Ritter, F. E., & Larkin, J. H. (1994). Developing process models as summaries of HCI action sequences. Human-Computer Interaction, 9(3&4), 345-383.

HUMAN–COMPUTER INTERACTION, 1994, Volume 9, pp. 345–383

Developing Process Models as Summaries of HCI Action Sequences

Frank E. Ritter
University of Nottingham

Jill H. Larkin
Carnegie Mellon University

ABSTRACT

We describe the utility of *process models* for summarizing the sequential actions of individuals. Such models describe why users did what they did, what information they used from the outside environment, and what knowledge they used to perform the task. These detailed explanations of users' thoughts and actions can enhance interface design by offering behavior summaries that are inspectable and transferable to new interfaces. Sequential data sets and models for human–computer interaction are often large and complex. We present a computer-supported methodology for developing these models as summaries of sequential data. We illustrate that this methodology can make building and using such models tractable by applying it to an existing model for using an on-line database.

Frank E. Ritter is a psychologist and computer scientist interested in cognitive modeling; he lectures in artificial intelligence and cognitive psychology and is associated with the Economic and Social Research Council's Centre for Research in Development, Instruction, and Training at the Department of Psychology, University of Nottingham. Jill H. Larkin is a psychologist, educator, and computer scientist interested in applications of psychology and psychological models to teaching reasoning; she is an Associate Professor of Psychology and a Senior Research Scientist at the Center for the Development of Educational Computing at Carnegie Mellon University.

CONTENTS

1. INTRODUCTION

Understanding human–computer interaction (HCI) often involves describing temporal sequences of interactions and understanding why the sequencess occurred. Such activity falls under the heading of *exploratory sequential data analysis* (*ESDA;* Sanderson & Fisher, 1994). A particularly rigorous analysis of such sequences is a *process model*–an information-processing algorithm that performs a given task with observable behavior similar in sequence and form to that of humans performing the same task. Thus, an HCI process model is an algorithm able to process interface events and generate interface commands so that the external behavior of the model is similar in content to that of a human user.

As a tool for understanding human behavior, process models have the following virtues:

1. Unlike verbally described models, process models are explicit. They make definite predictions about users' sequences of actions, about performance time (e.g., Card, Moran, & A. Newell, 1983), and about the nature and frequency of errors (e.g., Young & Whittington, 1990). Ideally, this explicitness clarifies the content of a model so that discourse focuses on strengths and weaknesses of alternative models in accounting for and predicting observations.
2. The computational mechanisms required to achieve the observed behaviors have led in several domains to understanding the mechanisms of human reasoning (e.g., A. Newell & H. A. Simon, 1972).
3. Without losing explicitness, process models can capture as much detail and variability as are present in the data. Because they provide a language for representing exceptions and detail, process models are an ideal vehicle for rich tasks–such as those in HCI–that produce varied and complex behavior.

Within HCI, process models can both guide or test designs and elucidate psychological mechanisms underlying humans' interactions with computers. (Olson & Olson, 1990, provided a summary.) Polson and Lewis (1990), for example, used process models to formulate general design guidelines. Because process models perform the task of interest, they make direct predictions about what a human user will do with an interface and can become a readily available "tester" of an interface or interface design (Howes & Young, 1991). This level of analysis and understanding is worthwhile for interfaces when the payoff is high–for example, when the tasks are crucial, when there are many users, when the users' time is valuable, or when the interface is expensive to produce or evaluate (e.g., Gray, John, & Atwood, 1993). Last, process models provide behavior explanations, involving psychological processes relevant to HCI, and so advance the basic science of the field.

However, building an information-processing algorithm and then matching its actions step by step against sequential data is a daunting task. Our goal here is to specify a methodology for this process and to show how it can become tractable, in particular, by means of good computer tools. In Section 2, therefore, we propose a methodology for developing and using process models to summarize sequential data, particularly for HCI tasks. In outline form, this methodology, *trace-based protocol analysis* (*TBPA*), consists of (a) using an initial model's sequential predictions (a *trace*) to find structure in the sequential data, (b) summarizing the model's successes and failures in accounting for the data, and (c) using the results of this summary to improve the model in an iterative cycle. In Section 3, we discuss the potential utility and difficulties of this approach.

