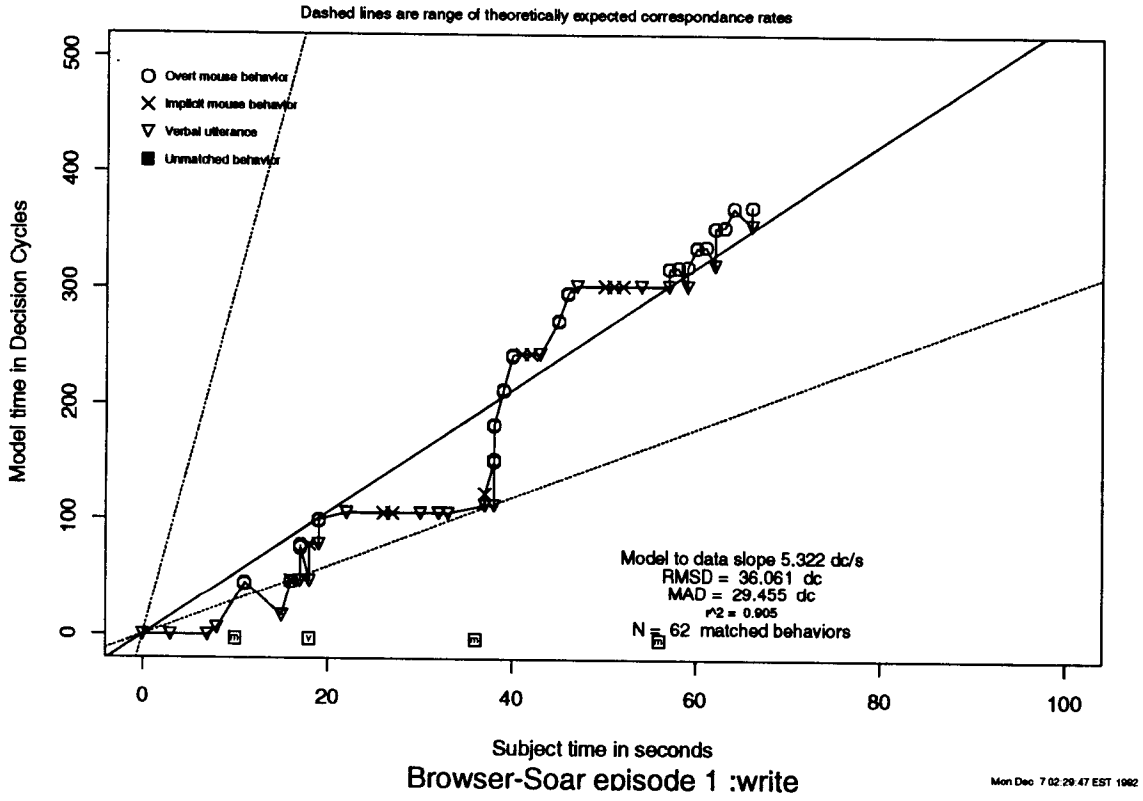


Figure 7-39: Relative processing rates display in decision cycles for the Write episode.

The first parts of display that give specific recommendations on ways to improve the model are the relatively vertical and horizontal sections of the line of correspondences. These sections represent periods where the model and the subject are processing information at relatively disproportionate rates. When the points on the horizontal section between 20 and 40 s are examined by clicking on them, one finds that they all matched to *Evaluate-current-window* operator. This operator is taking much longer for the subject to perform than it does for the model (this operator essentially reads at 100 words/s).

The model could be improved by incorporating a more complete operator to evaluate the current window, that is, read the help text.

The second part of the display to examine is the near vertical line at around 40 seconds. In this section the model is reading every word on a menu while the subject must be skimming the menu's contents, as suggested by the relative rates of processing. Here the model must be smarter about what it is doing, and do less processing than it currently does.

Table 7-30: Summary of raw measures for each episode and regression results.

	Episode	Segments	Time (s)	Words N	Words Rate w/min	Raw DCs	Slope dc/s	Slope DC-r2	Slope op/s	Slope op-r2	DCs/op
1	write	62	66	113	102	399	5.32	0.90	1.18	0.93	4.50
2	unit	40	39	91	140	331	11.29	0.68	2.40	0.80	4.70
3	array	96	68	151	133	517	9.48	0.69	2.26	0.78	4.19
3'	array'	96	68	151	133	346	6.37	0.59	1.49	0.75	4.27
4	precision	21	25	58	139	146	6.82	0.19	1.95	0.34	3.49
5	marker	32	47	162	206	116	2.58	0.43	0.91	0.33	2.83
6	axis	46	83	245	177	173	1.53	0.80	0.45	0.70	3.40
7	labelx	23	34	52	91	77	2.04	0.51	0.61	0.53	3.34
8	circle	52	65	136	125	395	7.32	0.79	1.74	0.78	4.20
9	vars	69	27	44	97	805	35.21	0.90	6.21	0.90	5.66
10	zcommand	140	108	213	118	1529	16.62	0.58	2.76	0.66	6.02
	Sum:	581	562	1265							
	Mean:	58	56	126	146	449	6.92	0.65	2.05	0.67	4.23
	SD:	37	27	68	36	439	4.65	0.22	1.66	0.21	1.32
	Normalized SD:	0.63	0.47	0.54	0.27	0.98	0.67	0.35	0.81	0.31	0.35

From left to right the columns display for each episode the total number of subject data segments, the time of the subject data being modeled, the number of words uttered during the segment and the rate in words per minute, the slope of the least squares regression line on the correspondences in decision cycles per second, the r^2 for that line, the slope of the regression line in operators per second, the r^2 for that line, and the relative rates in the episode of decision cycles per operator. Aggregate measures do not include the Array' episode. Each episode is equally weighted.

Across episodes. Several known problems of Browser-Soar are shown in these displays. Seeing the problems occur in ten episodes is more believable than seeing it in just one episode. Over individual episodes the regression line matched to the correspondences provides a good prediction of the subject's search time. The results of the regression for each episode are shown in Table 7-30.

The rate of the architecture, in decision cycles per second, is slightly slower in Browser-Soar than the Soar theory predicts. Across all the episodes, as shown in Table 7-30, the average rate of decision cycles is six per second. The Soar theory predicts ten per second, plus or minus half an order of magnitude. As this is an average, the actual rate on a single episode can be much lower. This implies that the model is still slightly lean, performing less of the task than the subject is, or that the theoretical analysis of decision cycle rate is slightly high. The first explanation, that the model performing more efficiently or doing less of the task is consistent with but not as far off as other model results (John & Vera, 1992; Newell, 1972; Ritter, 1988; Ritter, 1989; Rosenbloom & Newell, 1982). The large variance in the rate may be cause for some concern, or may just be artifact of the known problems in the *Evaluate-current-window* operator, the *Read-menu* operator, and their ratio in each episode.

In none of the episodes do we find that the line of correspondences is concave upwards, indicating that the subject's relative rate of performance is increasing relative to the model. The displays tend to display the opposite effect, that the line of correspondences is concave downwards. In general, this would suggest that the model was learning and using what it learned (intra-trial transfer) more than the subject was. I believe, however, that in these analyses, this is caused by the order of menu reading and

text reading in this task and the relative performance of the model with respect to the subject. In each episode the basic task units are first to read a menu and then to read some help text, and this sequence may be repeated. The model is slower than the subject at reading menus (causing the line to become more vertical) and faster at reading help texts (causing the line to become more horizontal). This is probably what is causing the curve of correspondences to be concave downwards. Any within episode learning effects will not be visible until these larger problems are ameliorated.

There is often an initial horizontal segment in the first 5 to 10 seconds where the subject is taking much longer than the model. The Write episode, for example, displays this effect. We can find out from the operator support display that this region is exclusively where the selection and evaluation criteria are decided upon. It appears that these operators are too simple, at least in terms of the amount of processing that they perform. This mismatch is smaller than the text and menu reading rates, but probably does reflect a basic problem.

We also can note some problems interpreting this display. The verbal utterances have durations, and currently only their starting point is taken into account. All operators are treated as taking the same amount of time. If substructure will be added at a later point to an operator, the analyst can not currently represent that it should take longer than a simple operator.

7.3.2 The processing rate display can be based on other measures of the model's effort

The relative processing rate display can represent the model's rate of processing in measures other than the decision cycle rate. In this subsection a version using operator applications is used to test Browser-Soar. This display is the same display as the display based on decision cycles, except the model's performance is viewed with a different metric.

Figure 7-40 provides an example display of the relative processing rates of the model (in operator applications) and the subject (in seconds). A complete set, one per episode, is included in the appendix to this chapter. A regression line is still fit to the line of correspondences to indicate outliers, but an expected range is not provided because the Soar theory does not provide one — it will depend on how often problem spaces are entered and exited, which is based on the task at hand and the knowledge that can be brought to bear.

The results of computing the relative performance of the model in terms of operator applications are reported in Table 7-30. The operator application rate (in seconds) has a wider relative range and varies more than the decision cycle rate does; the normalized standard deviation of the operator rate is higher. The number of operator applications the model took to perform the task correlated as highly with the subject's performance as did decision cycles. The correlation is slightly higher, but it is not significant ($t(9) = 1.05$), nor does it appear to a large enough difference to be important. This is not too surprising, operator applications are caused by and correlate highly with decision cycles.

The known problems with the two *Read-text* and *Read-menu* operators can again be seen in these displays. Relative learning rates within an episode can also be examined, but again, any relationships found probably are due to the big bad *Read* operator.

This display does not appear to tell us anything new about Browser-Soar, but other models may see an effect here if operators are less directly used, or more behavior occurs in each problem space. Similarly, it does not imply that other measures of the model's effort, such as rule applications, elaboration cycles, or problem space selections, will not prove useful in some way. It is, however, the most likely measure after decision cycles to prove useful.

We can note a constant relationship within Browser-Soar with this display that appears constant across episodes: the number of decision cycles per operator as computed from the two regression slopes. It is not clear yet what this really means, it may mean nothing, but if a relationship appears constant, there may be reason for it.

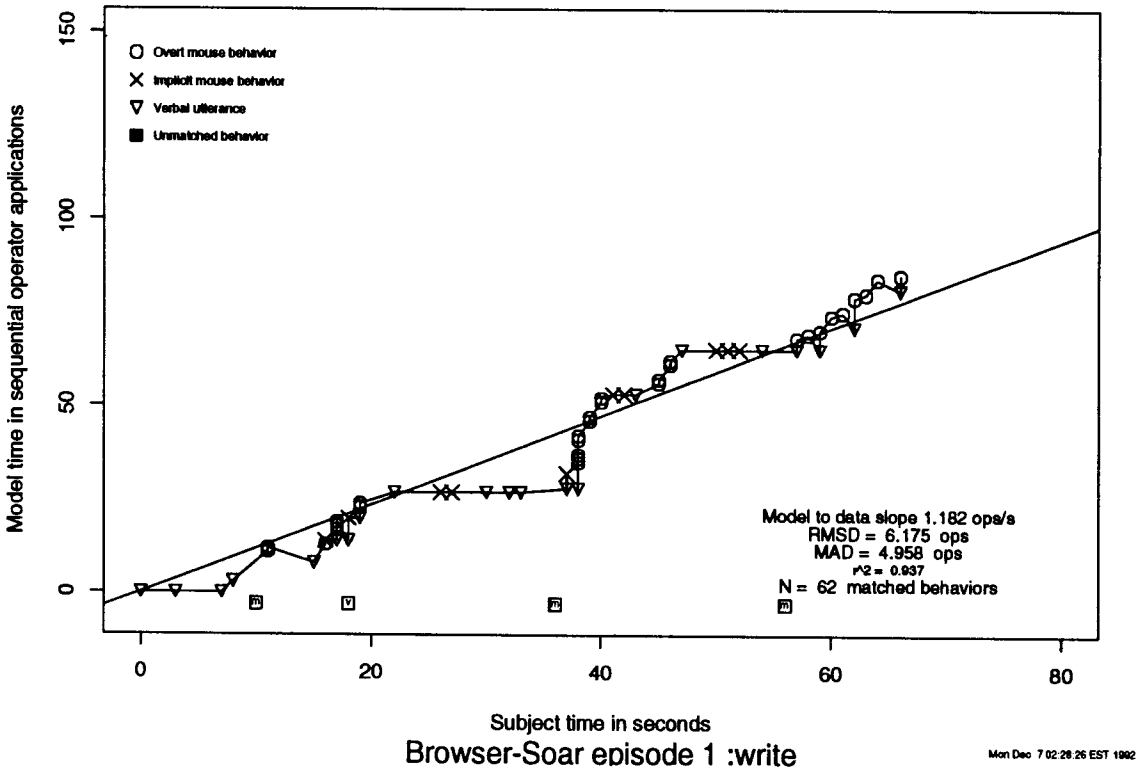


Figure 7-40: Operator applications vs. subject time display for the Write episode.

