Behavior Research Methods, Instruments, and Computers
2005, 37 (1), 71-81

Dismal: A spreadsheet for sequential data analysis and HCI experimentation

FRANK E. RITTER and ALEXANDER B. WOOD
Pennsylvania State University, University Park, Pennsylvania

Dismal is a spreadsheet that works within GNU Emacs, a widely available programmable editor. Dismal has three features of particular interest to those who study behavior: (1) the ability to manipulate and align sequential data, (2) an open architecture that allows users to expand it to meet their particular needs, and (3) an instrumented and accessible interface for studies of human–computer interaction (HCI). Example uses of each of these capabilities are provided, including cognitive models that have had their sequential behavior aligned with subject's protocols, extensions useful for teaching and doing HCI design, and studies in which keystroke logs from the timing package in Dismal have been used.

There are several facilities that researchers would like to have when studying behavior. Nearly all the time, they would like to record behavior. After that, they would like to analyze it. These analyses can take standard forms, such as computing means, but can also involve exploring the data. In the case of sequential data, analysis includes examining the sequence for patterns and aligning the sequence with predictions of that sequence. Sometimes, these analyses are common and are carried out by others. In this case, the analyses are often supported with existing computational tools. In other cases, the analyses are relatively or completely novel.

We present a tool developed for working with human–computer interface data and with protocol data relevant to cognitive models. This tool, called *Dismal*, provides support for several common analyses and support for creating novel analyses.

Dismal is a spreadsheet that was designed to gather and analyze behavioral data. Figure 1 provides an example display. Dismal has three features of particular interest to those studying behavior: (1) the ability to manipulate and align sequential data, (2) an open architecture that allows users to expand it to meet their particular needs, and (3) an instrumented and accessible interface for studies of human–computer interaction (HCI).

Dismal extends the GNU Emacs editor (Free Software Foundation, 1988), using GNU Emacs' extension language, Emacs Lisp. There is a large potential user community for Dismal, since GNU Emacs is a popular text editor that is commonly installed on UNIX and Linux systems. Dismal has been accepted for inclusion as part of the GNU Emacs distribution, so it will be included automatically on many machines in the future. Dismal does not rely much on operating system calls, so it is pretty portable. Thus, users on PCs running Emacs under Windows/* and under Linux and Macintoshes running Emacs can all use Dismal if they have a recent version of Emacs (19 or higher). Dismal has worked with versions of the XEmacs version of Emacs, but there continue to be small problems as XEmacs evolves differently from Emacs. These problems have been pretty easy to fix. Expert users of Xemacs should be able to modify the source code to make it run with their version.

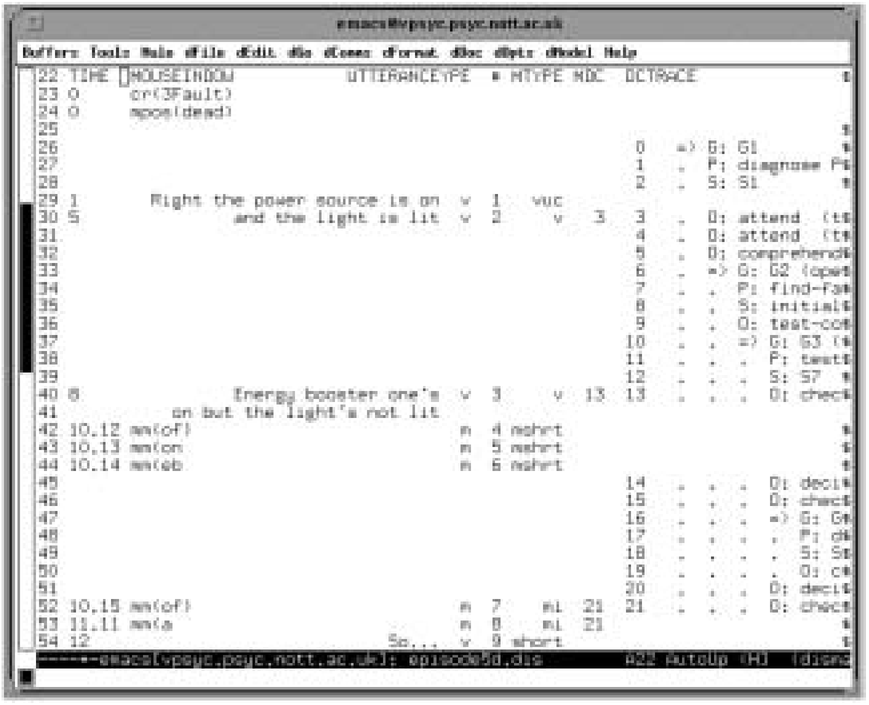
Example uses of these capabilities are provided to illustrate Dismal's use in sequential data analysis, its use for teaching and doing HCI design, and an application of using keystroke logs and alias creation tools. Dismal also illustrates some new features that designers of other spreadsheets and related tools may wish to include in their systems (and which some have). To introduce Dismal, we describe its general capabilities before describing its particular features of interest.

GENERAL CAPABILITIES

Emacs supports editing different types of files by mapping the standard set of editor keystroke commands to a contextualized set with a template, called a *major mode*. These modes, implemented as a set of Emacs Lisp func-

Initial work was sponsored in part by a training grant from the Air Force Office of Scientific Research, Bolling AFB, DC, and in part by the Avionics Laboratory, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U. S. Air Force, Wright-Patterson AFB, under Contract F33615-90-C-1465. David Fox wrote the first version and worked jointly on the initial development when he was at New York University. Dismal's later development was partially supported by the Economic and Social Science Research Council's Centre for Research in Development, Instruction, and Training, by Grant SPG 9018736 from the Joint Council Initiative in HCI and Cognitive Science, by a grant from DERA, Contract 2024/004, by Lockheed Martin, and by the U.S. Office of Naval Research, Awards N000140110547 and N000140310248. Numerous users have contributed bug reports. Erik Altmann, Stephen Eglen, Nigel Jenkins, David Lamkins, Sarah Nichols, and Roberto Ong have contributed extensive additions. Angela Barnhill, Andy Edmonds, Ying Guo, Vicenç Quera, and an anonymous reviewer have provided useful comments. Correspondence concerning this article should be addressed to F. E. Ritter, School of Information Sciences and Technology, Pennsylvania State University, University Park, PA 16802 (e-mail: frank.ritter@psu.edu).