The use of process models in HCI, as well as elsewhere, would be more tractable if computational aids could perform or facilitate these tasks. In

Figure 1. **Trace-based protocol analysis (TBPA) methodology for developing a process model accounting for sequential data.**

Task 0	Develop a preliminary process model, an environment for it to manipulate, and an initial set of transcribed sequential data to test it.
Task 1	Run the model to produce a trace of its actions. The trace should support alignment with protocol data by including: a. A sequence of predictions (including times) for each of the data streams (e.g., verbal, mouse and keyboard actions) and any responses from the task environment. b. Symbols clear to human analysts and unambiguous to automatic tools.
Task 2	Interpret the data and align them with the actions in the model trace. a. Create a sequence of matched data–trace pairs with codes for the type of match and associated annotations. b. When necessary, minimally shift data and model sequences relative to each other.
Task 3	Assess how well and in what respects the model accounts for the data, using summaries that: a. Show how the data correspond to the mechanisms of the model. b. Show where the data do and do not match the trace of the model.
Task 4	Revise the model to improve its account of the data based on understanding the mechanisms and dynamic behavior of the model.

Section 4, we review computational tools that support parts of TBPA, and we summarize their useful properties. With these properties in mind, in Section 5 we present a prototype environment, Soar/Model-Testing (SMT), that gives integrated support to TBPA. We illustrate TBPA as supported by SMT by applying it to an existing model and its data, illustrating how a good support environment can make developing process models more tractable and enlightening.

2. TBPA: A METHODOLOGY FOR DEVELOPING PROCESS MODELS

Figure 1 summarizes the tasks of TBPA–a general methodology for using process models to characterize sequential data. For complex tasks such as HCI, this methodology is usually tractable only with good computational tools. Therefore, we specify the tasks in detail–a necessary precursor for supporting them computationally.

To illustrate TBPA, we discuss a model ("Browser") and data for browsing in an on-line help system. Peck and John's (1992) paper is the primary reference for their work. Their data came from a subject browsing through an unfamiliar on-line help database. Browser is a typical process model in that it is computer implemented and makes predictions about sequential actions of a user working with an HCI interface.

2.1. Task 0: Provide a Preliminary Model and Protocol Data Set

Task 0 is outside the model-testing loop and is not repeated, but it gets the methodology started. Here we discuss appropriate models and data with which to begin the TBPA process proper.

Properties of Process Models. Process models consist of coordinated action specifications sufficient to produce a sequence of actions that perform a task. To start Task 1, the model trace need only correspond roughly with actions in the protocols. This restricts TBPA to tasks for which psychology and our model-building capabilities are powerful enough to build this initial model. A. Newell (1977) and Ohlsson (1990) provided approaches for creating this initial model. Process models are typically, but not always, organized as a rule set (e.g., production systems). Coordinating the action specifications so that each action occurs appropriately usually involves (a) associating with each action the conditions when it is useful and possible and (b) providing a task-relevant hierarchy of goals, so that complex tasks can be divided into subtasks. These models are complex enough that producing accurate action sequences by hand is difficult; thus, they are typically formulated as computer programs (A. Newell, 1977). If the task involves interaction with the external world (e.g., a computer interface), then there should be mechanisms that provide environmental inputs to the model appropriately (e.g., in response to actions taken by the model).

Properties of Protocol Data. Protocols are sequential data collected while a subject performs a task. Protocols may include multiple streams of data–for example, verbal utterances, motor actions, environment responses, and eye movements (A. Newell & H. A. Simon, 1972). Recording the responses of the environment is often necessary for understanding the behavior as well. Multiple data streams constrain the model by requiring it to account for more aspects of behavior. The Browser data, for example, include verbal utterances, mouse movements and clicks, and events in the browsing interface.

Performing a task *requires* certain actions–using a computer requires mouse actions or keystrokes, and driving requires steering and braking. We call these *task actions*. As the model performs the task, it too executes task actions, and there is a direct match between the task actions of the subject and those of the model. J. B. Smith, D. K. Smith, and Kupstas (1993) described task-action protocols and how to collect them. In contrast, nontask actions (e.g., verbalization, gestures) are not necessary for

achieving the task. Because a subject can omit them, their interpretation is more complicated. They can, however, suggest internal reasoning processes behind the task actions. Verbal "think aloud" protocols (recorded as a subject thinks aloud while performing the task) are particularly useful for this reason. According to theory developed by Ericsson and H. A. Simon (1993), this verbalization stream reflects the subject's current working memory and thus should correspond to the internal state of the running process model.