7.4 High level features of the Browser-Soar model made apparent

Examining Browser-Soar in the SX graphic display suggests further modifications based on how it models routine behavior. Performing a pseudo-model revision to incorporate the effects of learning suggests that Browser-Soar might be improved by using less problem spaces.

7.4.1 Browser-Soar as routine behavior is made directly visible

Search in a problem space means lacking knowledge about how to proceed, and search between alternatives where the solution path is unknown. The solution path in Browser-Soar is not unknown, or at least not substantially unknown. Most operators are the only one proposed, and most problem space impasses are resolved directly. We can see this in the graphic display while Browser-Soar runs. Figure 7-35, which shows Browser-Soar during a run, shows that there are not many operators applied in any one problem space. This is also visible in the problem space level statistics, few states are visited, and not many operators are applied.

Search, in Browser-Soar, when it occurs, also occurs as much as search through problem spaces for knowledge external to the initiating space. The name of "solution space" (Ohlsson, 1990) particularly here, makes more sense, with Browser-Soar more like a task (Ohlsson, 1990) than a problem. This result is noted by Peck and John (1992), but appears more clearly in these pictures and aggregate statistics than in the textual trace alone.

7.4.2 Noting Browser-Soar's large goal depth

The goal stack depth is relatively deep in Browser-Soar. As noted in Figures 7-33, 7-35, and 7-36, the goal stack often grows to be between four and six levels down from the top problem space. This appears to be a large number for what is described as routine behavior (but we have no real metric). In addition to the question of the depth of the goal stack, all the lower problem spaces for manipulating the mouse and screen represent expert level behavior in the subject, that is, behavior that does not significantly improve with practice. In Browser-Soar impasses still occur, and if learning was turned on, knowledge would migrate between them. In expert behavior, the lowest level of operators and problem spaces in Browser-Soar should not be visible because they have been learned by the problem spaces that use them.

7.4.3 Modifying Browser-Soar

With the learning constraint in mind, a modified version of Browser-Soar was created and tested using the pseudo-model revision method mentioned in Chapter 3. The modified version does not contain the lower level problem spaces that would have been learned. The actual output operators were migrated to higher level problem spaces, and intermediate operators and problem spaces that did not receive support from the data, such as the operators in the *Access-item* problem space. A complete listing of the modifications is provided in Table 7-31.

Table 7-31: Problem spaces and operators removed from the Browser-Soar model simulating the effects of learning.

- Browsing PS and OP,
- Find-criterion OP and PS,
- Mac-methods-for-change-current-window PS,
- Change-current-window OP,
- Drag OP,
- Scroll OP,
- Mac-method-of-scroll PS,
- Mac-method-of-drag PS,
- Mac-method-of-page PS,
- Access-item OP,
- Mac-methods-for-access-item PS,
- Click-on-item OP,
- Mac-method-of-click-on-item PS,
- Evaluate-help-text OP,
- Evaluate-help-text PS,
- Double-click-on-item OP,
- Mac-method-of-double-click-on-item PS, and
- All associated goals and states.

Figure 7-31 shows the problem space organization of this modified version of Browser-Soar. The organization can be compared with the original version shown in Figure 7-33. The new version has fewer problem spaces, and is flatter. The maximum goal stack depth of this version is four, with final

depths of two and three. It has 8 problem spaces compared with 17 problem spaces in the original Browser-Soar, 22 operators compared with 31 in the original, and a corresponding decrease in the number of intermediate states and impasses.

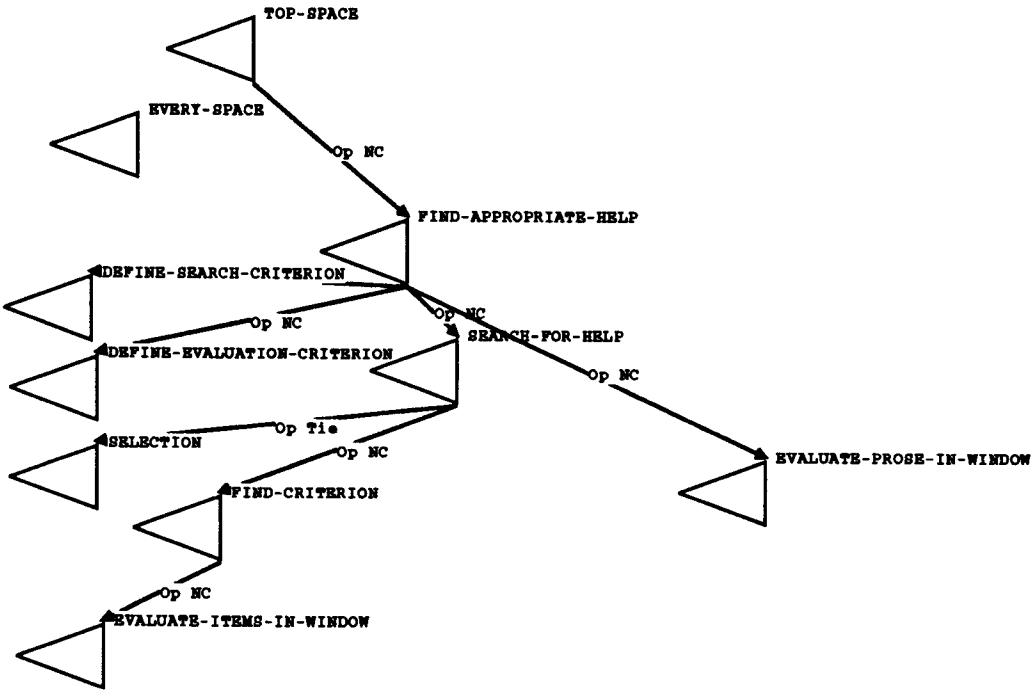


Figure 7-41: The nine problem spaces in the modified Browser-Soar (see Figure 7-33 for the original structure).

The revised model was not implemented on the production level, but was created using a more lightweight technique of trace revision. All the operator and problem spaces that were removed, were simply deleted from the trace for the Array episode, the second largest episode, and the trace was renumbered. This took approximately 20 minutes. These changes also could have been implemented by modifying the model, and rerunning it. Theoretically there would be no differences, in reality, actually editing the code instead of the trace probably represents an order of magnitude more work.

As the actual model was not modified, this represents an instance of pseudo-model based revision, where an aspect of the analysis changed in terms of the model, without the expense of completely implementing the changes on the production level.

7.4.4 Testing the modified Browser-Soar

After the revised trace was made, the two information streams were realigned from scratch. Because no model actions with support were removed, the realignment was essentially the same. It would have been faster to use the old alignment and modify it slightly, removing the empty cells, for no correspondences were cut, but I wanted to see what the total process could look like, and see how long a more modified model would take to test. The total time to perform the model manipulation, realignment, and generate the analyses was 2.5 hours.

Figure 7-42 shows the operator support displays for the two versions of Browser-Soar. The displays are essentially the same, the shape is the same, and the subject actions and the operators that they correspond to are all represented. The only difference is that the modified version is more compact; it

has less operator applications.

Figures 7-43 and 7-44 show the model fit displays for the modified version of Browser-Soar next to the original versions. These two displays show that the revised model has a denser level of support, the lines connecting the corresponding model and subject actions are closer together, and the RMSD and mean average deviation are lower. The rate of decision cycles to seconds ratio is also closer to the predicted mean, and visually the fit appears to be better. The modified version has slightly worse r^2, more so when the model time unit is decision cycles (.69 versus .59) than for operator applications (.78 versus .75). The correspondence rates in decision cycles and operator applications per second for the modified model also go down, as less is done.

It is hard to tell if these differences are important. It would perhaps become easier to tell after further revisions of the *Evaluate-current-window* operator, and with a more proper regression line (Kadane et al., 1981; Larkin et al., 1986). These results do point out that it is hard to distinguish learning on the single problem space level at this time grain. In order to clearly distinguish these two problem space representations we would have to look at more episodes, more subjects, or further constraints from data. Given the lack of real difference, parsimony would argue for using the simpler, modified version of Browser-Soar.

This analysis also calls into question the strict interpretation used. The subject must decide to move the mouse. The operators that were removed originally represented this choice. With a different interpretation function, these operators would have been supported and would not have been removable. As noted in the list of corrections available when the model's predictions mismatch the data (Table 2-6), the interpretation function can also change. This case raises the question of how to interpret data given Soar's hierarchical operators and state representation. This may remain a problem for some time.

7.5 Testing and extending the sequentiality assumptions of protocol generation theory

As noted in their initial description, the relative processing rate displays allow the sequentiality assumption of Ericsson and Simon's (1984) theory of verbal protocol production to be tested. That is, if verbalizations are produced in the order that the corresponding data structures appear in working memory. There is another aspect to this assumption, that inputs to operators will be reported before their outputs, but is a more specific form that will not be directly tested unless we run into problems. A model of what appears in working memory is currently necessary to test this assumption. There are no other ways to tell when information enters working memory, and thus that it is reported in order. Having a model of the contents of working memory also allows use to judge if the verbalizations are retrospective or prospective.

Browser-Soar provides predictions of the contents of working memory while using a specific on-line help system. By examining the relationship of these predictions with the subject's verbal utterances in the ten Browser-Soar episodes, the sequentiality assumption can be tested.

The predictions of the external task actions (mouse movements and button presses) can also be compared with the contents of working memory, but because getting the order of the external actions the same for both model and subject is essential for performing the task, in a well developed model like Browser-Soar there is not likely to be many mismatches. What will be interesting though, is using the external actions to compute how later (or early) the verbal utterances are.

Finding that this holds will not be an iron-clad proof that this assumption holds. If it is an assumption, then it cannot be proven, only shown that we meet it. If it is treated more as part of the theory of verbal protocol production, then there may be similar models of browsing behavior where the information is reported in a different order, and that the current set of verbal protocols would not match sequentially.

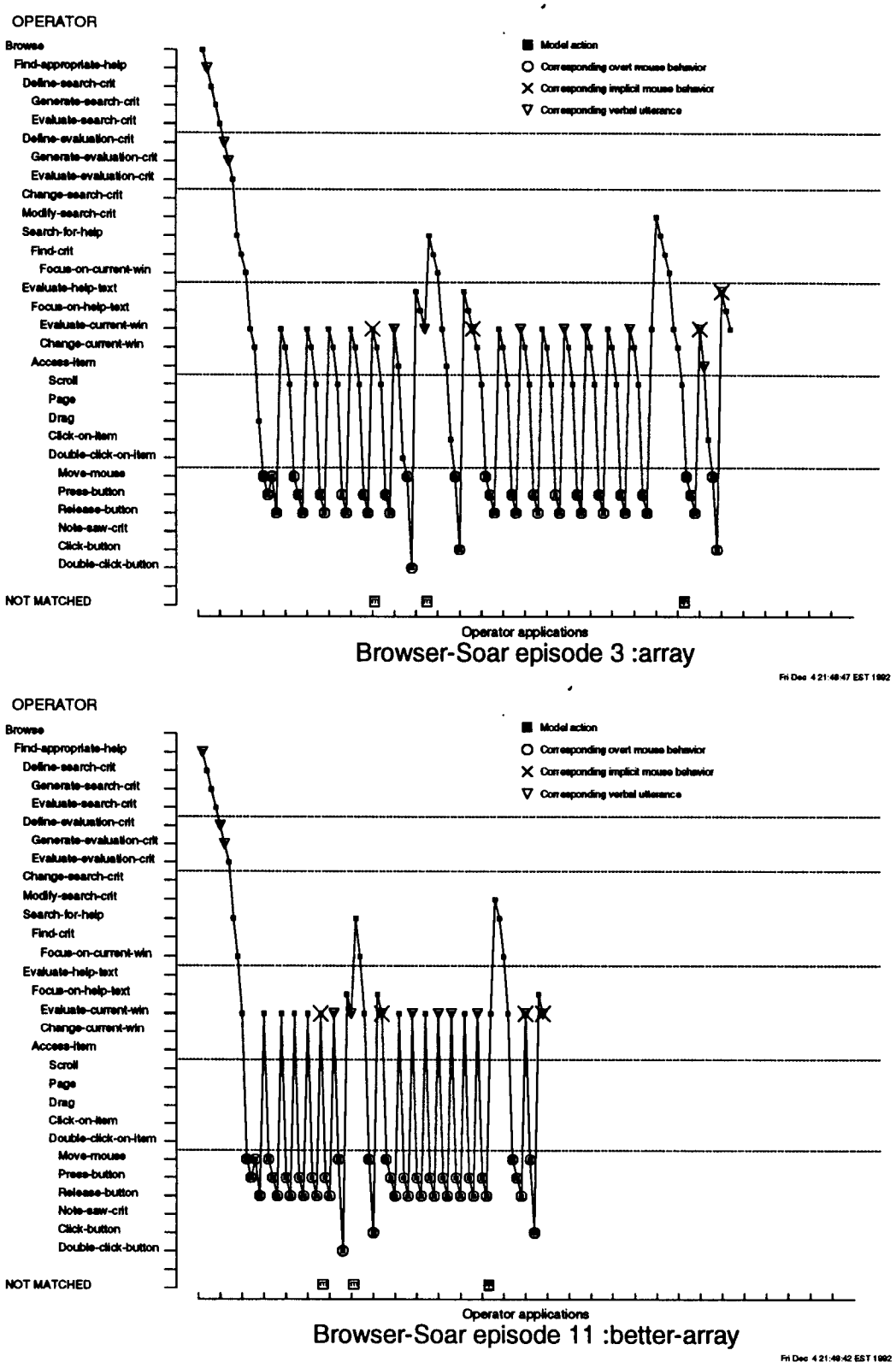


Figure 7-42: Operator support displays for the Array episode. The original Browser-Soar predictions are on the top, and the modified version on the bottom.