Note—This article was accepted by the previous editor, Jonathan Vaughan.


emacs@vpsyc.psyc.nott.ac.uk
Buffers Tools Mule dFile dEdit dGo dComms dFormat dDoc dOpts dModel Help

```
22 TIME  MOUSEINDOW            UTTERANCEYPE   * MTYPE MOC   DCTRACE              $
23 0     cr(3Fault)
24 0     mpos(dead)
25                                                                               $
26                                                          0    => G: G1        $
27                                                          1    .  P: diagnose P$
28                                                          2    .  S: S1        $
29 1        Right the power source is on    v  1   vuc
30 5               and the light is lit     v  2     v   3  3    .  O: attend  (t$
31                                                          4    .  O: attend  (t$
32                                                          5    .  O: comprehend$
33                                                          6    .  => G: G2 (ope$
34                                                          7    .  .  P: find-fa$
35                                                          8    .  .  S: initial$
36                                                          9    .  .  O: test-co$
37                                                          10   .  .  => G: G3 ($
38                                                          11   .  .  .  P: test$
39                                                          12   .  .  .  S: S7  $
40 8               Energy booster one's     v  3     v  13  13   .  .  .  O: chec$
41            on but the light's not lit
42 10.12 mn(of)                             m  4 mshrt                           $
43 10.13 mn(on                              m  5 mshrt                           $
44 10.14 mn(eb                              m  6 mshrt                           $
45                                                          14   .  .  .  O: deci$
46                                                          15   .  .  .  O: chec$
47                                                          16   .  .  .  => G: G$
48                                                          17   .  .  .  .  P: d$
49                                                          18   .  .  .  .  S: S$
50                                                          19   .  .  .  .  O: c$
51                                                          20   .  .  .  O: deci$
52 10.15 mn(of)                             m  7    mi  21  21   .  .  .  O: chec$
53 11.11 mn(a                               m  8    mi  21                       $
54 12                             So...     v  9 short                           $
-----*-emacs[vpsyc.psyc.nott.ac.uk]: episode5d.dis        A22 AutoUp (H)  (dismo
```

Figure 1. Dismal running within an X windows environment, showing sample data to model predictions alignment, taken from Ritter and Bibby (2001, Episode 5).

tions, adapt the basic editing commands to suit what is being edited. For example, the command, bound to a keystroke, to move to the next paragraph (meta-control-f) should mean different things, since the "paragraphs" are different when text is edited and when a C program is edited.

Dismal is implemented as one of these major modes.[1] Dismal changes nearly all the commands, menus, and key bindings to have Emacs work like a spreadsheet with a given file. So, in the same manner that the user could move forward a word with a key binding, he or she now can move forward a spreadsheet cell.

Dismal includes most of the major functions that one now expects from a spreadsheet, such as (1) the addition, deletion, clearing, and pasting of cells, rows, columns, and ranges, (2) formula entry and evaluation, (3) movement within the spreadsheet with keystrokes and mouse movements, and (4) the ability to format each cell's display. The more global layout of spreadsheets and their components are reconfigurable by adjusting the location, width, and alignment of columns. As with any spreadsheet, the ability to put formulas into cells and to create new functions allows many analyses to be performed directly. For example, several functions have been added to translate grades between letters and number equivalents.

Users can interact with the spreadsheet through keystrokes, menus, and function calls. Commands are available on keystrokes, as well as on pull-down menus under X Windows and a continuously available command line menu (Ritter & Ong, 1994). The commands are implemented as Emacs Lisp functions, so further extensions or customizations can be created, such as the keystroke-level model below, and a few users have done these types of extensions.

FACILITIES FOR SEQUENTIAL DATA ANALYSIS

Sequential data—that is, sequences of actions, utterances, or other behaviors that have order—have been used to study behavior in many areas. These areas include animal behavior, conversations and children's language acquisition (MacWhinney, 1991; MacWhinney & Snow, 1990), HCI (Finlay & Harrison, 1990; Ritter & Larkin, 1994; Sanderson, Scott, et al., 1994), and problem solving (Newell & Simon, 1972). There are several reviews available (Clarke, 1990; Fielding & Lee, 1991; Gottman & Roy, 1990; Hilbert & Redmiles, 2000; Ritter & Larkin, 1994). Sequential data let us examine what occurs and start to derive descriptions of behavior that are more complex than simple reaction times. Patterns of behavior, including strategies, precursors, and causality, can start to be explored with sequential data.

Sanderson and Fisher (1994) have noted that there are several types of exploratory sequential data analysis (ESDA). Some analyses arise when sequential data are displayed and the temporal patterns are visible. More advanced analyses include looking for cycles, computing transition matrices, and aligning two sequences. Hilbert and Redmiles's (2000) review of sequential data analysis techniques provides seven categories. Dismal includes support for their Sync & Search, Counts & Stats, Transforms, and Sequence comparisons. Dismal can be extended to include their Sequence detection and Sequence characterization. If extended in these ways, Dismal might also be seen to provide their Integrated support.

Of particular interest to us, sequential data have also been used to develop and test cognitive models in a variety of domains, including, for example, problem solving (Newell & Simon, 1972; Ritter & Bibby, 2001) and menu use (Byrne, 2001). Dismal was developed to work directly with sequences of verbal and nonverbal protocols to build and test models of cognition. It should also support the analysis of other types of sequential data.

Dismal provides support for simple and complex ESDA analysis, ranging from the simple analyses of computing word counts, to searching for lines matching given patterns, to semiautomatically assigning codes to segments, to the more advanced analyses of automatically aligning protocols (subjects' verbal utterances or a series of actions) with the sequential actions of an information-processing cognitive model. These activities are parts of building and testing cognitive models (Ritter, 2004; Ritter & Larkin, 1994), but they are also steps used in theory building and testing in general. We will explain these capabilities in more detail in the following sections, including the desirability of a tabular display for such data.

Simple Measures and Simple Analyses of Sequences

Dismal's tabular display supports simple, initial visual inspection of sequential data. The tabular display of the information sequences, their arrangement side-by-side, and the ability to print out the correspondences support ESDA. Most tools for manipulating protocols include the ability to segment protocols appropriately and then to assign codes to the segments as a step in model formation (Ritter & Larkin, 1994). The general spreadsheet functions support the preliminary step of resegmenting verbal protocols by adding additional cells and breaking an initial segment of protocol text into segments, as appropriate.

Through a menu or keystroke command, a segment can be assigned one of a set of codes. These codes can be entered by the user as they code or loaded from a saved file of previously used codes representing, for example, an ACT–R or Soar model. When we have taken the codes directly from a cognitive model in the Developmental Soar Interface (Ritter & Larkin, 1994), the savings did not seem to be large, because the codes have to be imported only once. But interfacing this part of Dismal with a task representation tool, such as CTTE (Paternò & Santoro, 2002), remains an interesting possibility.

With the codes in hand, the sequences of behavior can then be analyzed. The underlying spreadsheet provides formulas for computing simple aggregate measures on the coding. A database facility (somewhat similar to the database facilities in Excel) is included in GNU Emacs, including the ability to display all segments that do (or do not) match a given pattern. Emacs supports the use of regular expressions (Friedl, 2002) to describe patterns.