Some data streams include identifiable actions (e.g., clicking a mouse, looking at a word). Others, especially verbal utterances, must be segmented into "actions" to allow comparison with the discrete actions of a model. For verbal protocols, an action usually consists of an episode of speech, unbroken by appreciable pauses, conveying one simple idea.

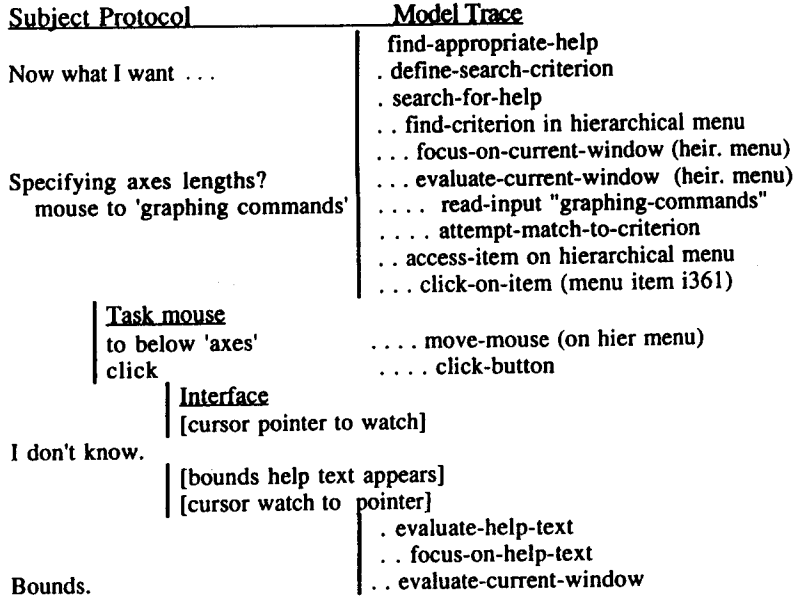
2.2. Task 1: Produce a Trace

Running the model to produce a trace of its actions while performing the task is the first task of the TBPA loop. This trace forms the model's predictions for all the sequential data streams. If verbal utterances are included, then the trace must include corresponding features of the internal memory of the model. If the subject manipulates an active environment, the trace should show changes in the simulated environment (e.g., a Macintosh window open following a mouse double-click). If timing is predicted by the model, then times must appear as well. This trace should be well formatted for human interpretation and sufficiently standardized for use by computational alignment tools.

2.3. Task 2: Interpret and Align the Data and Model Trace

The trace of a process model forms the model's predictions about the sequential actions in the data. Task 2, then, is a process of matching observation sequences to predictions. For these matches to be valid or useful, they must be replicable and self-consistent. Matching two elements may thus require information not only about those elements but also about temporally adjacent portions and (to maintain consistency) other matches of a similar type. Figure 2 shows a protocol fragment aligned with an edited trace produced by the Browser model. Both columns include task actions (mouse clicks and movements) and mental (nontask) actions (e.g., a decision to search for the length axes command, second line in the subject protocol). An external file, prepared by hand, provided interface information to the model, and the trace here omits this information. The complete aligned Browser trace and

Figure 2. **Simplified sample of the match between model and data.**



data include 5,554 lines of the form shown here, roughly the size of a small-town phone book.

Corresponding elements of trace and model are aligned by inserting blank elements in either sequence to achieve the best match throughout the dual sequence (see Figure 2). The alignment task may be tedious but conceptually trivial (e.g., aligning an unambiguous set of button clicks to a well-fitting model), or it may be extremely demanding (e.g., finding indefinite references within verbal utterances). Task actions (e.g., mouse clicks, moving a disk) are usually easiest to match—an observable action from the data matches the same action described in the trace. Elements corresponding in content may be out of order, or several elements in one sequence may correspond to one element in the other sequence.

According to Ericsson and H. A. Simon (1993), verbal reports reflect the content of working memory, or changes in this memory. However, in a well-defined model, each working-memory change is produced by an action, and utterances can be equivalently matched to these model actions (A. Newell & H. A. Simon, 1972, p. 157). This is the approach Peck and John (1992) used, and we follow it here. In Figure 2, for example, the "define-search-criterion [of label axes]" model action matches the "Now what I want ... [label axes]" utterance.