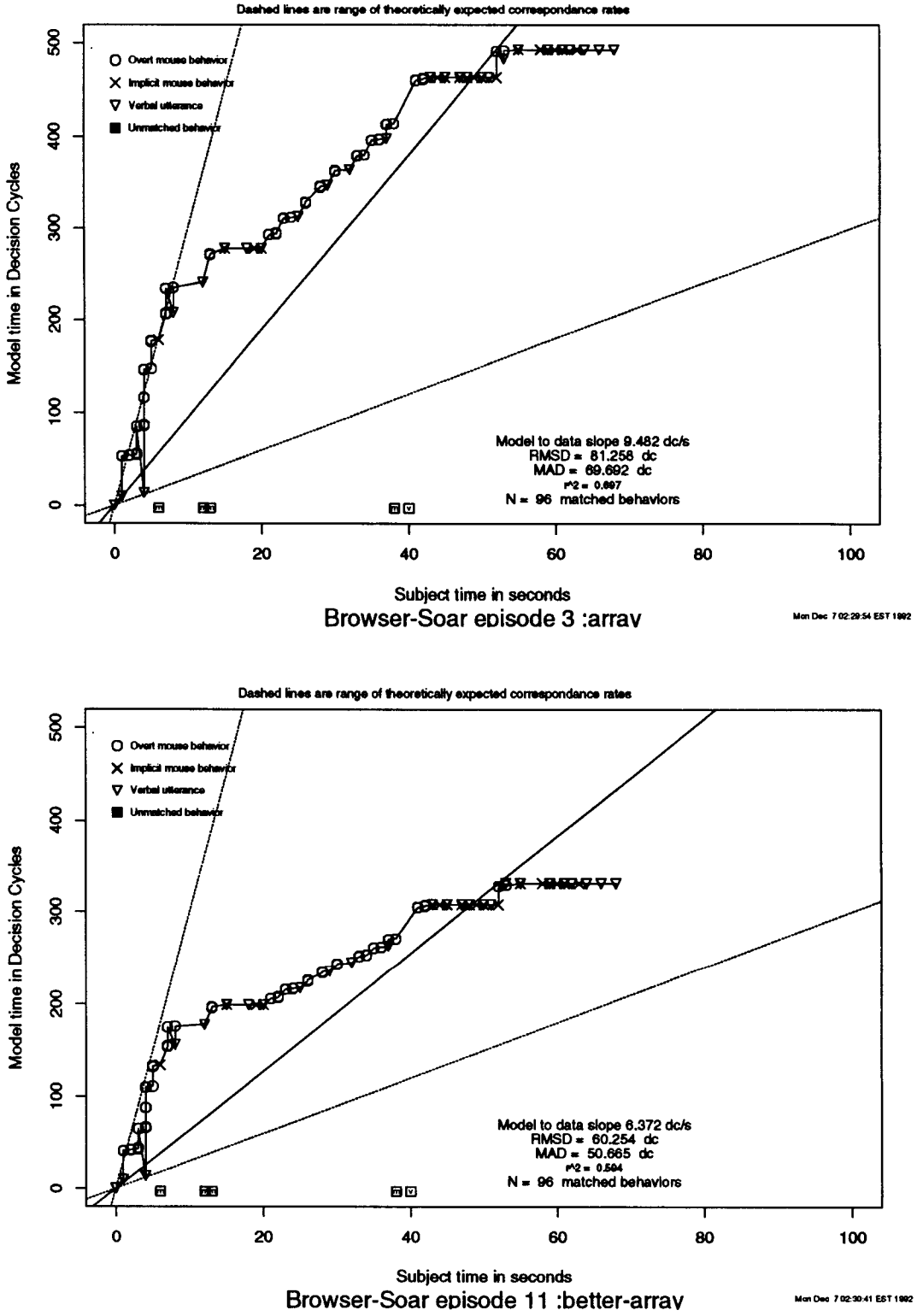


Figure 7-43: DC time based plots for the Array episode. The original Browser-Soar predictions are on the top, and the modified version on the bottom.

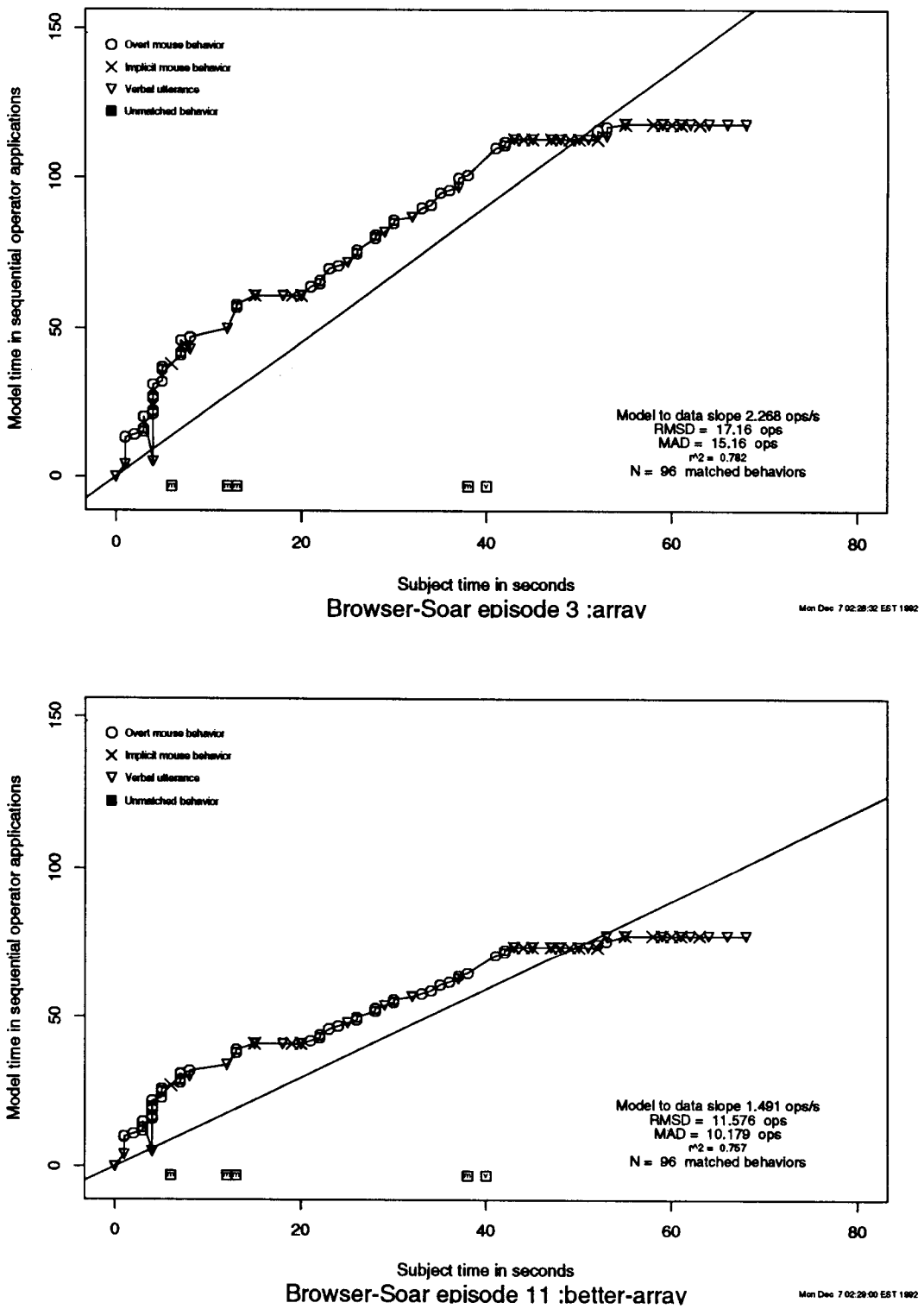


Figure 7-44: Relative processing rates displays based on operator applications for the Array episode. The original Browser-Soar predictions are on the top, and the modified version on the bottom.

7.5.1 Are verbalizations generated sequentially?

Of the 220 verbal utterances in the ten episodes, 195 can be aligned with the model's predictions. The remaining 25 are mostly too short to compare. The remaining segments make up 210 pairs of immediately sequential utterances that can be tested against the sequentiality assumption. This test can be performed by eye with the displays, and the initial analyses did this because it was so easy and direct. The final counts were taken from the data structure used to create the displays.

All 210 pairs follow the sequentiality assumption; for all the pairs, the later segment in each pair either matches the same model trace action as the first segment matches or a later model trace action. So this appears to be another constraint that Browser-Soar meets.

7.5.2 Are mouse actions generated sequentially?

In a similar way the mouse movements and mouse button actions can be tested for sequentially. Because these actions were used as fixed points to automatically align the subject's protocol and the model's trace, in order to match out of sequence they would had to have been moved by hand out of sequence, or items that could not be automatically aligned would have had to be aligned by hand.

Of the 404 mouse actions in the ten episodes, 373 can be aligned with the model's predictions.[9] These 373 actions make up 363 pairs of sequentially contiguous actions. Again, a preliminary examination of the displays showed that none matched the model out of order, and an analysis of the data base confirmed that.

7.5.3 Does the sequentiality assumption hold across verbalizations and mouse actions?

All the subject's actions can be tested for sequentiality. As explained in Chapter 5, this can be done by examining the connected correspondences in the relative processing rate displays. Starting from the first correspondence and moving along the line of correspondences, a connecting segment with a negative slope indicates that the second correspondence matched earlier in the model than the first correspondence, violating the sequentiality assumption. Simply examining the displays shows that several verbal utterances lag the mouse movements noticeably. Of the 624 total segments, 568 are aligned with the model's actions in the ten episodes.[10] These 568 actions make up 558 pairs of sequentially contiguous actions, and 21 pairs do not meet the sequentially assumption, that is, in these pairs, the second subject action is a verbal utterance that matches an earlier prediction than the first action that is a mouse action.

The lag of verbal utterances was computed by comparing the decision cycle number of the model prediction corresponding to the verbal utterance with the decision cycle of the previously matched mouse action. Figure 7-45 shows the distribution of these times. Across all verbal utterances in all episodes the average lag was 9 decision cycles, or roughly 1 second. This is an acceptable number, indicating that while some verbal utterances appear to have been produced quite late compared to the mouse movements, overall the subject was not providing retrospective reports.

Most of the verbal statements (144 out of 195) match the model's predictions sequentially, not matching earlier portions of the model than their proceeding segment. Based on their starting points these utterances can be considered as truly concurrent protocol — it is generated as the subject doing the task and it matched the predictions of the contents of working memory. The ends of the utterances have not been included in these analyses, although Peck and John included this length in their data set.

---

[9]An astute reader may note that there are five more mouse movements matched by subject actions in this analysis than in the original analysis reported by Peck and John. One of these discrepancies has been found so far, and it was a typo.

[10]An astute reader may again note that there are five more predictions matched by subject actions in this analysis than in the original by Peck and John. Even with a semi-automatic tool, analysts will make mistakes.