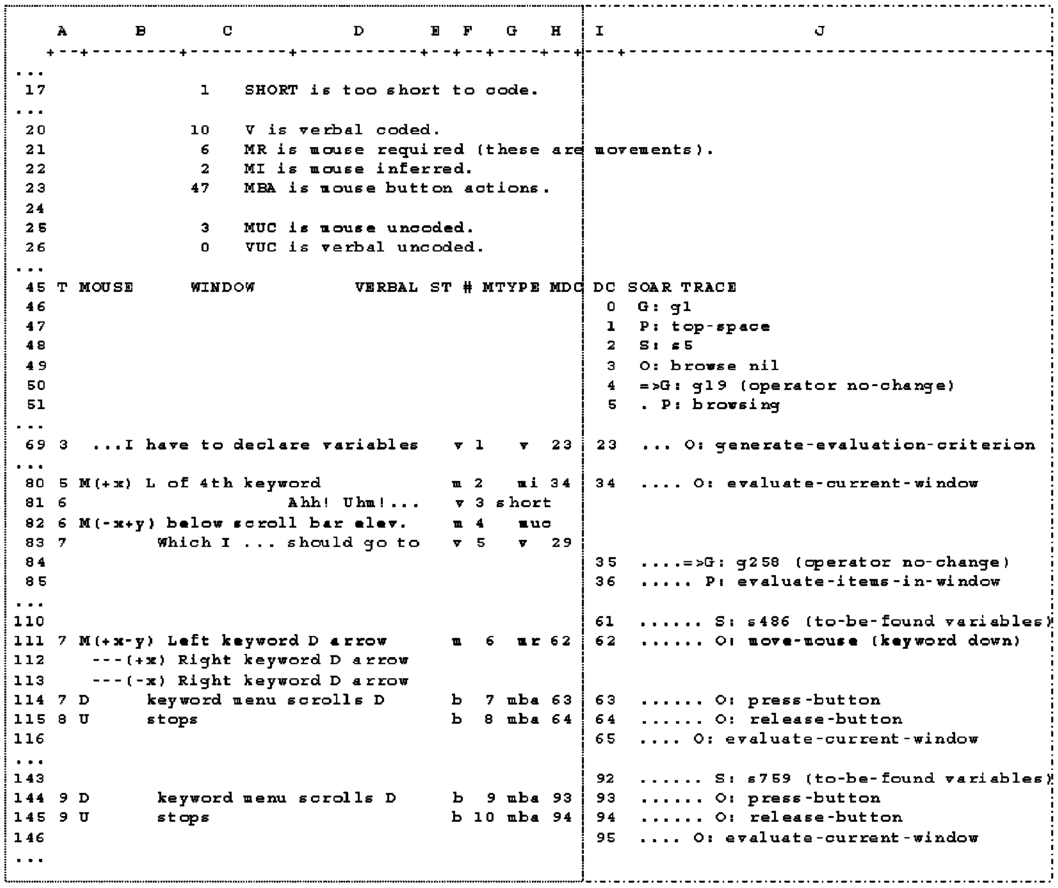
To support these analyses, Dismal introduces the idea of *metacolumns*. This is necessary when the set of variables making up the theory's predictions or the set of data variables span more than one simple spreadsheet column. For example, most analyses will use two columns to hold the theory's predictions—one to hold the predicted actions and one to hold their predicted times. These two simple columns should stay together as cells are added or aligned. To assist this process, Dismal allows columns to be linked into metacolumns. During alignment operations the simple columns making up the metacolumns are manipulated together so that cells within a metacolumn remain aligned. In the examples included here, the columns on the right are the data columns, and the columns on the left hold the components that make up the model's predictions (informally called a *trace* in the cognitive-modeling community[2]), but this choice of sides is arbitrary.

Figure 2 shows an example alignment of columns of human behavior data and model performance. In the figure, columns A–H are grouped into one metacolumn (the data), and I and J are grouped into a metacolumn holding the model's predictions. This grouping is shown by the boxes overlaid onto the figure highlighting the metacolumns. The alignment algorithms explained in the next section thus move columns A–H together and I and J together. A count of the codes in this alignment is included in lines 17 and 20–26, where a function counts the codes used in the alignment (from column G, "MTYPE").

In this display, lines 0 through 44 are used as a header, but like any other spreadsheet, these rows are not fixed as header rows. Line 45 holds a ruler, which names the columns. When the user scrolls, the contents of the ruler row are redrawn as the top line of the display. This ruler can be adjusted to any row or can be omitted.

As a spreadsheet, Dismal provides a tabular display of these codes and correspondences between sets (columns) of codes. The tabular display helps the analyst see how to align the two columns (e.g., predicted codes and observed codes) and to understand the alignment by providing the context of each match (the sequence of items both before and after it), along with a visual operation (scanning a row) to identify the prediction and the data that are paired. The tabular display also shows how much of the predicted codes are matched and unmatched to the observed codes, and it starts to show patterns across columns.

This tabular display, with the field names shown as column headings, provides a compact way to display large amounts of data. It allows more data to be displayed on the screen than a database approach can provide. The tabular display reflects the underlying matrix organization of the data into rows of segments that each include several fields displayed as columns. Automatic alignment programs and semiautomatic tools thus have a uniform and appropriate data structure holding the segments and protocols to be manipulated.



```
    A      B       C           D       E  F   G   H  I                  J
   +--+--------+---------+-----------+--+--+----+--+--+----------------------------------
...
 17              1   SHORT is too short to code.
...
 20             10   V is verbal coded.
 21              6   MR is mouse required (these are movements).
 22              2   MI is mouse inferred.
 23             47   MBA is mouse button actions.
 24
 25              3   MUC is mouse uncoded.
 26              0   VUC is verbal uncoded.
...
 45 T MOUSE      WINDOW          VERBAL ST # MTYPE MDC DC  SOAR TRACE
 46                                                    0  G: g1
 47                                                    1  P: top-space
 48                                                    2  S: s5
 49                                                    3  O: browse nil
 50                                                    4  =>G: g19 (operator no-change)
 51                                                    5  . P: browsing
...
 69 3  ...I have to declare variables   v 1    v  23  23   ... O: generate-evaluation-criterion
...
 80 5 M(+x) L of 4th keyword            m 2    mi 34  34   .... O: evaluate-current-window
 81 6                    Ahh! Uhm!...   v 3 short
 82 6 M(-x+y) below scroll bar elev.    m 4    muc
 83 7         Which I ... should go to  v 5    v  29
 84                                                   35   ....=>G: g258 (operator no-change)
 85                                                   36   ..... P: evaluate-items-in-window
...
110                                                   61   ...... S: s486 (to-be-found variables)
111 7 M(+x-y) Left keyword D arrow      m   6  mr 62  62   ...... O: move-mouse (keyword down)
112    ---(+x) Right keyword D arrow
113    ---(-x) Right keyword D arrow
114 7 D      keyword menu scrolls D     b   7 mba 63  63   ...... O: press-button
115 8 U      stops                      b   8 mba 64  64   ...... O: release-button
116                                                   65   .... O: evaluate-current-window
...
143                                                   92   ...... S: s759 (to-be-found variables)
144 9 D       keyword menu scrolls D    b   9 mba 93  93   ...... O: press-button
145 9 U       stops                     b  10 mba 94  94   ...... O: release-button
146                                                   95   .... O: evaluate-current-window
...
```

Figure 2. Example display of a model trace aligned with data (taken from the Vars episode of Browser-Soar; Peck & John, 1992). As labeled in row 45, the left-hand columns including "T" (time of the subject's actions) through "MDC" (matched decision cycle) are one metacolumn (columns A–H), and the columns labeled "DC" and "Soar trace" on the right (I and J) are another metacolumn. The codes used in column G are explained below in Table 1. The dashed lines are added to this figure to indicate the metacolumns and do not appear on the screen.