2.4. Tasks 3 and 4: Assess the Prediction–Data Correspondence and Revise the Model

The result of alignment is a large volume of associated model-trace elements. Primitive measures (e.g., fraction of elements matched) may give some idea of the adequacy of the model and illustrate whether it is worth taking seriously. But, to guide improvement of a preliminary model requires richer ways of summarizing and assessing in what respects the model does and does not account for the data. Perhaps the most useful approach is summarizing the match of the predictions and data in terms of the central structures in the model. If entire structures account well or badly for data, this information can either guide revision or suggest differences among subject groups. The pattern of support can indicate individual differences in theoretical terms of established models (Miwa & H. A. Simon, 1993; Ohlsson, 1990). Later, we discuss several computer-implemented graphical analyses intended to serve these functions.

Revising the model is a design problem. This makes it the least specified task in TBPA–which can make it the most difficult as well. In our discussion of computational tools for TBPA, we illustrate how revisions are suggested by tools for understanding the model and its relation to data. Over time, however, because models cannot ever be proved, the useful evaluation measures indicate where the model succeeds and fails so it can be improved (Grant, 1962; J. B. Smith et al., 1993). Ritter (1993) reviewed statistics in this area and started to formalize a method of model evaluation based on assessment.

3. UTILITY AND DIFFICULTIES OF PROCESS MODELS IN HCI

In this section, we illustrate the potential utility of process models in HCI, using the Browser work as an illustration. We also discuss difficulties and how the work described in this article may help to alleviate these difficulties.

The Browser model characterizes data from a novice subject using a help system (unfamiliar to the subject) on the Macintosh (familiar to the subject). Initially, Peck and John (1992) expected this behavior to consist largely of deliberation and search as the user coped with the unfamiliar situation. In fact, Peck and John accounted for 5,554 lines of protocol data with the Browser model, which executes only what they called "routine" behavior–straightforward execution of methods without search.

Most important, the model indicates the mechanisms by which the novice used the unfamiliar database so smoothly–mechanisms of "routine" behavior (Peck & John). Each task in Browser is implemented through an already known sequence of subtasks. Subtasks, in turn, may be

implemented by their own methods, forming a task–method hierarchy. However, there is no trial-and-error, no generate-and-test, no internal, mental search for a solution that characterizes nonroutine problem solving. Browser uses general search methods at the top of its hierarchy. If words like *page* and *location* were substituted for words like *window* and *menu,* the task hierarchy could reflect search procedure for anything from a dictionary definition to a mislaid mop. The bottom part of the task hierarchy consists of the methods for using the Macintosh interface, also familiar to the user. Thus, the model predicts that interface designs are likely to be usable by novices if the needed knowledge consists only of general procedures and procedures for a familiar interface.

Browser also provided a more typical contribution from HCI studies: a measure of consistency in the interface (e.g., Bovair, Kieras, & Polson, 1990). It identified one instance in which the help system violated the Macintosh interface conventions and one instance in which it did not provide an access facility consistently across the interface (B. E. John, personal communication, 1993).

Could these conclusions be reached without developing a process model and laboriously comparing the sequential predictions of the model with sequential data? To a certain extent, they could. These conclusions are not so odd that they were exclusively discovered through a detailed comparison with the data. However, this level of comparison is useful for two reasons. First, as with many process models, the main features of the Browser model were not obvious from the beginning. Peck and John (1992) reasonably expected that a novice user would engage in a great deal of nonroutine search for methods to use the database. It was only through detailed comparison of the user's actions with those of a routine-behavior model that Peck and John came to believe that most of the user's behavior was also routine.

Second, all of Peck and John's (1992) conclusions are supported by large amounts of detailed data. A large segment of a novice's behavior is modeled using *only* general and interface knowledge. The only exceptions indicated where the interface violated the Macintosh conventions. These results would be less believable without this level of comparison.

Difficulties. Studies involving process models use large amounts of complex, ordered information from the model traces and from the protocols. Unanalyzed, the raw information provides little insight. Comparing long sequences of human actions with the predictions is a complex, symbolic manipulation, which has often required doing the task by hand. The work is tedious, and the size of the data sets makes the manipulations prone to error. Furthermore, as multiple versions of a model are compared with multiple protocols, the amount of information to be managed grows quickly. A major problem is simply finding the information when it is

needed. A minute of protocol data can ultimately require as much as 5,000 min of analysis and model-building time (Ohlsson, 1982; Ohlsson, personal communication, 1992; Peck & John, 1992; Peck & John, personal communication, 1992). Analyzing these amounts of symbolic data is an overwhelming task when done by hand.