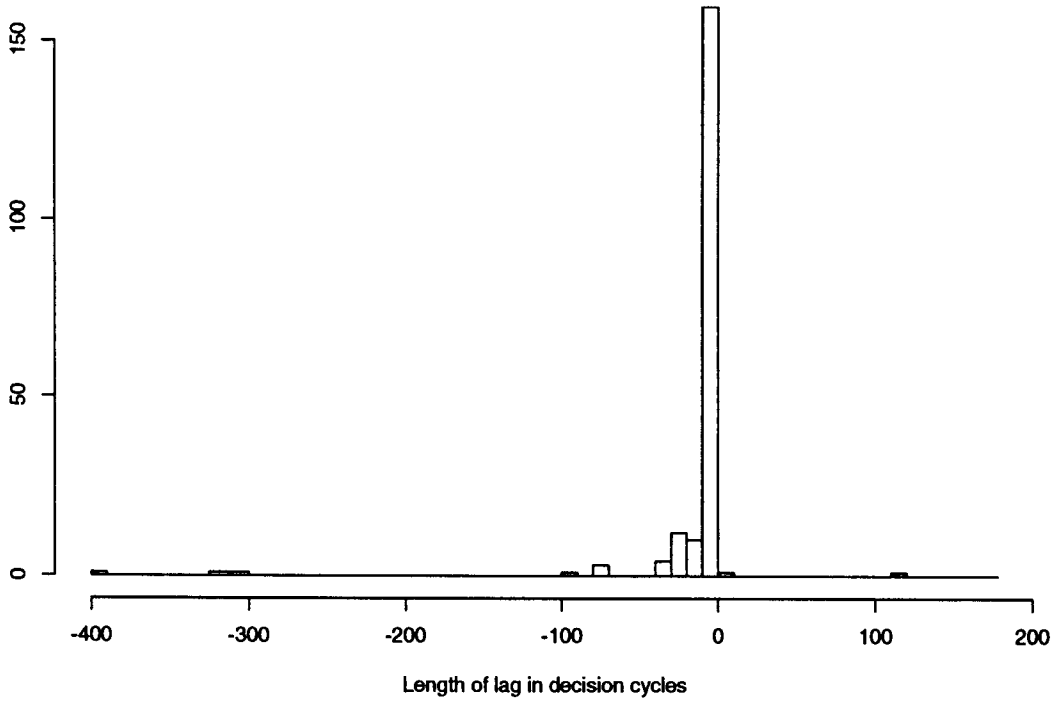


Figure 7-45: Histogram of the lags (in decision cycles) of the verbal utterances.

While these segments are not long generally, it is possible that their tail end ceases to be concurrent.

There are two prospective utterances, one in the axis episode, which upon inspection was an typo in alignment. The segment was properly concurrent, but misaligned by four decision cycles in the spreadsheet. The other utterance occurred in the Vars episode and is more interesting. It has a positive offset of 111 decision cycles (nominally 11 seconds). It is hard to see on the relative processing rate display because it is surrounded by several mouse movements, which is the cause of it being interpreted as early. When the segment is examined, it turns out that the verbal utterance is not so much prospective, but that the model's menu reading ability falls behind the subjects at that point, and the model has to perform an extra 100 cycles of work before it can match the verbal utterance.

The remaining 49 utterances all lag their previous segment, matching an earlier prediction. When an utterance lags, it lags on average 38 decision cycles, or roughly 4 seconds. Again this remains a modest amount. This amount of time is consistent with the amount of time items can exist in working memory. A very small number, three, lag over 300 decision cycles.

Characterizing the long lags Many short lags of the verbal utterances appear to be partly (but not completely) an artifact of the Browser-Soar model. The model does not read individual words but whole screens at a time, which leads to many of the short lags that occur late in an episode when the subject is reading a help text. Including predictions of reading individual words would remove this cause.

The three longest lags, however, are worrisome. They lag over three hundred decision cycles, and represent a mismatch on the order of 20 to 40 seconds. The problem space of the operator they match

has long been removed from the goal stack, and several other problem spaces on that level have been used as well. When these segments are examined they are found to be statements of the search or evaluation criteria that occur after the search has started and numerous items have been examined. While an operator put them on the state, at the point they are uttered, they clearly represent state information that has been guiding the search for some time. Other operators could be refreshing them, but if that is what lead to these utterances, then the operator used to interpret them is still the wrong one.

Finding this lag in the literature. The actual lag of verbal protocols has not been computed in this way to my knowledge. It requires an architecture that makes predictions about the time to perform a task, external actions to provide fixed points of reference, and the predictions must be aligned to this detail. We can see a lag in other data sets, however. The verbal data used to develop HI-Soar (John & Vera, 1992; John, et al., 1990) can be fixed relative to the performance of external actions. The verbal protocols lagged behind the external actions so much that they were ignored when testing the model.

7.6 Conclusions about Browser-Soar and the TBPA methodology

Having performed these analyses, we can summarize the results into several suggested changes to Browser-Soar, which is the point of testing a process model. In general, Browser-Soar performed very well. The operators in the model that performed best were the ones that are essential to browsing on-line help systems: manipulating the mouse, choosing windows, and evaluating text items. On a higher level, testing Browser-Soar also generated some lessons for the methodology and for the environment that should be incorporated into the environment.

This methodology was stretched in a particular direction through testing Browser-Soar. Browser-Soar and the data used to test it have some very particular characteristics: (a) very close matches, (b) very routine behavior and typical problem solving by the subject, (c) a highly interactive task, (d) mostly a mental task (the perception and motor actions were routine). This example application did not deal with every type of data. It is easy to name several data features that have not been touched: (a) very bad matches between data and model, (b) perceptually based reasoning, (c) how to create a model in the first place, or drastically revise it, (d) tasks that cannot be modeled as search through or in problem spaces, and (e) extremely long or short protocols. Adding any of these features to the data and task is likely to add further lessons and stretch the methodology in a new way.

7.6.1 Some conclusions about Browser-Soar

The analyses performed suggest several ways to improve Browser-Soar. Most, if not all, are known to the authors of Browser-Soar, but the importance and location of the changes should be clearer after these analyses. These changes are presented in Table 7-32.

Browser-Soar's ability to predict large amounts of the data should also be clearer as well. Chapter 2 put forward the idea that analytic testing would not only point out where to improve a model, but it also would make it more believable by presenting it more clearly. Several diagrams and tables were created in performing these analyses that should make the model more believable. There are more visual descriptions of the model (Figure 7-33), its performance (Figure 7-35), a rough measure of the amount of knowledge in each problem space (Figure 7-36), and a picture of the calling order of its operators (Figure 7-38). Aggregated measures of which operators and problem spaces are used and how often have been presented (Table 7-29). The analytic displays show when operators are supported, and by which type of data (Figure 7-38 and the Appendix to this chapter), and the relative processing rates of the model and subject over time (Figures 7-39 and 7-40, and the appendix to this chapter).

Table 7-32: Suggested changes to Browser-Soar based on analyses performed.

- Operators without evidence, *Scroll*, *Page*, *Drag*, and *Click-on-item*, must be considered for removal from the model, or be supported with non-protocol data such as aggregate timing results.
- Fitt's law should be included in the model of moving the mouse.
- A more complete *Read* operator for reading text that takes longer.
- A less complete *Read* operator for reading menus faster, more like scanning.
- Overall, the model's performance is slightly lean, but this must be reevaluated after some other problems, most importantly the reading operator, have been improved.
- Include learning, and decrease the goal stack depth.
- Include state information in the trace and match to it.

7.6.2 Some conclusions about the methodology

Performing these analyses pointed out that it is nearly always good to have context, and sometimes it is required. Just providing information on a single item is often not enough to understand the item. The item's context is also needed. In several places, particularly in examining the model fit displays, users can now click on a data point and get a segment and a selectable amount of its context displayed.

Different grain sized operators and different commitments to operators lead to problems in the analysis, and should be avoided if possible. Soar in particular, as a general architecture for intelligence, provides the ability to model every action. As a unified theory of cognition it highlights the desire to provide a complete model, covering all the data. By definition, some portions of each model will be weaker than others.

Soar models are much finer in their grain size than Newell and Simon's (1972) systems; more actions occur that cannot be tested, such as goals and many problem spaces. Other items might be found, but are not found in every episode. It may be desirable to omit these items automatically and appropriately when performing an analysis.

While Newell and Simon (1972, p. 179) propose that states and operators are equivalent, the reanalysis of Browser-Soar shows that they are only equivalent for information purposes. When the timing of the correspondences is included, they are not equivalent. States, and the information they contain, last much longer. It may be possible to continue to match verbal utterances primarily to operators, but when this breaks down, one must match to the state. Using the state properly is not a trivial task, and will require designing and extending the trace. It will require further mechanations in the interpretation algorithms to find the appropriate items to support in the model when this does occur.

No problem spaces or goals are used to interpret the subject's behavior. Together their creation and selection make up a substantial portion of the model's behavior. What it would mean to match their prediction is not clear, problem spaces may be supported by their operators and states, goals by the indication of a lack of knowledge in some way. If they will not be directly supported, the cognitive modeler may desire to removal them from the trace if not the model.

Finally, we see that testing the model points out that the model is not complete without rules describing how to interpret the data with respect to the predictions. For example, the *Page*, *Scroll*, *Drag*, and *Click-on-item*, were considered for removal because they were not supported. A more generous interpretation of the mouse actions might have included the decision to click (e.g., to *Page*) as being supported as well.

Appendixes to Chapter 7

1 Alignment of the Write episode of Browser-Soar

```
Wed Nov 25 14:35:42 1992 - Dismal (0.83) report for user ritter
For file /afs/cs.cmu.edu/user/ritter/spa/browser/write/write5.spa

To print use  "enscript -r -c -G -fCourier7 -L66  /afs/cs.cmu.edu/user/ritter/spa/browser/write/write5.dp"
-----------------------------------------------------------

    A       B               C                D    E F  G   H  I                        J                                                K
 +--+---------------+-------------------+----------+-+-+-----+---+---+----------------------------------------------+------------------------------------------+
 vapeck   25-Sep-91 revised 13-Jan-92 -FER            3-Jun-92                                                              To do:
 From original transcription by dcr, 16-Mar-90, and verbal transcription by s.esch,  Jun-91
 Transcription of the 15-Jun-89 cT browser tape 2

 Browsing for information about writing the values of variables to the screen.
 Previous Goal:     define the 'loop' construct that will label the x-axis
 Current Goal:      to figure out how to write the value of 'ExpCondNames' in
                    order to label the x-axis with experimental condition names
 Windows:           program window out front, right side and bottom of help win.,
                    only the left edge of the commands win., right side of execution window
 Program Window     line 1  "***********"    line 2  "unit   DrawGraph"    line 3   "gorigin   110?, 340?"
 Execution Window:  error box with message at the top
 Help Text Window:  line 1  "coarse 'Typing-Paper' Coordinates"
 Keyword Menu       line 1  "coarse" (selected)"
 Hierarchicl Menu:  line 1   "at   Positioning a Display"   (no lines selected)
 Commands Window:   (not used)
 Cursor:            positioned at the end of the 'gat' command line within the 'loop' contruct
 Mouse:             located (+x) -3/4 in. from the end of the 'gat' command line
    (mouse is currently a line)
                                 See also /afs/cs/project/soar/member/vapeck/browser-soar/current/episodes/write/01-22.write.complete.log
 69 total behaviors                          69
 11 distinct operators evidenced in behavior                       Loaded from:
                                                                   /afs/cs/project/soar/member/vapeck/browser-soar/current/LOAD-write.lisp
 25 total verbals                                                  Loading /afs/cs.cmu.edu/user/ritter/spa/browser/ep-trace-additions.lisp.


  last time verbal/mouse matches information used by an operator.
 21 total mouse movements               MTYPE is type of match:
 9  unnecessary movements that give evidence            87   Total
 6  necessary movements

 24 Total mouse button actions                           4    B4 is actions before model coverage.
                                                         2    SHORT is too short to code.
 TIME is timestamp of action in s.                      14    CONT is segment continued from previous line.
 DURATION is length in ms of behavior.
 VERBAL is verbal protocol.                             22    V is verbal coded.
 Mouse Action is the user's mouse movements.             6    MR is mouse required (these are just movements).
 ST is Segment Type                                     11    MI is mouse inferred.
 # is segment number                                    24    MBA is mouse button actions (necessary by default).
 MTYPE is type of match
 MDC is matched DC.                                      3    MUC is mouse uncoded.
 DC is decision cycle in Soar model.                     1    VUC is verbal Un Coded.
 SOAR TRACE is the literal Soar Trace

    0.94           percent subject data matched
    0.89           percent model matched
    0.18           seconds/decision cycle

                   total words               113

 Time starts at 12400
  T  Mouse actions   Window actions   Verbal     ST #  Mtype MDC  DC                     Soar trace                       Comments
 0                                    I believe   v  1 short
                                                                 0 G: g1                                               From:  raw-trace3.txt, 17-Mar-92
                                                                 1 P: p4 (top-space                                    Operator names cleaned up by hand,
                                                                 2 S: s5                                               6-july-92 FER
                                                                 3 O: browse ()
                                                                 4 =>G: g19 (operator no-change
                                                                 5   P: p26 (browsing
                                                                 6   S: s39 ((unknown) (unknown)
                                                                 7   O: find-appropriate-help
                                                                 8   =>G: g43 (operator no-change
                                                                 9     P: p50 (find-appropriate-help
                                                                10     S: s59 ((unknown) (unknown)
                                                                11     O: define-search-criterion
                                                                12     =>G: g65 (operator no-change
                                                                13       P: p72 (define-search-criterion
                                                                14       S: s79 ((unknown)
 6                                    write       v  2     v  15  15       O: generate-search-criterion ((write))
 9                                    write       v  3     v  15
 13                                   write       v  4     v  15
    M(+x) (R of prog win)                                  B4                                                          B-Soar doesn't model bringingup the
                 mouse line to pointer                                                                                      help win.
                                                                16       O: evaluate-search-criterion                  *** generated search criterion 'write' **
                                                                17     O: define-evaluation-criterion
                                                                18     =>G: g103 (operator no-change
                                                                19       P: p110 (define-evaluation-criterion
                                                                20       S: s117 ((unknown)
 14                                   Can I write v  5     v  21  21       O: generate-evaluation-criterion ((value-of-something))
 15 M(+y) (top of screen)                                  B4                                                          unmodeled (before the task) mouse movement
 15 M(-x-y) (portion of help win below prog win)           B4
 16 C            help win comes forward                    B4
```