Complex Protocol Coding: Alignment of Predicted and Actual Actions

An advanced form of sequence analysis is to use a process model (e.g., Newell & Simon, 1972) to predict the sequence of actions in behavior. The model is tested by interpreting and aligning a sequence of behaviors with respect to a model's sequential actions that serve as predictions. This can be an important component of cognitive model testing (Ritter, 2004; Ritter & Larkin, 1994). The sequences do not have to come from cognitive models such as ACT–R or Soar; a broad range of theories can be used to generate these sequences.

The Types of Segment Alignments

The types of codes for possible alignments between behavior data and model predictions are shown in Table 1. Versions of these codes are also used in Figures 1 and 2 in the column labeled MTYPE (for match type). The example alignment in Figure 1 includes all of the ways in which Dismal can display how the model's predictions are matched by the data. Alignment here thus means noting which model actions (predictions) match to the human protocol and which model actions do not match and displaying this correspondence. The alignment of model overt actions to human actions can be straightforward. In this case, identical actions are aligned (allowing for possibly different names from two different systems through the use of paired regular expressions to represent matches across the columns). Alignment of the model's mental processes to human verbal protocols requires more interpretation but is supported by an extensive theory of how verbal utterances correspond to mental states (Ericsson & Simon, 1993). This alignment can be done automatically, semiautomatically, by hand, or through a combination of these approaches. Figure 2 shows a Dismal display with the sequential data and model predictions aligned.

Formulas can then directly support many low-level analyses of the alignment, including summary statistics, such as counting the matches and their types and counting the number and types of operators matched. All of the simple model/data measures presented in Ritter (1993), such as *goodness* = *hits* − *false-alarms*, are directly supported in Dismal. Additional measures of sequence sim-

Table 1
Types of Correspondences That Are Used in Data to Model Predictions Alignments in Dismal

An *uncodable subject action*, one that cannot be interpreted with respect to the model, is shown on line 81 (as numbered on the left-hand side)—a verbal utterance that is too short to code. Waterman (1995) might describe this type of mismatch as strings taken from a different alphabet, since the datum is not intended to be brought into correspondence to a model action, and the model should not be penalized for not matching this datum.

Uncoded model actions are shown in lines 46–51, 69, and so on. They are indicated by lines of model trace without corresponding subject actions. Waterman (1995) would label this a *deletion*.

Hits are shown in lines 69, 80, 82, and so on. The match is indicated in columns E–H. Column E notes the type of the match of the segment (ST), which in this case is mouse movement inferred, or "mi." The type of correspondence (column G, labeled MTYPE) is of a matched mouse movement. The simulation cycle that is matched by the mouse movement is shown in Column H as the matched decision cycle (MDC). Waterman (1995) would label this an *identity*.

A *miss* is shown on line 82. The subject has clicked the mouse, and there is no corresponding action in the model's trace. Waterman (1995) would label this an *insertion*.

A pair of actions that are *crossed* in time is shown in lines 75 and 80. The corresponding behaviors cannot be directly aligned while keeping them in order, so the matched decision cycle column is used as a reference for the subject action matched, and there is some reason to believe there is some correspondence. Waterman (1995) calls this an *inversion*.

Note—All line numbers refer to Figure 2.

ilarities, including those reviewed in Waterman (1995), such as the number of model actions matched and the number of words in a protocol, have been or could be added as formulas or special functions.

There is one type of correspondence that is awkward to represent in Dismal: a subject action that is matched by multiple model actions. The current representation assumes that what matches a given data segment is a single action of the model. This lack of representation serves as a useful constraint; segments that match more than one model action probably are not segments. Either the model is more fine-grained than the data, or the segment should be considered for resegmenting.

Aligning segments by hand. Dismal provides two functions for aligning the metacolumns by hand. The most powerful function aligns two user-selected rows. The analyst can also insert blank rows into each metacolumn individually, which allows offsetting metacolumns while maintaining alignment within each metacolumn.

Aligning unambiguous segments automatically. Aligning predictions with data by hand is tedious and error prone, so some assistance is warranted. Complete and automatic alignment is beyond current natural language parsing technology. However, other data streams are relatively unambiguous. Dismal provides a simple alignment algorithm that can automatically align model actions with unambiguous protocol segments (Ritter, 1992; Ritter & Larkin, 1994). The automatic alignment can be supplemented and corrected with manual alignment commands. For example, mouse clicks support automatic alignment with a sequence of process models' predictions.

The algorithm used by Dismal to compute the alignment is based on Card's algorithm (Card, Moran, & Newell, 1983, appendix to chapter 5). Card's algorithm is a version of the maximum common subsequence (MCS) algorithm (also sometimes called the longest common subsequence; Hirschberg, 1975; Wagner & Fisher, 1974). In addition to computing the maximum subsequence length and the maximum matched subsequence itself, Dismal's algorithm also aligns the two metacolumns based on this MCS. The Card2 algorithm in Dismal includes three further additions to Card's original algorithm.

(1) *Matches that start at the beginning of the subject protocol and model action sequence are preferred*. There may be several possible "best" alignments (all of the same length). Because we are interested not just in how much could be aligned, but in using the alignment to understand how the alignment between the predicted sequence and the actual sequence can be improved, which possible match set gets chosen is of interest.

Most theories in this area—for example, ACT–R and Soar models—will generate the sequences, starting at the beginning, as a series of actions. Aligning from the beginning of a data sequence thus matches the theory at the beginning of performance, and when the two sequences mismatch, it is clearer where to start to improve the match by modifying the theory—typically, by extending or correcting the cognitive model's knowledge. Card2 satisfies the requirement of starting the match at the beginning of the two sequences by using the subsequence that starts closest to the beginning of the two input sequences.

For example, consider aligning the two simple strings DUC and DUDUDU. Card2 would return an edit list (referenced by position) that would call for aligning the first *D*s of each sequence together. This results in aligning the initial predictions, which is a more stable alignment if changes to the sequences can come at the rear. For example, if additional D tokens were later added to the shorter list, the alignment will change less.