A second difficulty in using process models is more subtle but just as pervasive. The goal is to explain, using the dynamic mechanisms of the model, the nature of behavior. It is not trivial to understand why the model behaves as it does and to use its structure as a guide throughout analysis. Drawing inferences from a process model requires intimate familiarity with how it works because the dynamic structures are not directly inspectable. Although the model specifications may be readily available, their interactions and the structures that arise as the model executes are not. For example, to determine what fraction of Browser's information processing involves the Macintosh interface would require running the model and examining its behavior, including its internal state, at every action.

Limited Use. Unfortunately, testing the sequential predictions of process models with verbal protocol data is done rarely, by few individuals, and with small amounts of data. We believe that providing computational support can make such work more tractable, so that the utility we have suggested can be realized more often.

Figure 3 lists 22 such studies, which we were able to compile through a year of computer searches, inquiries, previous reviews (Kaplan, 1987), and discussions with people active in the field. In contrast, Bentler (1980) estimated that there were more than 10,000 papers using multivariate analysis. Besides being few, these studies are small in scale compared with other forms of analysis. Eleven of the 22 studies involve just one subject. The total protocol time analyzed ranges from 220 sec (less than 4 min) to 10,000 sec (2.7 hr). Only 4 studies involve more than 1,000 segments. The most complex tasks (e.g., geometry, physics, medical diagnosis) may typically become tractable by using fewer, longer duration segments. Although an average segment length is probably 1 to 4 sec, the average segment time for these studies, when reported, is much longer than a single verbal utterance. Last, only about half the studies using verbal protocols also use task-action data–that is, actions done in the real world in service of the task. Perhaps these restrictions are inevitable given that all this work was done by hand.

As shown in Figure 3, Browser is one of the larger efforts in model size and amount of data. Its size and complexity make the Browser study a good test-case for formalizing TBPA and developing a supporting environment. However, compared with what is needed for HCI, Browser is relatively simple in two ways:

Figure 3. **Research reports using verbal protocol data to test sequential predictions of process models.**

		Data					Model	
Author(s)	Task	N[a]	ΣT[b]	#S[c]	ST[d]	R[e]	M[f]	#S/M[cf]
VanLehn (1989), Experiment 1	River crossing	1		9			2	4.5
Kieras (1982)[g]	Reading	10	36,000	42	85		20[i]	21
Ohlsson (1990)	Linear syllogisms	4	220	56		*	17	3.3
Greeno (1978)	Geometry	6		72		*		
Koedinger (1991)	Geometry	5		98		*		
Luger (1981)	Physics	5		102				
VanLehn & Jones (1993)[h]	Physics	4	500	151	3		114	1.3
Ritter (1994)	Trouble-shooting	1	124	175	1	*	440	0.25
Feldman (1962)	Binary choice	1	1,000[i]	200	5		39[i]	5.1
Anzai & H. A. Simon (1979)	Tower of Hanoi	1	5,400	224	22	*	50[i]	4.5
VanLehn (1991)	Tower of Hanoi	1	5,400	224	22	*	10	22
A. Newell (1990)	Cryptarithmetic	1	200	238	1		54	4.4
A. Newell & H. A. Simon (1972)	Chess	1		242			6	40
Larkin, McDermott, D. P. Simon, & H. A. Simon (1980)	Physics	21	8,000[i]	254	32	*		
Johnson et al. (1981)	Medical diagnosis	12		264			200	1.3
Baylor (1971)	Block puzzle	1	727	423	2		14	30
Peck & John (1992)	On-line database	1	560	624	1	*	430	1.5
Moran (1973)	Visual imaging	1	1,200	656	2		175	3.8
Ohlsson (1982)	Linear syllogisms	6	10,300	1,555	6.6		50[i]	31
A. Newell & H. A. Simon (1972)	Cryptarithmetic	1	810	1,900	0.42	*	51	37
	Cryptarithmetic	5	10,000	2,000	5		50[i]	40
	Logic	10		3,700			50	74

[a]Number of subjects. [b]Total time (sec). [c]Number of segments aligned. [d]Average segment time (sec). [e]Required task (as well as verbal) actions. [f]Number of model units (usually rules). [g]Model data only used. [h]Several analyses ignored strict order, using as many as 4,000 segments. [i]Estimated from other measures in the paper.