51	T	Mouse actions	Window actions	Verbal	ST	#	MType	MDC	DC	Soar trace	Comments
82	16	M(+x+y) (R of 'coarse' at top of keyword menu)			mm	6	mac				
83		--(+x-y) (just L of keyword down arrow)					coat				
84									22	O: evaluate-evaluation-criterion	*** generated evaluation criterion
85									23	O: search-for-help	'value-of-something' **
86									24	=>G: g137 (operator no-change	
87									25	P: p144 (search-for-help	
88									26	S: s155 ((to-be-found write) (value-of-something)	
89									27	O: find-criterion (keyword)	
90									28	=>G: g161 (operator no-change	
91									29	P: p168 (find-criterion	
92									30	S: s178 ((to-be-found write) (value-of-something)	
93									31	O: focus-on-current-window	
94									32	O: evaluate-current-window	
95									33	=>G: g239 (operator no-change	
96									34	P: p246 (evaluate-items-in-window	
97									35	S: s256 ((to-be-found write) (value-of-something)	
98									36	O: read-input (coarse)	
99									37	O: attempt-match	
100									38	O: read-input (comment)	
101									39	O: attempt-match	
102									40	O: read-input (comp_x)	
103									41	O: attempt-match	
104									42	O: read-input (compute)	
105									43	O: attempt-match	
106									44	O: read-input (constant)	
107									45	O: attempt-match	
108									46	O: read-input (cos_x)	
109									47	O: attempt-match	
110									48	O: read-input (cursor)	
111									49	O: attempt-match	
112									50	O: read-input (datain)	
113									51	O: attempt-match	
114									52	O: change-current-window	
115									53	=>G: g423 (operator no-change	
116									54	P: p430 (mac-methods-for-change-current-window	
117									55	S: s438 ((to-be-found write)	
118									56	O: scroll (keyword)	
119									57	=>G: g451 (operator no-change	
120									58	P: p458 (mac-method-of-scroll	
121									59	S: s467 ((to-be-found write)	
122	17	M(+x) to (keyword dn arrow)			mm	7	mr	60	60	O: move-mouse (keyword down)	
123	17	D	keyword menu scrolls		mb	8	mba	61	61	O: press-button	
124	18		keyword menu scrolls								
125	19		keyword menu scrolls								
126	20		keyword menu scrolls								
127	21			write	v	9	v	32			
128	22	U	scrolling stops		mb	10	mba	62	62	O: release-button	
129	22	M(-x+y) (R of item2, keywd menu:wrong (write			mm	11	mi	63	63	O: evaluate-current-window	
130	23			wrong?	v	12	v	63			
131									64	=>G: g507 (operator no-change	
132									65	P: p514 (evaluate-items-in-window	
133									66	S: s526 ((to-be-found write) (value-of-something)	
134									67	O: read-input (wrong)	
135									68	O: attempt-match	
136									69	O: read-input (wrongv)	
137									70	O: attempt-match	
138									71	O: read-input (xin)	
139									72	O: attempt-match	We are left matching operators
140									73	O: read-input (xout)	for we have not states.
141									74	O: attempt-match	
142									75	O: read-input (xaltered)	
143									76	O: attempt-match	
144									77	O: read-input (xansent)	
145									78	O: attempt-match	
146									79	O: read-input (xarrows)	
147									80	O: attempt-match	
148									81	O: read-input (temptemptemp)	
149									82	O: attempt-match	
150									83	O: change-current-window	
151									84	=>G: g696 (operator no-change	
152									85	P: p703 (mac-methods-for-change-current-window	
153									86	S: s711 ((to-be-found write)	
154									87	O: scroll (keyword)	
155									88	=>G: g734 (operator no-change	
156									89	P: p731 (mac-method-of-scroll	
157									90	S: s740 ((to-be-found write)	
158	23	M(+x-y) (keyword up arrow)			mm	13	mr	91	91	O: move-mouse (keyword up)	
159	23	D	keyword menu scrolls		mb	14	mba	92	92	O: press-button	
160	23	U	scrolling stops		mb	15	mba	93	93	O: release-button	
161	24			no	v	16	v	63			
162	24	M(-x-y) (2nd keyword from bottom, xin)			mm	17	mi	94	94	O: evaluate-current-window	
163	24			Na ha haaa	v	18	vuc				
164	25			write.	v	19	v	94			
165									95	=>G: g777 (operator no-change	
166									96	P: p784 (evaluate-items-in-window	
167									97	S: s794 ((to-be-found write) (value-of-something)	
168									98	O: read-input (user-vars)	
169									99	O: attempt-match	
170									100	O: read-input (vbar)	
171									101	O: attempt-match	
172									102	O: read-input (vector)	
173									103	O: attempt-match	
174									104	O: read-input (write)	
175									105	O: attempt-match	
176									106	O: access-item (keyword)	
177									107	=>G: g878 (operator no-change	
178									108	P: p885 (mac-methods-for-access-item	
179									109	S: s893	
180									110	O: click-on-item (i888)	i888 is an unnamed item
181									111	=>G: g899 (operator no-change	
182									112	P: p906 (mac-method-of-click-on-item	
183									113	S: s913	
184	25	M(+y) (3 items up to 'write')			mm	20	mr	114	114	O: move-mouse (keyword unspecified)	

S1	T	Mouse actions	Window actions	Verbal	ST	#	MType	MDC	DC	Soar trace	Comments
185	25	C		mouse pointer to watch	mb	21	mba	115	115	O: click-button	
186	26			'write' help text appears	io						
187	27			'write' becomes bold & moves	io						
188									116	O: evaluate-help-text	
189									117	=>0: g827 (operator no-change	
190									118	P: p934 (evaluate-help-text	
191									119	S: s943 ((accessed write) (value-of-something)	
192									120	O: focus-on-help-text	
193	28			convenient way to	v	22	v	121	121	O: evaluate-current-window	
194	29			write out short	coat						
195	30			of text that loo	coat						
196	31			in your program	coat						
197	32			the text command	coat						
198	33	M(-x-y) (3/4 dn help text scrollbar below elevam				23	mi	121			
199	33			umm...	v	24	short				
200	33	M(-x-y) (bottom R quad of help text win)			mm	25	mi	121			
201	36			show comman	v	26	v	121			
202	38			are used	v	27	v	121			
203	39			to display	v	28	v	121			
204	41			so thats what I	coat						
205									122	=>0: g966 (operator no-change	
206									123	P: p973 (evaluate-prose-in-window	
207									124	S: s984 ((accessed write) (value-of-something)	
208									125	O: read-input	
209									126	O: comprehend	
210									127	O: compare-to-criteria	
211	42	M(+x+y) (mid of keywd scrollbar, over elev)			mm	29	mac				
212	43			is show com	v	30	v	128	128	O: change-search-criterion ((accessed write))	*** changed search criterion 'write' **
213											*** changed search criterion 'show' **
214									129	O: search-for-help	
215									130	=>0: g1025 (operator no-change	
216									131	P: p1032 (search-for-help	
217									132	S: s1043 ((to-be-found show) (value-of-something)	
218									133	O: find-criterion (keyword)	
219									134	=>0: g1049 (operator no-change	
220									135	P: p1056 (find-criterion	
221									136	S: s1066 ((to-be-found show) (value-of-something)	
222									137	O: focus-on-current-window	
223	43	M(-x-y) (-1/2 in R of keyword 'nanscat')			mm	31	mi	138	138	O: evaluate-current-window	goes by but doesn't stop on nanscat,
224	43	-- (+x+y) (keyword scroll bar, above elevator)					coat				
225	43	-- (+y) (above keyword up arrow)					coat				
226									139	=>0: g1092 (operator no-change	micro-codable as:
227									140	P: p1099 (evaluate-items-in-window	162 O: read-input (nanscat)
228									141	S: s1109 ((to-be-found show) (value-of-something)	
229									142	O: read-input (write)	
230									143	O: attempt-match	
231									144	O: read-input (wrong)	
232									145	O: attempt-match	
233									146	O: read-input (wrongv)	
234									147	O: attempt-match	
235									148	O: read-input (xin)	
236									149	O: attempt-match	
237									150	O: read-input (xout)	
238									151	O: attempt-match	
239									152	O: read-input (saltered)	
240									153	O: attempt-match	
241									154	O: read-input (nanscat)	
242									155	O: attempt-match	
243									156	O: read-input (narroom)	
244									157	O: attempt-match	
245									158	O: change-current-window	
246									159	=>0: g1276 (operator no-change	
247									160	P: p1283 (mac-methods-for-change-current-window	
248									161	S: s1291 ((to-be-found show)	
249									162	O: scroll (keyword)	
250									163	=>0: g1304 (operator no-change	
251									164	P: p1311 (mac-method-of-scroll	
252									165	S: s1320 ((to-be-found show)	
253	44	M(-y) (keyword up arrow)			mm	32	mr	166	166	O: move-mouse) (keyword up)	
254	44			so let's ju	v	33	v	128			
255	44	D			mb	34	mba	167	167	O: press-button	
256			keyword menu scrolls & stops								
257	44	U			mb	35	mba	168	168	O: release-button	
258									169	O: evaluate-current-window	
259									170	=>0: g1358 (operator no-change	
260									171	P: p1365 (evaluate-items-in-window	
261									172	S: s1375 ((to-be-found show) (value-of-something)	
262									173	O: read-input (use)	
263									174	O: attempt-match	
264									175	O: read-input (user-vars)	
265									176	O: attempt-match	
266									177	O: read-input (vbar)	
267									178	O: attempt-match	
268									179	O: read-input (vector)	
269									180	O: attempt-match	
270									181	O: read-input (write)	
271									182	O: attempt-match	
272									183	O: read-input (wrong)	
273									184	O: attempt-match	
274									185	O: read-input (wrongv)	
275									186	O: attempt-match	
276									187	O: read-input (xin)	
277									188	O: attempt-match	
278									189	O: change-current-window	
279									190	=>0: g1547 (operator no-change	
280									191	P: p1554 (mac-methods-for-change-current-window	
281									192	S: s1562 ((to-be-found show)	
282									193	O: scroll (keyword)	
283									194	=>0: g1575 (operator no-change	
284									195	P: p1582 (mac-method-of-scroll	
285									196	S: s1591 ((to-be-found show)	
286	44	D			mb	36	mba	197	197	O: press-button	
287			keyword menu scrolls & stops		io						