(2) *The match process is based on regular expressions, not just constants*. In the original MCS task, the comparison between the two information streams is that of strict equality across identical alphabets—that is, are the tokens the same tokens across the sequences? Here, the two information streams will often contain different sets of names (alphabets) for the tokens in each streams. The comparison step in Card2 has been modified to do a more flexible comparison based on pairs of regular expressions (Friedl, 2002)—for example, the pattern "mouse-move" in the model column matches "M(*)" in the data column, where * represents "matches anything." This is a relatively small but important addition that allows sequences that use different names for the same tokens to be aligned. These differences appear to arise quite often when one works with cognitive models and sequential data. The basic functionality of matching regular expressions is provided by GNU Emacs. An example of this match pair is shown in line 111 in Figure 2. Lines 114,

115, 144, and 145 in Figure 2 show similar pairs of correspondences ("D" and "press-button," and "U" and "release-button") that represent identical elements but are presented differently in the model and the data columns.

The matching process also is a location in which to incorporate a natural language matching process. With regular expressions, the analyst can start to compare keywords from expressions with verbal utterances in a simple way. The model's predictions represent the actions and knowledge structures that one expects to find in the subject's actions and verbal utterances. Because there already exists a strong model of what will be said and done, the knowledge structures (the model's predictions) for a general parse are available, creating a restricted situation for natural language parsing. The scope of the natural language parsing problem, as well as our experience, makes it clear that a simple keyword parser will not be adequate for interpreting the data with respect to the predictions completely automatically in this restricted form. Even if the parse is only approximate, however, it will make the alignment easier to perform. Although automatic alignment was not done for the alignment in Figure 2, a correspondence between "which *" and "generate-eval-criterion" would be an example of using a regular expression to do automatic alignment. Also, codes could be applied by an analyst to the verbal utterances, and these then could be automatically aligned with the predicted sequence.

(3) *The last duplicated item in a matched subsequence is preferred*. Human behavior often has multiple actions that are matched by only one model code, as is shown in Table 2. We prefer to have the last possible item in a series of equal tokens to be used in the alignment. For example, when the Browser-Soar (Peck & John, 1992) data illustrated in Figure 2 are analyzed, the earlier Ms (mouse movements) represent mistaken or preliminary applications of the M (mouse move) operator, whereas the final movement tends to more closely represent the subject's behavior. For example, in matching "TM--CT" to "TMMMCT," there are two possible "best" sequences, as is shown in Table 2.

Table 2
Possible Best Alignments

```
TM--CT
TMMMCT
```

(a) Aligned from back in matched subsequence (lower envelope, Waterman, 1995)

```
T--MCT
TMMMCT
```

(b) Aligned from beginning in matched subsequences (upper envelope, Waterman, 1995)

The overall alignment is clearer if the subalignment in Table 2B is preferred, and the timing correspondences will be more accurate and usable as well. For example, mouse moves will be a common problem, for there may be multiple mouse moves before a click. The alignment is clearer if the user's last mouse move is aligned to the model's "move-mouse."

Example Sequential Analyses

The autoalignment facility in Dismal has been used to align human behavior protocols with model action sequences. Figure 1 shows its application to the Diag-Soar cognitive model (Ritter & Bibby, 2001). The mouse actions in the second column were automatically aligned with the mouse moves in the model actions column on the far right (labeled "Trace"). The verbal utterances were then aligned by hand. Figure 2 shows (in a slightly modified form) its first application to the Browser-Soar cognitive model (Ritter & Larkin, 1994). In these two examples, cognitive models written in Soar (Laird, Newell, & Rosenbloom, 1987) have been used to generate the predicted sequences.

The model in Figure 2 had a large number of external actions (mouse moves and clicks), which allowed the automatic alignment algorithm to do most of the final alignment. The model in Figure 1 had far fewer external actions. Although the alignment algorithm was useful for the analysis in Figure 1, the analyst performed much of the final alignment by pairing up segments by hand.

FACILITIES FOR EXTENSIONS

When the previously explained analyses prove to be inadequate or incomplete, Dismal can be extended using Emacs Lisp. Users can write their own custom commands—for example, to count the occurrences of sequences in a column. Users also can change the way cells are displayed, the way numbers are represented, or how cells are updated. Emacs Lisp is an interpreted language for which there is a primer (Chassell, 2001) and a manual (Lewis, Laliberte, Gnu Manual Group, Stallman, & Lewis, 2000), both of which are available on line. Emacs Lisp also includes a powerful compiler.

Emacs includes a library of extensions, to serve as examples, with the editor, as well as further ones on line (see, e.g., www.emacswiki.org). These extensions include terminal emulators, shell (process) control programs, debuggers, and even games. Emacs' software architecture makes it a straightforward task to tie these packages together. This permits systems using Dismal to be written in the native Emacs Lisp code or to be called as an associated UNIX process.

Since the source code is provided, Dismal can be used as a testbed for evaluating various interface designs. This has been done as part of student projects. Two students in Lisp programming classes at the University of Nottingham have provided useful extensions to Dismal as class projects, and students at the University of New Brunswick have done similar work. Dismal users have created their own packages to compute flight expenses, control machine tools, and draw graphs from the spreadsheet data.

A useful approach toward improving interface design is to incorporate known HCI theory in design tools. As a step toward this, we have created a tool incorporating several known psychological results (i.e., alias generation rules and the keystroke-level model). The tool, simple additions to Dismal, helps to create theoretically motivated aliases for command line interfaces and to compute the expected savings. This tool is similar to user interface design tools such as Glean (Kieras, Wood, Abotel, & Hornof, 1995) and CTTE (Paternò & Santoro, 2002), but implemented in a spreadsheet. We will review our tool here to illustrate how Dismal can be extended.

An Example Extension for Creating an HCI Design Tool

The ideas behind the alias creation tool could, in fact, be applied to any command set, but the command set initially optimized with this tool was Soar (Laird et al., 1987). Soar has previously been command line driven, and over 50 commands are available in Soar Version 7 (Congdon & Laird, 1995). A more complete description of this analysis is available (Nichols & Ritter, 1995).

There are a couple of reasons to create aliases for the Soar command line. Although there is a graphical interface, the command line interface remains an important part of Soar, since some members of the Soar community prefer a command line. Also, some Soar users cannot use the graphic interface, due to hardware limitations, or prefer not to because of program speed and size concerns. Soar was (and is) used within our local environment. It is worthwhile enhancing the system for both local users and the Soar community worldwide. There is also the potential to get command use frequency information and usability feedback from the local users. The most important reason is that providing aliases will improve the usability of Soar. Some of the commands that exist within Soar are quite long; this suggests that users could benefit greatly from the introduction of aliases.

The HCI theories were implemented by creating two Emacs Lisp functions that are now part of Dismal. The first function (make-alias) used the rules noted below to automatically generate command aliases. The second (key-val) took a command and calculated the number of mental operators and keystrokes used in the execution of the command. It then used the estimated typing speed to calculate a time prediction.