1. The data consist of 10 protocols, each approximately 1 min. The analysis would be far more difficult for a single 10-min protocol because there would be far more successive opportunities for the model and protocol to diverge.
2. The behavior is relatively straightforward. A major result of the Browser study is that this browsing behavior corresponded to a simple model without search or deliberation. Yet, the analyses still take considerable time.

Larger protocols and more complex models may prove a challenge, but the Browser corpus forms a good starting point.

Computational Support–The Way Out. As discussed in the remainder of this article, we see the growing ability to use computational tools as a way of making this potentially valuable methodology more tractable. The methodology is in a detailed form so that we can recognize the right tools and algorithms if we saw them and so that we can expand them. In Section 4, we discuss tools that are being used or that could be used to support individual tasks of TBPA.

4. TOOLS FOR BUILDING PROCESS MODELS

Existing tools address some of the individual tasks in TBPA, so we start by summarizing their major categories and then note the lessons they provide when designing an environment to support the full methodology.

4.1. Tools for Protocol Analysis

Simple protocol-coding tools address the labor and bookkeeping of assigning consistent codes to protocols. Although independent of a process model, these tools illustrate the basic functionalities for analyzing sequential qualitative data: (a) presentation of the protocol text, most commonly as a set of segments, (b) modifiable code-lists with easy assignment to segments, and (c) facilities for counting the occurrences of each code or group of codes. Reviews of these tools are available in Bainbridge and Sanderson (in press) and Ritter (1993).

Concept analysis tools code continuous text for the concepts (or declarative knowledge) contained in them–ignoring the sequential order of the concepts. The earliest tool, the General Inquirer (Stone, Dunphy, M. S. Smith, & Ogilvie, 1966), used a database of word meanings to count concepts in documents and could analyze corpuses of 100,000 words. Carley (1988) built computational tools to help code verbal protocols into semantic networks, a more sophisticated representation. These systems

demonstrate that automatic aids can routinely analyze large amounts of verbal data (e.g., automatic book-indexers; Waltz, 1987), but they have not yet been applied to code protocols into labeled sequential elements.

Commercial software is sufficiently sophisticated that its general capacities equal or exceed many simple, specially built protocol-coding tools. Software used in this way includes spreadsheets (Excel [Peck & John, 1992]; Lotus 1-2-3 [Payne, Cohen, & Pastore, 1992]), databases and statistical packages (e.g., Filemaker Pro™), and even multicolumn text editors (Prep; Neuwirth, Kaufer, Chandhok, & Morris, 1990). Some packages also now include a programming environment (e.g., macros in Excel 4.0™). Operating systems on popular microcomputers permit integrated software use. These tools offer tantalizing views of raw functionality and can well support some of the tasks of developing a process model. For example, the row-and-column format of spreadsheets provides a dense display that groups related data together on the same row while showing adjacent data across rows. Databases offer alternate displays of the same data, which can be tailored to the current task. However, obtaining a range of functionality requires sophisticated use of several applications. It is important to note that, with source code unavailable, more graceful integration is rarely possible, and it remains difficult to incorporate the process model itself into an analysis system.

Sequential patterns in protocols are global features that any process model must reproduce, and finding them is often a useful step toward creating a process model. Tools for finding such patterns (e.g., MacSHAPA [Sanderson, 1993; Siochi & Hix, 1991]; Textlab [J. B. Smith et al., 1993]) incorporate sophisticated coding schemes–methods for summarizing data and for relating them to theoretical constructs. For example, MacSHAPA can detect looping behavior (a repeated occurrence of the same code or series of codes) and relate these loops to initiation and completion of a goal. The Textlab tools can produce cognitive grammars as summaries of action patterns. Childes-Clan, with approximately 250 users, is perhaps the most widely used of these systems (MacWhinney, 1991; MacWhinney, personal communication, October 1992). Childes-Clan includes a structured editor and database language for creating a hierarchy of codes, similar in functionality to MacSHAPA, and includes facilities for counting a wide variety of patterns and combinations of codes, including those between two data streams. It was used to create the Childes database of children's early language, which contains more than 100 megabytes of transcribed and annotated protocols. These tools characterize the data with static structures or patterns, in contrast to our emphasis on active process models, but they show the advantage of directly including the model within the tool–the results are presented in terms of the model, and the analyst can directly modify the model based on the results.