51	T	Mouse actions	Window actions	Verbal	ST #	MType	MDC	DC	Soar trace	Comments
288	44	U			mb 37	mba	198	198	O: release-button	
289	45			sure I know how	cont					Continued from what matched dc 128
290								199	O: evaluate-current-window	
291								200	=>G: g1622 (operator no-change	
292								201	P: p1629 (evaluate-items-in-window	
293								202	S: s1639 ((to-be-found show) (value-of-something)	
294								203	O: read-input (tan)	
295								204	O: attempt-match	
296								205	O: read-input (tent)	
297								206	O: attempt-match	
298								207	O: read-input (touch)	
299								208	O: attempt-match	
300								209	O: read-input (unit)	
301								210	O: attempt-match	
302								211	O: read-input (use)	
303								212	O: attempt-match	
304								213	O: read-input (user-vars)	
305								214	O: attempt-match	
306								215	O: read-input (vbar)	
307								216	O: attempt-match	
308								217	O: read-input (vector)	
309								218	O: attempt-match	
310								219	O: change-current-window	
311								220	=>G: g1811 (operator no-change	
312								221	P: p1810 (mac-methods-for-change-current-window	
313								222	S: s1826 ((to-be-found show)	
314								223	O: scroll (keyword)	
315								224	=>G: g1839 (operator no-change	
316								225	P: p1846 (mac-method-of-scroll	
317								226	S: s1855 ((to-be-found show)	
318	45	D			mb 38	mba	227	227	O: press-button	these scrolls, all within 1 s in the human,
319			keyword menu scrolls & stops							don't correspond to this novice like model.
320	45	U			mb 39	mba	228	228	O: release-button	-- some of this will chunk up in the human.
321								229	O: evaluate-current-window	
322								230	=>G: g1886 (operator no-change	
323								231	P: p1893 (evaluate-items-in-window	
324								232	S: s1903 ((to-be-found show) (value-of-something)	
325								233	O: read-input (string)	
326								234	O: attempt-match	
327								235	O: read-input (syntaxlevel)	
328								236	O: attempt-match	
329								237	O: read-input (tan)	
330								238	O: attempt-match	
331								239	O: read-input (tent)	
332								240	O: attempt-match	
333								241	O: read-input (touch)	
334								242	O: attempt-match	
335								243	O: read-input (unit)	
336								244	O: attempt-match	
337								245	O: read-input (use)	
338								246	O: attempt-match	
339								247	O: read-input (user-vars)	
340								248	O: attempt-match	
341								249	O: change-current-window	
342								250	=>G: g2075 (operator no-change	
343								251	P: p2082 (mac-methods-for-change-current-window	
344								252	S: s2090 ((to-be-found show)	
345								253	O: scroll (keyword)	
346								254	=>G: g2103 (operator no-change	
347								255	P: p2110 (mac-method-of-scroll	
348								256	S: s2119 ((to-be-found show)	
349	46	D			mb 40	mba	257	257	O: press-button	
350			keyword menu scrolls & stops		io					
351	46	U			mb 41	mba	258	258	O: release-button	
352	47	M(-x-y) ('showh' 2nd f/ top of list)			mm 42	mi	259	259	O: evaluate-current-window	
353	47	M(+x+y) (-1/4in R of 'showb', the 1st item)			mm 43	mi	259			
354	48	M(+x) (just R & below keyword up arrow)			mm 44	mi	259			partial move to get ready to scroll
355	48	-- (+x) (up arrow)			mm	cont				again, Fitts law!
356	49			show R show v	45	v	259			
357								260	=>G: g2150 (operator no-change	
358								261	P: p2157 (evaluate-items-in-window	
359								262	S: s2167 ((to-be-found show) (value-of-something)	
360								263	O: read-input (showb)	This is a patched in trace
361								264	O: attempt-match	from dc 263 to 277
362								265	O: read-input (showh)	
363								266	O: attempt-match	
364								267	O: read-input (showo)	
365								268	O: attempt-match	
366								269	O: read-input (showt)	
367								270	O: attempt-match	
368								271	O: read-input (sign)	
369								272	O: attempt-match	
370								273	O: read-input (sin)	
371								274	O: attempt-match	
372								275	O: read-input (sinh)	
373								276	O: attempt-match	
374								277	O: read-input (sine)	
375								278	O: attempt-match	
376								279	O: change-current-window	
377								280	=>G: g2339 (operator no-change	
378								281	P: p2346 (mac-methods-for-change-current-window	
379								282	S: s2354 ((to-be-found show)	
380								283	O: scroll (keyword)	
381								284	=>G: g2367 (operator no-change	
382								285	P: p2374 (mac-method-of-scroll	
383								286	S: s2383 ((to-be-found show)	
384	51	D			mb 46	mba	287	287	O: press-button	
385	51	U			mb 47	mba	288	288	O: release-button	
386								289	O: evaluate-current-window	
387								290	=>G: g2415 (operator no-change	
388								291	P: p2422 (evaluate-items-in-window	
389								292	S: s2432 ((to-be-found show) (value-of-something)	
390								293	O: read-input (scalex)	

S1	T	Mouse actions	Window actions	Verbal	ST #	MType	MDC	DC	Soar trace	Comments
391								294	0: attempt-match	
392								295	0: read-input (scaley)	
393								296	0: attempt-match	
394								297	0: read-input (set)	
395								298	0: attempt-match	
396								299	0: read-input (setfile)	
397								300	0: attempt-match	
398								301	0: read-input (show)	
399								302	0: attempt-match	
400								303	O: access-item (keyword)	
401								304	=>O: g2527 (operator no-change	
402								305	P: p2534 (mac-methods-for-access-item	
403								306	S: s2542	
404								307	O: click-on-item (i2537)	
405								308	=>O: g2548 (operator no-change	
406								309	P: p2555 (mac-method-of-click-on-item	
407								310	S: s2562	
408	52	M(-x-y) ('showb', 3rd from bot, keyword menu)			mm 48	mr	311	311	O: move-mouse (keyword unspecified)	
409		-- (+y) ('show')				coat				
410	53	C			mb 49	mba	312	312	O: click-button	
411								313	O: evaluate-help-text	
412								314	=>O: g2576 (operator no-change	
413								315	P: p2583 (evaluate-help-text	
414								316	S: s2592 ((accessed show) (value-of-something)	
415								317	O: focus-on-help-text	
416	53			I don't kno	v 50	v	318	318	O: evaluate-current-window	
417	54			show binary, pro		coat				
418	55			show expression.		coat				
419	56			is an infinite f		coat				
420	56	M(-y) (middle R side of help text)			mm 51	mi	318			
421	57	M(-y) (a little lower)			mm 52	mi	318			
422	58			uhm...	v 53	abort				
423	58	M(-y) (a little lower)			mm 54	mi	318			
424	60			but I wonde	v 55	v	318			
425	62	M(+x-y) (just L of dn arrow for help txt win)			mm 56	mmc				
426	63			for	v 57	v	318			
427								319	=>O: g2615 (operator no-change	
428								320	P: p2622 (evaluate-prose-in-window	
429								321	S: s2633 ((accessed show) (value-of-something)	
430								322	O: read-input	
431								323	O: comprehend	
432								324	O: compare-to-criteria	
433								325	O: change-current-window	
434								326	=>O: g2658 (operator no-change	
435								327	P: p2665 (mac-methods-for-change-current-window	
436								328	S: s2673 ((accessed show)	
437								329	O: scroll (help-text)	
438								330	=>O: g2685 (operator no-change	
439								331	P: p2692 (mac-method-of-scroll	
440								332	S: s2701 ((accessed show)	
441	63	M(+x) (down arrow)			mm 58	mr	333	333	O: move-mouse (help-text down)	
442	64	D	help text win. scrolls		mb 59	mba	334	334	O: press-button	
443	65			markers	v 60	v	318			
444	65	U			mb 61	mba	335	335	O: release-button	
445								336	O: evaluate-current-window	
446								337	=>O: g2744 (operator no-change	
447								338	P: p2751 (evaluate-prose-in-window	
448								339	S: s2762 ((accessed show) (value-of-something)	
449								340	O: read-input	
450								341	O: comprehend	
451								342	O: compare-to-criteria	
452								343	O: change-current-window	
453								344	=>O: g2707 (operator no-change	
454								345	P: p2794 (mac-methods-for-change-current-window	
455								346	S: s2802 ((accessed show)	
456								347	O: scroll (help-text)	
457								348	=>O: g2814 (operator no-change	
458								349	P: p2821 (mac-method-of-scroll	
459								350	S: s2830 ((accessed show)	
460	66	D	help text win. scrolls		mb 62	mba	351	351	O: press-button	
461	67	U			mb 63	mba	352	352	O: release-button	
462	68			okay	v 64	v	336			@is - This had been 332, a state?
463								353	O: evaluate-current-window	30-jun-92 FER
464								354	=>O: g2864 (operator no-change	
465								355	P: p2871 (evaluate-prose-in-window	
466								356	S: s2882 ((accessed show) (value-of-something)	
467								357	O: read-input	
468								358	O: comprehend	
469								359	O: compare-to-criteria	
470								360	O: change-current-window	
471								361	=>O: g2907 (operator no-change	
472								362	P: p2914 (mac-methods-for-change-current-window	
473								363	S: s2922 ((accessed show)	
474								364	O: scroll (help-text)	
475								365	=>O: g2934 (operator no-change	
476								366	P: p2941 (mac-method-of-scroll	
477								367	S: s2950 ((accessed show)	
478	68	D	help text win. scrolls		mb 65	mba	368	368	O: press-button	
479	69	U			mb 66	mba	369	369	O: release-button	
480								370	O: evaluate-current-window	
481								371	=>O: g2984 (operator no-change	
482								372	P: p2991 (evaluate-prose-in-window	
483								373	S: s3002 ((accessed show) (value-of-something)	
484								374	O: read-input	
485								375	O: comprehend	
486								376	O: compare-to-criteria	
487								377	O: change-current-window	
488								378	=>O: g3027 (operator no-change	
489								379	P: p3034 (mac-methods-for-change-current-window	
490								380	S: s3042 ((accessed show)	
491								381	O: scroll (help-text)	
492								382	=>O: g3054 (operator no-change	
493								383	P: p3061 (mac-method-of-scroll	

S1	T	Mouse actions	Window actions	Verbal	ST	S	MType	MDC	DC	Soar trace	Comments
494									384	S: s3070 ((accessed show)	
495	70	D		help text win. scrolls		mb	67	mba 385	385	O: press-button	
496	72			well, I'll		v	68	v 370			
497	72	U				mb	69	mba 386	386	O: release-button	
498									387	O: evaluate-current-window	
499									388	=>O: g3104 (operator no-change	
500									389	P: p3111 (evaluate-prose-in-window	
501									390	S: s3122 ((accessed show) (value-of-something)	
502									391	O: read-input	
503									392	O: comprehend	
504									393	O: compare-to-criteria	
505									394	=>O: state no-change	
506									395	=>O: g3152 (goal no-change	
507									396	=>O: g3159 (goal no-change	
508									397	=>O: g3166 (goal no-change	
509									398	=>O: g3173 (goal no-change	
510									399	=>O: g3180 (goal no-change	