The Alias Generation Function

An aliases-generating function was created incorporating available alias generation guidelines (Ehrenreich & Porcu, 1982; John & Newell, 1987; Payne & Green, 1986; Watts, 1984), primarily using truncation and minimum-to-distinguish. These guidelines have been shown to be the most efficient form of abbreviation (John & Newell, 1987), partly due to the ability of having consistency in the resulting command set (Ehrenreich & Porcu, 1982).

The characteristics of the command language itself were also considered—particularly, that many of the commands consisted of several words. Therefore, we added the following additional guidelines in order to create a consistent alias set: (1) Include in the alias the first letter of each word in the case of multiword commands, and (2) if the length of the command is five letters or less, a one-letter alias may be provided if clashes do not occur as a result of the new abbreviation. In general, this means that already short, already abbreviated commands (e.g., "pwd") do not get shortened to one letter but short one-word commands (e.g., "watch") do get abbreviated to a single letter.

The alias generation function takes as an argument another cell, and it returns an alias. Clashes are kept on a global list that can be inspected. The automatic function generated 80% of the final aliases for the Soar command set, where the rules could be strictly adhered to.

The Time Predictor Functions

Functions are provided to predict the time to execute the aliases and the original command set according to the keystroke-level model (Card et al., 1983). These functions could be further extended to apply to other input modalities, such as speech.

The keystroke-level model (Card et al., 1983) was used as a measure of design efficiency, since it predicts the time taken for a set of commands or aliases to be executed on the basis of subtask speeds and frequency of tasks. It is a useful and practical simplification of GOMS (goals, operators, methods, and selection) analysis at the level of individual keystrokes, when the user's interaction sequence can be specified in detail, as it could be in this case. The keystroke-level model has been shown to be useful as an engineering design model (e.g., Card et al., 1983; Haunold & Kuhn, 1994).

Our current model can be adjusted for typing speed. The inclusion of mental operators is governed by heuristics specified by Card et al. (1983)—specifically, Rule 2 in their Figure 8.2. An important question to consider is how the aliases might impose a burden on memory and mental operator time. The keystroke-level model makes several suggestions about how the aliases should be generated consonant with the generation guidelines noted here. If a rule set is used to generate the aliases, once the initial (very small) rule set has been learned, only a single mental operator is required to enter a command alias, and that behavior can become automated more quickly—rather than having to learn by rote each alias, a rule can be applied (Card et al., 1983).

Estimations of Task Frequency

The keystroke-level model uses task frequency to compute the total time it takes to do a set of tasks. Command frequency data, necessary for computing time savings and useful for arbitrating alias clashes, can be difficult to obtain. An initial analysis of approximately 2 h of naturalistic subject data, provided by Altmann, was performed to compute these frequencies. Perusal of additional transcripts suggested that command usage is highly dependent on the task. There were large individual differences. In order to generate meaningful frequency data, enormous amounts of keystroke logs would be required (we estimate that this would be on the order of hundreds of hours, on the basis of the pilot data).

In an attempt to generate useful frequencies more quickly, four expert Soar users, all with more than 3 years experience working with Soar, were asked to provide frequency estimates for their own use of the original command set. These were easy to provide, although they correlate only modestly well (mean pairwise correlation of .49).

Testing the Predictions Using Dismal

Figure 3 shows the estimated time savings based on the time to perform the original command set and the alias set, balanced for each expert's estimated frequency distribution. The maximum height of a bar was 100%, which would be the time to enter the original commands based on the expert's estimate of the usage of each command in the command set. When the keystroke values of the commands and aliases were calculated and weighted using a flat uniform command distribution (i.e., assuming that all commands were used equally often), it was found that the aliases provided a 55% saving in time over the original command set. As is shown in Figure 3, the estimated time savings for the alias set ranged between 38% and 53% across the experts' distributions.

This approach to generating aliases could be implemented within any programming language, but we prefer a spreadsheet to perform this analysis. This visual presentation and the use of functions to compute and display the expected times make the process easy to follow and provide updates automatically to the designer. We found that expert users can quickly provide useful and reasonably consistent estimates and that the time savings predictions were robust across their predictions and when compared with a flat command frequency distribution.

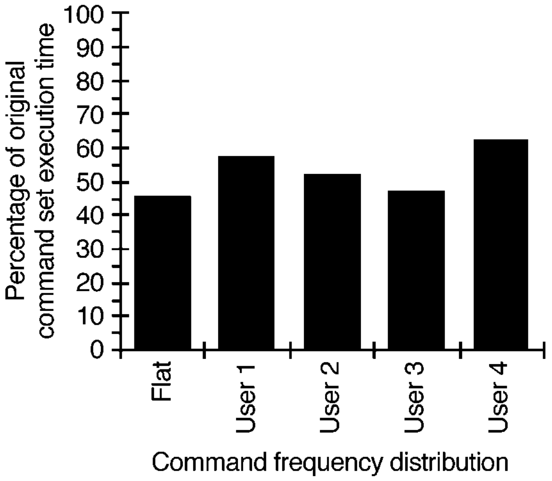


Figure 3. Comparison of predicted benefits of generated aliases for different command frequency distributions. The original command set is represented by 100%.

The Effect of Aliases on Time and Errors

No user types perfectly, and decreasing command length may also decrease user errors. The simplest prediction is that simple typing errors are related to the number of keystrokes, no matter what the user's typing accuracy or speed. For an example set of commands novices would use to learn Soar, this alias set reduced the number of keystrokes by 68%.

With the keystroke logger in Dismal, we recorded the keystrokes of users. In an unpublished study of 20 subjects, we found that even while learning the Soar command line interface through use, novices employing this alias set had a 62% reduction in typing time and a 48% reduction in errors, as compared with novices employing the original command set (Ritter & Bishop, 1997).

A further analysis with this approach suggests that automatic command completion, although more useful than no support, would be inferior to this alias set. Command completion would not reduce the keystrokes in this set of commands as much as the aliases do and would not decrease the mental operators at all.

Overall, this tool appears to be a robust and inexpensive way of applying simple HCI theories to design to reduce command execution time. Aliases can be constructed and tested very easily, and, with the use of an HCI tool, the principles behind the forms of command aliases can be applied routinely and uniformly. Savings estimates can also be documented directly and used to guide design when it is necessary or desirable. The use of a system in which guidelines are employed to generate aliases means that the alias forms are easy to learn and can generally be predicted. The local Soar user group has found the alias set to be generally useful, and the aliases have also been distributed to the Soar community at large. Aliases improve the interface at quite modest cost. There appears to be no reason not to take this efficiency gain.