4.2. Tools for Aligning Data Elements and Model Actions

Aligning corresponding elements of two sequences–whether codes from two coders or a protocol and the predictions of a model–is a ubiquitous and tedious part of ESDA. There are two complementary ways of supporting this task. When no subtle judgments are required, automatic alignment algorithms are available. They can process relatively vast amounts of data (e.g., Card et al., 1983).

In most instances, however, comparing verbal data and model predictions exceeds interpretation abilities for automatic tools. A task-specific editor can facilitate this by-hand alignment. Trace&Transcription (John, 1994) displays elements of the two sequences in side-by-side columns. The user clicks on two elements to be aligned, and the system adds additional blank cells in one or the other sequences to bring the elements into alignment.

4.3. Tools for Automatically Developing Process Models

Machine learning tools have been used to develop a model from protocols. The earliest attempt, PAS–I and PAS–II (Waterman & A. Newell, 1971, 1973), consisted of a detailed, 28-step procedure, together with a computational tool incorporating an editable process model. These programs coded verbal protocols into problem behavior graphs–diagrams that represent the actions of a process model as links between hypothesized states of knowledge (A. Newell & H. A. Simon, 1972). These systems suffered from primitive display technology and a baroque command-line interface, but their major difficulty was the inability (still present today) to automate natural language interpretation. Working from hand-coded protocols bypasses this problem so that problem behavior graphs (Bree, 1968) or decision rules for choosing operators in a preexisting model (Langley & Ohlsson, 1984; Kowalski & VanLehn, 1988) can be automatically created. ASPM (Polk, 1992) adjusts a set of knowledge-use parameters to capture reasoning patterns of individual subjects, with new computational algorithms handling the resulting complexity.

Semi-automatic modeling tools analyze coded protocol data to test a predicted task ordering (SAPA: Bhaskar & H. A. Simon, 1977) or to identify which of a set of models best characterizes the protocol (KO: Dillard, Bhaskar, & Stephens, 1982; Gascon, 1976). Ethno (Heise, 1991) iterates though a database of protocol events asking the analyst to identify precursors for each event and automatically creating rules representing the reported causal relations.

Achieving automatic refinement of models from data has allowed relatively few possible types of adjustments in the model and has required well-developed initial models along with well-defined relations between the model and data. By incorporating the model being tested, these systems have been able to tailor their displays and system responses and, in their limited way, illustrate the possibility of making process models a routine and tractable way of characterizing sequential data.

Some intelligent tutoring systems automatically code students' responses and collect them as direct tests of the process model underlying the tutor (e.g., Anderson, Farrell, & Sauers, 1981; Sleeman, Hirsh, Ellery, & Kim, 1990). The initial process models used in the tutors may come from previous by-hand analysis (cf. Singley, 1987), but automatically analyzed data from the tutor are also used to refine existing theories (e.g., Koedinger & Anderson, 1990). This work is impressive psychologically, pedagogically, and in its ability to routinely compare protocols with the performance of a model. However, it cannot be simply extended to analyzing any behavior. Actions taken by the student are limited to those permitted by the tutoring system, and the underlying model must be well developed. Finally, when a student's actions mismatch the actions of the model, the student can be "reset" through hints or actions performed for the student–bringing the student back into synchronization with the model. This is easier to control than resetting the model. Despite these limitations, this work illustrates a practical payoff and shows how process models can be used to routinely characterize large amounts of data.

4.4. Architectures and Expert Systems

Process models consist of (a) the knowledge used to perform each particular task and (b) an architecture–a fixed set of structures underlying performance on all tasks–to interpret and apply the knowledge (A. Newell, 1990). Architectures allow a variety of models to be built on a framework of common assumptions. Soar (A. Newell, 1990) is an example of such an architecture (for other examples, see Anderson, 1993; ACM Special Interest Group on Artificial Intelligence [SIGART], 1991). As a proposed unified theory of cognition, Soar is intended to capture basic principles of human cognitive function sufficiently well that it can be used to model all cognitive tasks. Although the vision exceeds the current reality, these architectures, particularly Soar and ACT, have been used to model a wide variety of tasks, including perception, social interaction, problem solving, and natural language processing. If an architecture is incorporated into model-building and data-analysis tools, then the model developed can be more readily explained in terms of the architecture, and the analyses can be specifically appropriate for and based on the mechanisms in the architecture.