2 Displays of each analytical measure for each episode of Browser-Soar

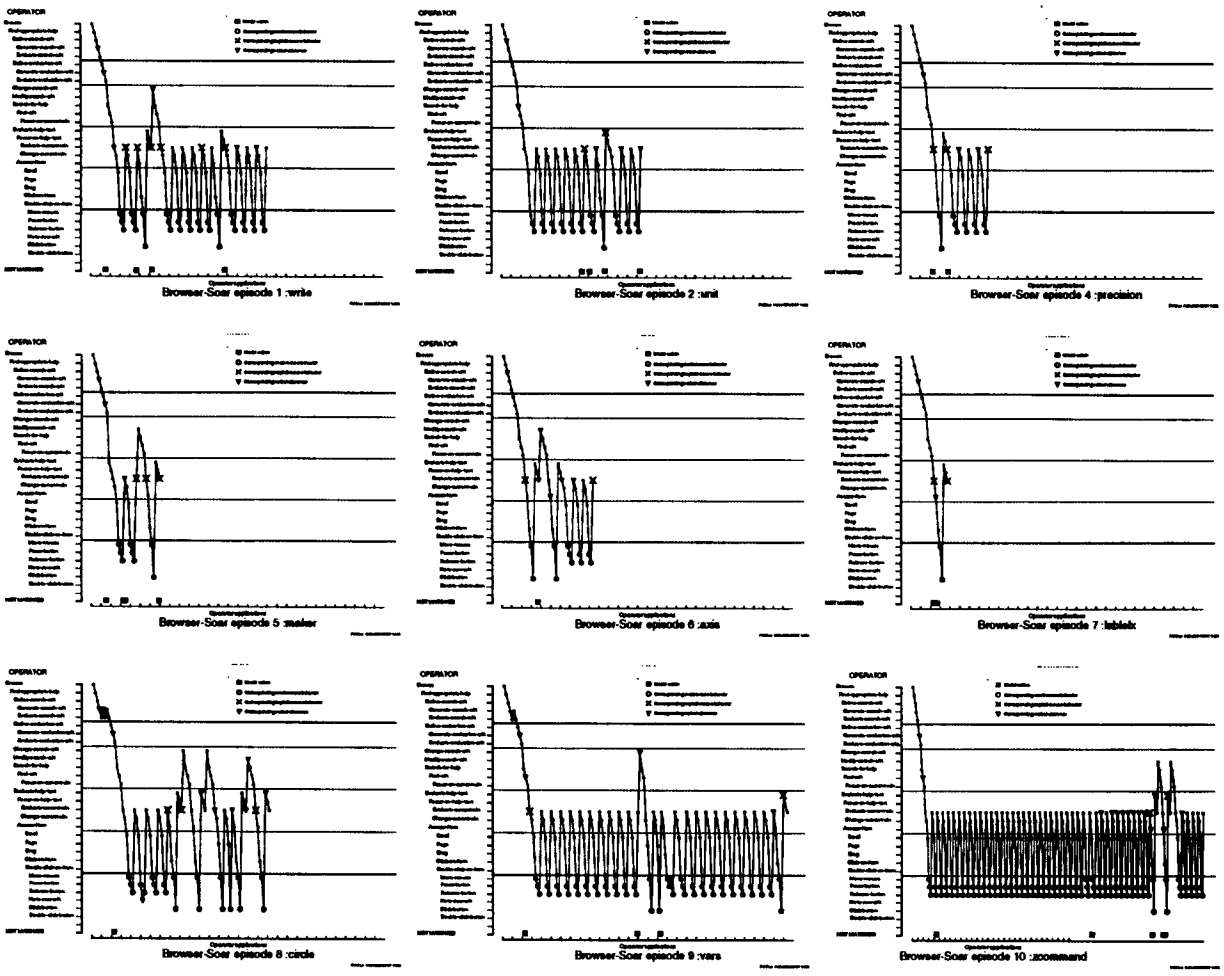
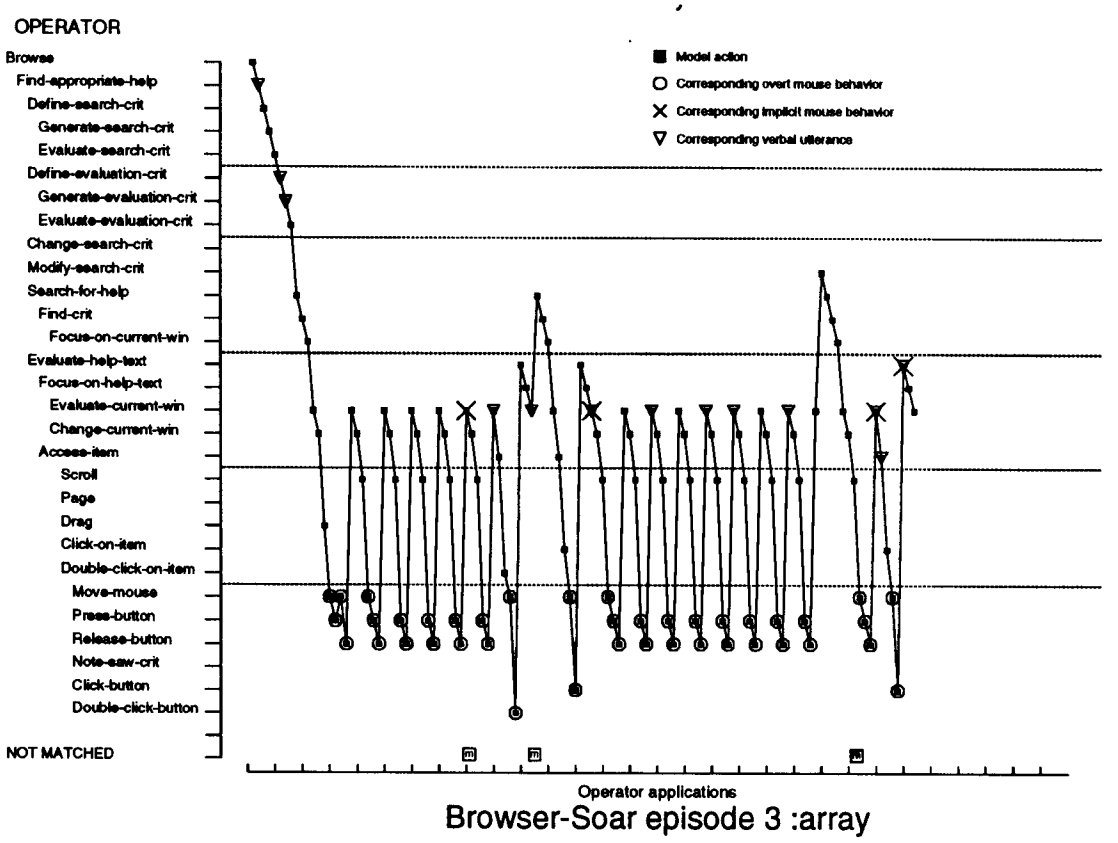


Figure 46: The operator support displays for each of the episodes.



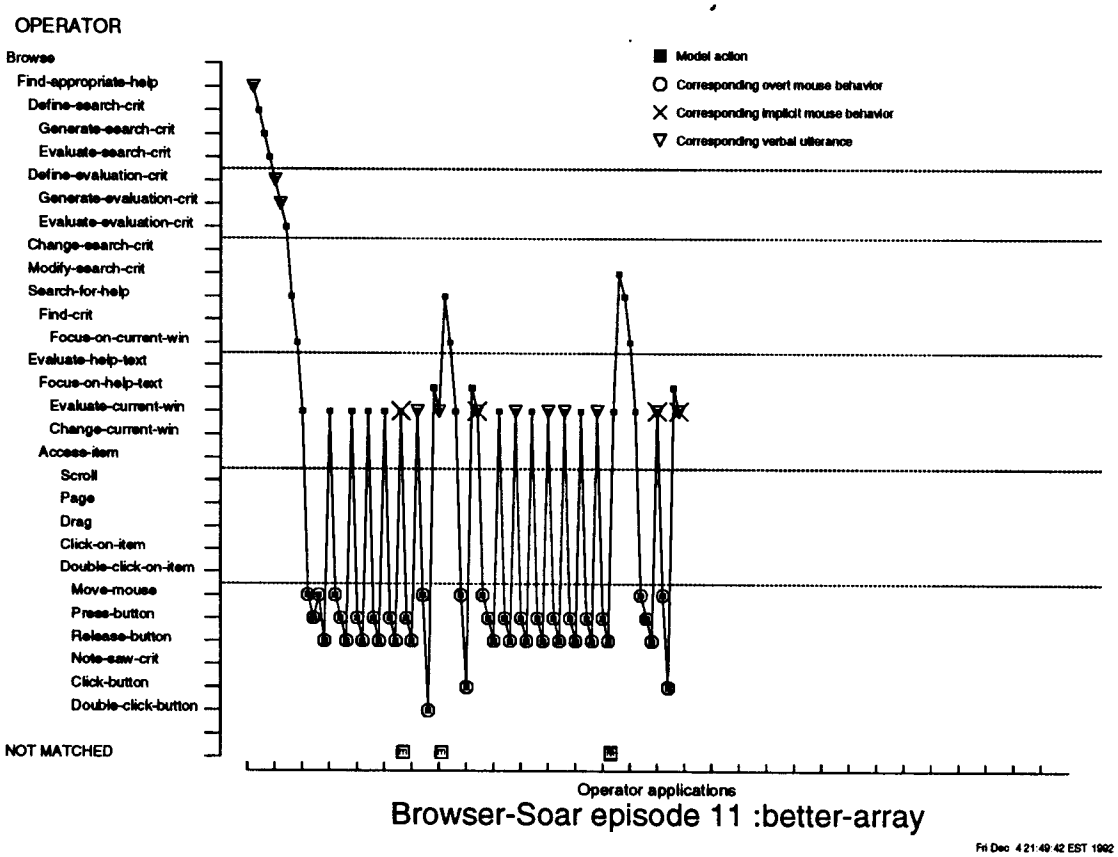


Figure 46: The operator support displays for each of the episodes (cont.).

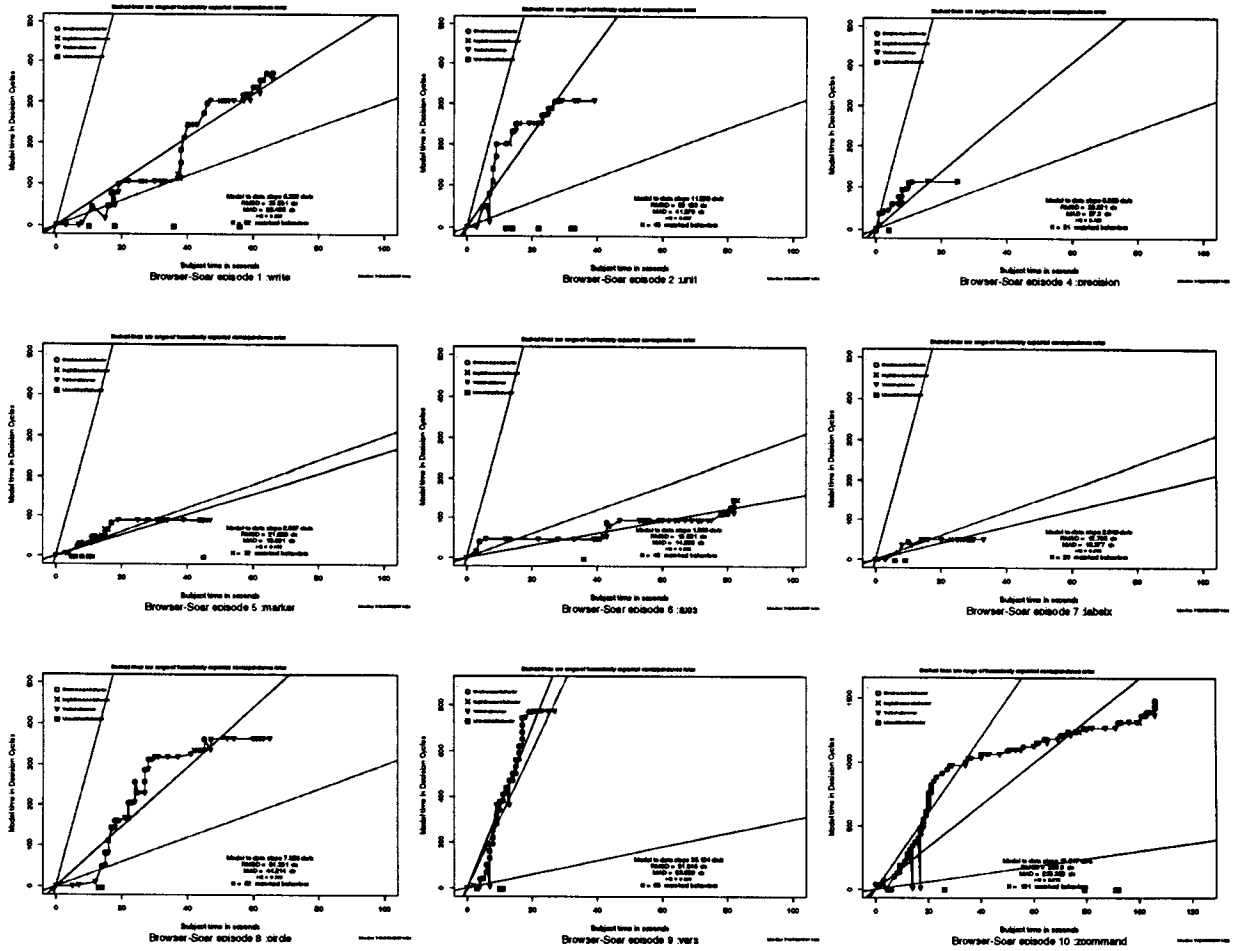
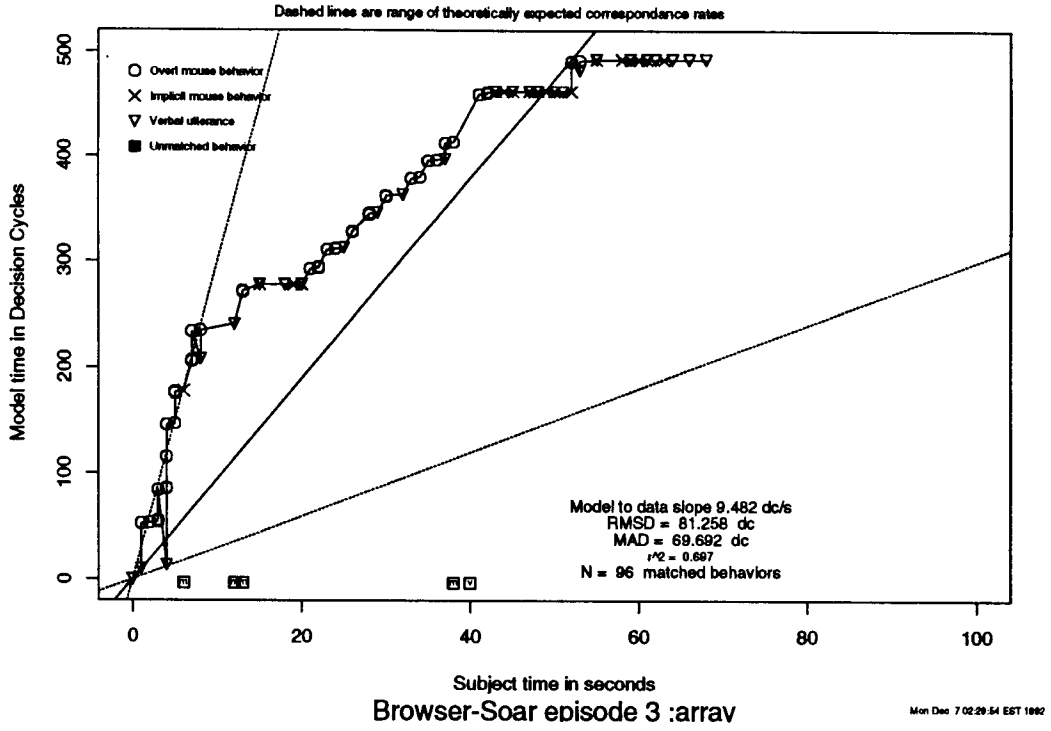


Figure 47: The relative processing rates displays based on decision cycles for each of the episodes.