FACILITIES FOR HCI EXPERIMENTATION

Dismal supports experimentation on itself. Dismal is instrumented; it is possible to automatically record each user action and the time it occurred. We have found that undergraduates are able to use Dismal in a 5-week practical module to gather realistic user data and test HCI theories (e.g., the keystroke model of Card et al., 1983). Installation of this part of Dismal is more difficult and requires compiling a C program. The UNIX installation process does this automatically, but this is not generally available on the PC or Mac versions but should be possible in the future.

The data representation of Dismal's logging feature, shown in Table 3, is similar to other logging programs (e.g., Westerman et al., 1996). The data columns include a timestamp in seconds and milliseconds, the keystroke, and the keystroke command binding. Analysis tools are included to find and aggregate the commands between carriage returns. The open architecture of Emacs would allow these logs to be played back, but we have not created such a facility.

The resulting data can be used several ways. It can be simply aggregated to compute time on task and to note what commands are executed. This level analysis is related to exploratory sequential data analysis (Hilbert & Redmiles, 2000; Sanderson & Fisher, 1994). The resulting data could also be compared and aligned with a model's predictions or with other users' behavior when the alignment tools included in Dismal are employed (Ritter & Larkin, 1994).

This facility allows time data to be gathered on other interfaces that can be run within an Emacs shell. This facility has been employed to gather data on using Emacs for programming (Altmann & John, 1999).

Dismal is also useful for exploring HCI design, particularly as student projects. Since the source code is provided, Dismal can be used itself as a testbed for evaluating various interface designs, and comparisons can be based on actual user data. Two students in Lisp programming classes at the University of Nottingham have provided useful extensions to Dismal's interface as class projects.

AVAILABILITY

Dismal and its source code are available at no cost. It is copylefted,[3] which basically means if you pass Dismal on, you must include its source code. Dismal is available through a variety of sources. The most convenient is via FTP. Its most recent version is available through www.gnu.org/software/dismal/dismal.html. It is also available by posting one 3.5-in. high-density diskette or a blank CD and your address to Frank Ritter, School of Information Sciences and Technology, Penn State University, University Park, PA 16802. If you would like to be put on the Dismal users' e-mail list, please send an e-mail to frank.ritter@psu.edu.

Table 3
Example Keystroke Logs Generated by the Timing Package

```
User: s12
Creation-date: Mon Feb 10 13:20:52 1997
System: upsyc.psyc.nott.ac.uk
Start-buffer: *scratch*
80855.017    ^M    minibuffer-complete-and-exit
80878.006    S     self-insert-command
80879.615    o     self-insert-command
80891.763    a     self-insert-command
80892.497    t     self-insert-command
80892.646    ^?    delete-backward-chars
80906.405    r     self-insert-command
80962.430    ^M    comint-send-input
80965.701    i     self-insert-command
80965.800    n     self-insert-command
...
```

A README file accompanies the tar, gzipped file, providing installation instructions. The individual installing Dismal does not have to know GNU Emacs or Emacs Lisp but should be somewhat familiar with UNIX and the local file system. Macintosh or Windows users will need an untar application. Users already familiar with Emacs will be the most productive. Previous versions of Dismal have been ported to work with XEmacs, an offshoot of GNU Emacs. However, it is unlikely to work with variants of Emacs other or older than Version 19.

COMPARISON WITH SIMILAR TOOLS

As compared with Excel or other commercial tools, Dismal includes some new commands not supported in other spreadsheets, such as automatic alignment of two sequences, the data-coding commands, and tools for removing blank rows automatically. But the major difference between Dismal and similar tools is that Dismal's source code is in the GNU-Emacs distribution, making Dismal available and extendable. Others have found developing macros and full extensions to Dismal straightforward, and it has been integrated with software for developing and testing cognitive models (Ritter & Larkin, 1994).

Some of Dismal's initial design and iconography for sequence alignment comes from Trace&Transcribe (John, 1994) and MacShapa (Sanderson, McNeese, & Zaff, 1994; Sanderson, Scott, et al., 1994). Dismal goes significantly further than Trace&Transcribe, a tool for aligning protocols, in its representations and in the tools it provides for interpreting and aligning the transcribed protocol data with respect to the predictions. Dismal does not include as many exploratory sequential data analysis tools as MacShapa, although its alignment algorithm has been added to the MacShapa analysis program.

Dismal includes a logging package that can time users, including users of other Emacs-based tools. This is not usually provided with similar software. In most other ways, however, Dismal is a relatively modest spreadsheet. It is slower, less robust, and lacks some of the built-in features of other spreadsheets.

SUMMARY

We have described a spreadsheet that is useful in several ways for gathering and working with a variety of types of behavioral data. Dismal can gather keystroke logs of users. These logs or other measures can be manipulated within a spreadsheet. Of particular note are the capabilities for coding and aligning sequences. This has particularly been used for aligning model actions with data, but it works with any two sequences. When a complete description of correspondences can be provided, such as keystrokes by the subject and the model, the two sequences can be automatically aligned. When the comparison is less clear, such as between a subject's verbal utterances and representations in the model, semiautomatic commands allow an analyst to align items. Together, these commands substantially reduce the work of testing cognitive models with protocols by up to a factor of five (Ritter & Larkin, 1994), allowing such analyses to be performed more often and with more insight.

These analyses can be extended using Emacs Lisp and the Dismal source code. Further experiments can be created, or the tool extended more globally. Users have done this by adding menus and speed optimizations. These changes could be studied using users' keystroke logs, closing the loop. Examples of these capabilities have been provided. A set of functions for creating aliases and predicting their effect with the keystroke-level model (Card et al., 1983) was included as an example. The example predictions were tested by running subjects with the timing package available in Dismal, closing the loop of theory creation and theory testing.

In the future, Dismal will need to be improved further. Missing features include rather general features, such as the ability to split a buffer in two, better math functions, and the ability to manipulate cells directly with the mouse.

Dismal has continued to develop over the last 10 years because it serves several real needs in research, teaching, and administration. Further users and uses will strengthen it.

REFERENCES

Altmann, E. M., & John, B. E. (1999). Episodic indexing: A model of memory for attention events. *Cognitive Science*, **23**, 117-156.

Byrne, M. D. (2001). ACT–R/PM and menu selection: Applying a cognitive architecture to HCI. *International Journal of Human–Computer Studies*, **55**, 41-84.

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

Chassell, R. J. (2001). An *introduction to programming in Emacs Lisp* (2nd ed.). Cambridge, MA: Free Software Foundation. Retrieved May 2, 2004, from www.gnu.org/software/emacs/emacs-lisp-intro/.

Clarke, D. D., & Crossland, J. (1985). *Action systems: An introduction to the analysis of complex behaviour*. London: Methuen.

Congdon, C. B., & Laird, J. E. (1995). *The Soar user's manual, Version 7*. Ann Arbor: University of Michigan, Electrical Engineering and Computer Science Department.