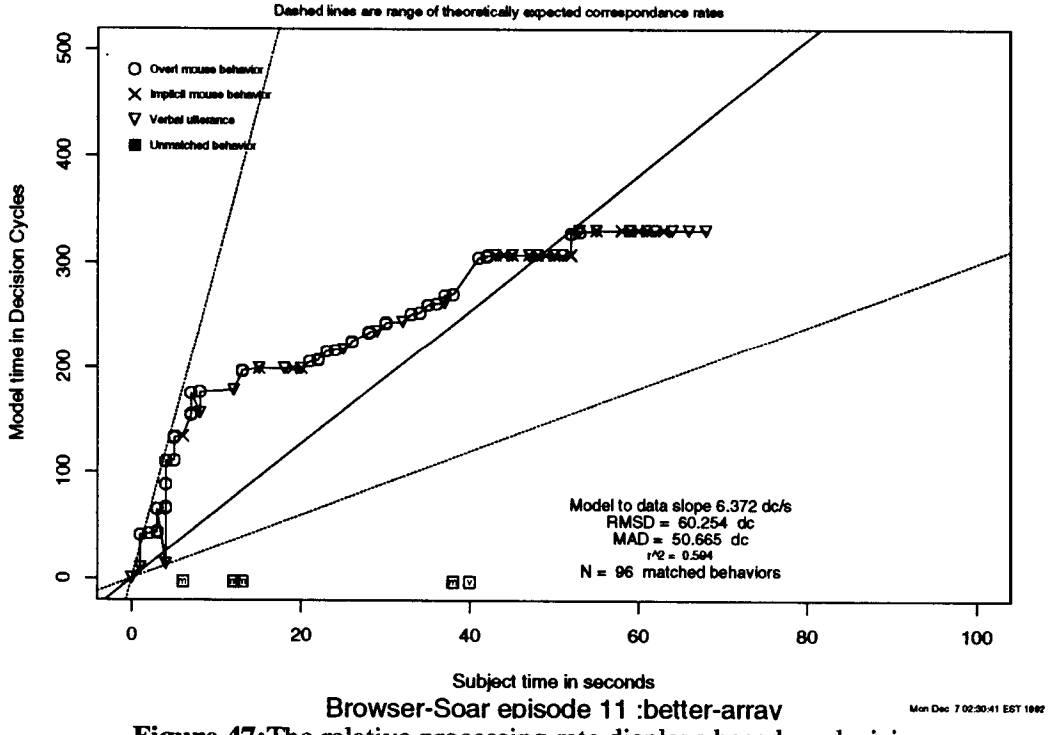


Figure 47: The relative processing rate displays based on decision cycles for of the episodes (cont.).

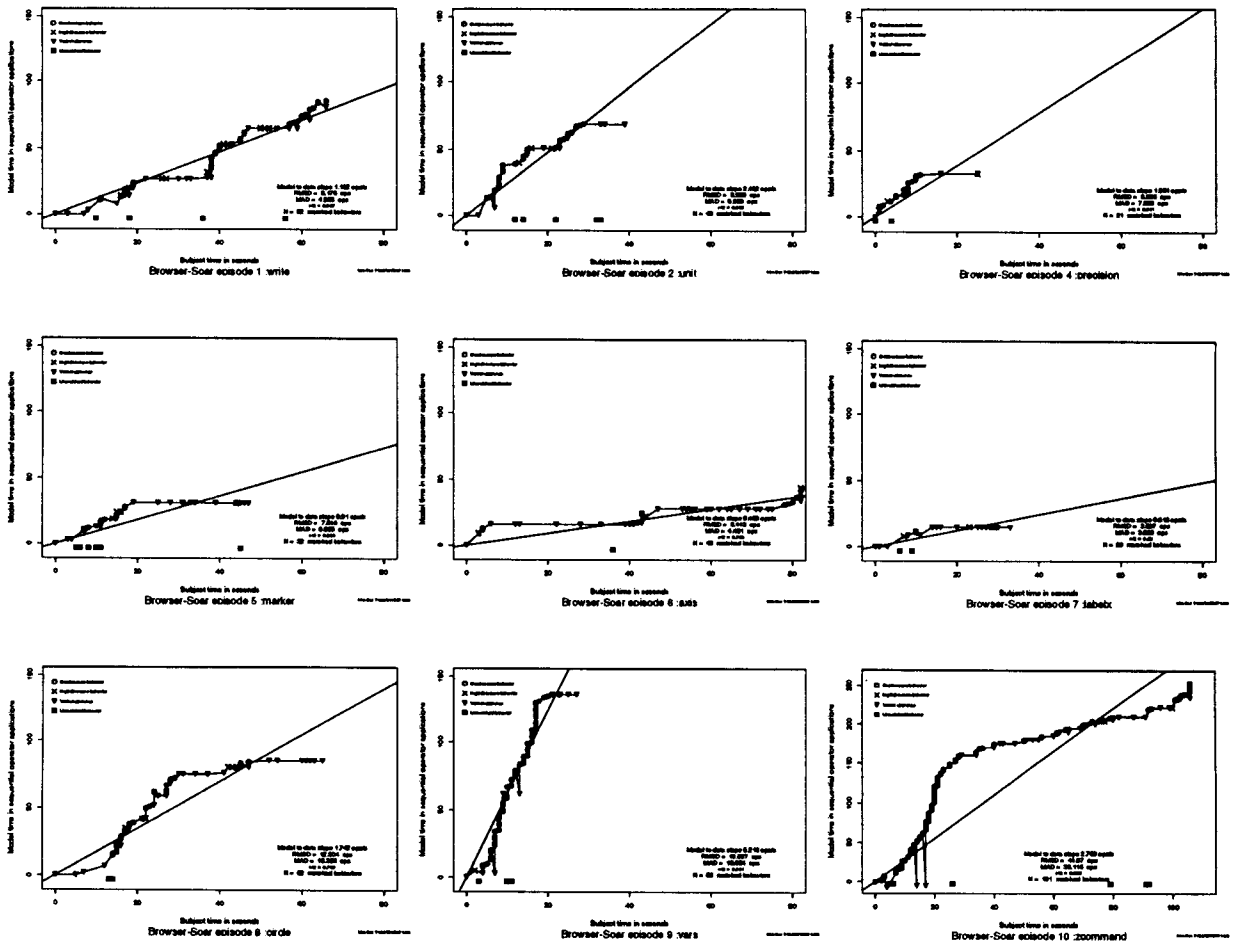
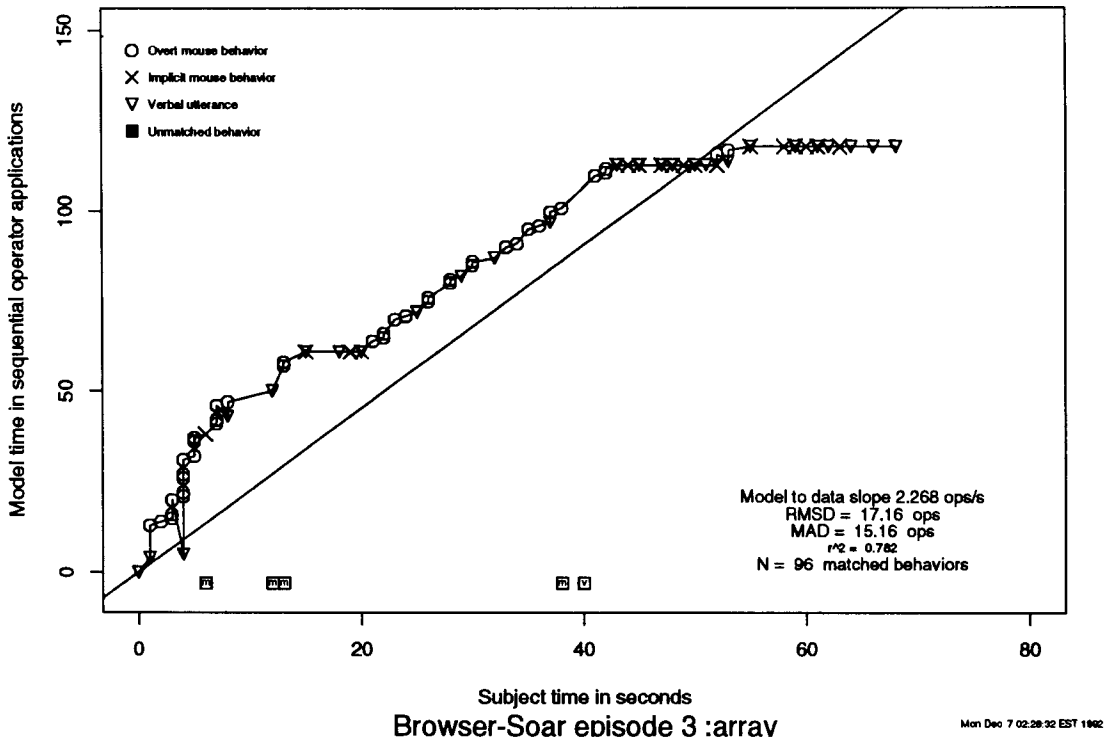


Figure 48: The relative processing rates displays based on operator applications for each of the episodes.



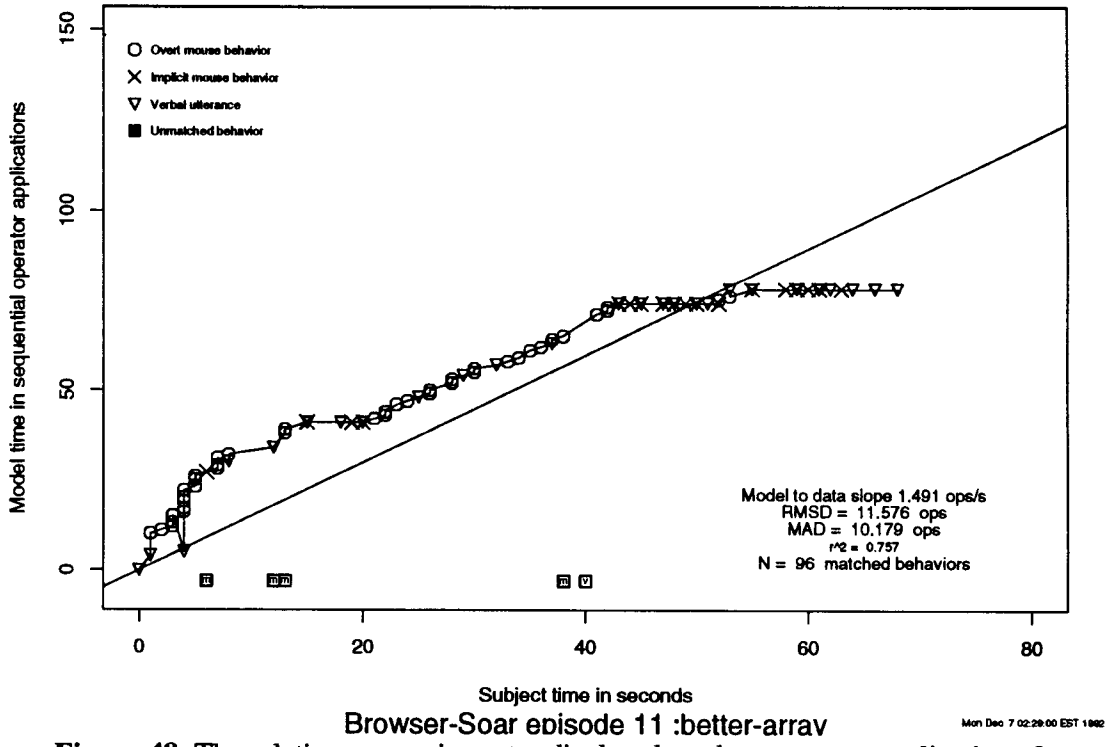


Figure 48: The relative processing rates displays based on operator applications for each of the episodes (cont.).