Ehrenreich, S. L., & Porcu, T. (1982). Abbreviations for automated systems: Teaching operators the rules. In A. Badre & B. Shneiderman (Eds.), *Directions in human/computer interaction* (pp. 111-136). Norwood, NJ: Ablex.

Fielding, N. G., & Lee, R. M. (Eds.) (1991). *Using computers in qualitative research*. London: Sage.

Finlay, J., & Harrison, M. (1990). Pattern recognition and interaction models. In D. G. D. Diaper, G. Cockton, & B. Shakel (Eds.), *Human–computer interaction–INTERACT'90* (pp. 149-154). Amsterdam: Elsevier.

Free Software Foundation (1988). *GNU Emacs*. Cambridge, MA: Author.

Friedl, J. E. F. (2002). *Mastering regular expressions*. Sebastopol, CA: O'Reilly.

Gottman, J. M., & Roy, A. K. (1990). *Sequential analysis: A guide for behavioral researchers*. Cambridge: Cambridge University Press.

Haunold, P., & Kuhn, W. (1994). A keystroke level analysis of a graphics application: Manual map digitizing. In B. Adelson, S. Dumais, & S. Olson (Eds.), *Proceedings of the CHI'94 Conference on Human Factors in Computer Systems* (pp. 337-343). Boston: ACM.

Hilbert, D. M., & Redmiles, D. F. (2000). Extracting usability information from user interface events. *ACM Computing Surveys*, **32**, 384-421.

Hirschberg, D. S. (1975). A linear space algorithm for computing maximal common subsequences. *CACM*, **18**, 341-343.

John, B. E. (1994). *A database for analyzing "think-aloud" protocols and their associated cognitive models* (No. CMU-HCII-94-101). Pittsburgh: Carnegie-Mellon University Human–Computer Interaction Institute.

John, B. E., & Newell, A. (1987). Predicting the time to recall computer command abbreviations. In J. M. Carroll & P. P. Tanner (Eds.), *Proceedings of the CHI'87 Conference on Human Factors in Computer Systems* (pp. 33-40). New York: ACM.

Kieras, D. E., Wood, S. D., Abotel, K., & Hornof, A. (1995). GLEAN: A computer-based tool for rapid GOMS model usability evaluation of user interface designs. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'95)* (pp. 91-100). New York: ACM.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, **33**, 1-64.

Lewis, B., LaLiberte, D., Gnu Manual Group, Stallman, R. M., & Lewis, B. (2000). *GNU Emacs Lisp reference manual: For EMACS Version 21*. Cambridge, MA: Free Software Foundation. Retrieved May 2, 2004, from www.gnu.org/software/emacs/elisp-manual/.

MacWhinney, B. (1991). *The CHILDES project: Tools for analyzing talk*. Hillsdale, NJ: Erlbaum.

MacWhinney, B., & Snow, C. (1990). The Child Language Data Exchange System: An update. *Journal of Child Language*, **17**, 457-472.

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

Nichols, S., & Ritter, F. E. (1995). A theoretically motivated tool for automatically generating command aliases. In *Proceedings of the CHI'95 Conference on Human Factors in Computer Systems* (pp. 393-400). New York: ACM.

Paternò, F., & Santoro, C. (2002). One model, many interfaces. In C. Kolski & J. Vanderdonckt (Eds.), *Computer-aided design of user interfaces III: Proceedings of the 4th International Conference on Computer-Aided Design of User Interfaces CADUI'2002* (pp. 143-154). Dordrecht: Kluwer.

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

Peck, V. A., & John, B. E. (1992). Browser-Soar: A computational model of a highly interactive task. In P. Bauersfeld, J. Bennett, & G. Lynch (Eds.), *Proceedings of the CHI '92 Conference on Human Factors in Computing Systems* (pp. 165-172). New York: ACM.

Ritter, F. E. (1993). *TBPA: A methodology and software environment for testing process models' sequential predictions with protocols* (Tech. Rep. CMU-CS-93-101). Pittsburgh: Carnegie-Mellon University, School of Computer Science. Available at reports-archive.adm.cs.cmu.edu/anon/1994/CMU-CS-94-102.ps.

Ritter, F. E. (in press). Choosing and getting started with a cognitive architecture to test and use human–machine interfaces. *MMI-Interaktiv-Journal*, **7**, 17-37.

Ritter, F. E., & Bibby, P. (2001). Modeling how and when learning happens in a simple fault-finding task. In *Proceedings of ICCM 2001 Fourth International Conference on Cognitive Modeling* (pp. 187-192). Mahwah, NJ: Erlbaum.

Ritter, F. E., & Bishop, M. (1997). *The effect of abbreviation on time and errors: Even novices can profit from aliases*. Unpublished manuscript.

Ritter, F. E., & Larkin, J. H. (1994). Using process models to summarize sequences of human actions. *Human–Computer Interaction*, **9**, 345-383.

Ritter, F. E., & Ong, R. (1994). *The simple-menu package, Release 1.2*. Now part of Dismal.

Sanderson, P. M., & Fisher, C. A. (1994). Exploratory sequential data analysis: Foundations. *Human–Computer Interaction*, **9**, 251-317.

Sanderson, P. M., McNeese, M. D., & Zaff, B. S. (1994). Handling complex real-world data with two cognitive engineering tools: CO-

GENT and MacSHAPA. *Behavior Research Methods, Instruments, & Computers*, **26**, 117-124.

Sanderson, P. M., Scott, J., Johnston, T., Mainzer, J., Watanabe, L., & James, J. (1994). MacSHAPA and the enterprise of exploratory sequential data analysis (ESDA). *International Journal of Human–Computer Studies*, **41**, 633-681.

Sankoff, D., & Kruskal, J. B. (Eds.) (1983). *Time warps, string edits, and macromolecules: The theory and practice of sequence comparison*. Reading, MA: Addison-Wesley.

Wagner, R. A., & Fisher, M. J. (1974). The string-to-string correction problem. *Journal of the Association for Computing Machinery*, **21**, 168-172.

Waterman, M. S. (1995). *Introduction to computational biology: Maps, sequences and genomes*. London: Chapman & Hall.

Watts, R. A. (1984). *Introducing interactive computing*. Manchester: NCC Publications.

Westerman, S. J., Hambly, S., Alder, C., Wyatt-Millington, C. W., Shrayane, N. M., Crawshaw, C. M., & Hockey, G. R. J. (1996). Investigating the human–computer interface using the Datalogger. *Behavior Research Methods, Instruments, & Computers*, **28**, 603-606.

NOTES

1. This gives rise to its name, DIS Mode Ain't Lotus, coined by David Fox, its initial developer.

2. The model's actions are often referred to as a *trace*. This is in contrast to the use of *trace* as an alternative way of representing an alignment of two sequences in Sankoff and Kruskal (1983).

3. For more information, see www.gnu.org/copyleft/copyleft.html.

(Manuscript received December 27, 2001;
revision accepted for publication April 25, 2004.)