Expert systems are process models of expertise built with the aim of producing the same results as human experts, but not necessarily by using the same reasoning process. Development environments have been designed to build expert systems–providing an architecture and an associated, structured, integrated editor. Knowledge-based knowledge-acquisition tools have built on this framework using an expert system recursively to monitor the process of further system building–for example, by pointing out incomplete knowledge structures. (See ACM SIGART, 1989, for a summary.) Some knowledge-acquisition tools (e.g., KEATS: Motta, Eisenstadt, Pitman, & West, 1988) include the ability to tie verbal protocols or other texts to various facets of the model–showing that a protocol segment has had its knowledge extracted. This work typically does not analyze data in the same way psychologists would but shows that better tools for building and understanding process models are possible, though expensive.

Tools for understanding why a process model produces a particular sequence of actions are an ongoing need, but, as yet, there are few examples. Displays for understanding these models are typically limited to the usual tools of programming languages–the ability to trace the action of the surface behavior of the program and to examine elements of the current state.

4.5. Implications for an Integrated Environment

The Browser example and this review of computational support tools suggest that a computational environment supporting TBPA should have the following four features.

Complementary Automation and User Support. We have noted repeatedly the amount of by-hand analysis required in TBPA, for it is impossible to automate completely all the tasks. An environment for TBPA cannot at this time be like a statistical package, in which data go in and interpretative results come out. One must blend gracefully the automatic facilities with support for doing tasks by hand–what we call semi-automatic analysis.

The variety of protocol-encoding tools illustrates this range. The semi-automatic protocol-coding tools present protocol segments conveniently, provide modifiable menus of codes, and provide facilities for counting the occurrences of each code or group of codes. Tools for finding patterns in protocols automatically identify larger scale sequential patterns in these data. The power of concept-analysis systems suggests that automatic tools may one day be able to code natural language protocols.

Interactive Displays Supporting Analysis Tasks. Evaluating process models involves large amounts of information, including both sequential data and sequential output from the model. Well-designed displays can show extremely large amounts of information so that the important patterns become visually apparent. Larkin and colleagues (Casner & Larkin, 1989; Larkin & H. A. Simon, 1987) have suggested that displays are useful if they reduce search for needed information or allow easy spatial judgments to replace more difficult analytic judgments. Thus, the maximum of a set of numbers is easier to see in a graph than in a list. Displays need not be limited to graphics but can include formatted text (outlines, spreadsheets, semantically meaningful indentations in code and traces).

Such displays can be forceful final summaries of work, but they are most useful as part of the ongoing interaction between the model and data. Because these displays involve large amounts of information, they cannot readily be produced by hand. Therefore, we believe that a good support environment should produce useful information summaries automatically and easily. The graphical summaries should also manage access to the large amount of information, letting the user see currently important information and then shift views or elaborate elements for further information.

Integration of the Process Model. Using a model to account for protocol data is easier if it is consistently available throughout the environment, providing a structure for all other tasks. This organization is hard to realize unless the model is integrated with the analysis system. We have also observed the need for tools and displays to illustrate the major structures and functions of the model. In an integrated support system, these tools are readily available throughout the analysis. Additionally, as we illustrate later, there can be tools that summarize the current correspondence between the data and structures in the model and to test alternate model forms.

Expandability. No application, even for relatively simple tasks (e.g., graphing), ever provides all the functionality a user needs. This is particularly true for exploratory analyses like model building and testing. Serious and innovative users always move beyond the tool developers' imagination. This intrinsic limitation is particularly relevant for research tools, because the aims and methods of research vary widely, even within a single methodology. Therefore, an environment for TBPA must be expandable.

5. A PROTOTYPE ENVIRONMENT FOR DEVELOPING PROCESS MODELS

We have started to explore, using a prototype system, how a unified computational environment can enhance the value of TBPA, as specified Figure 1. This prototype environment incorporates a modeling architecture, as we have argued it should, and takes its name, *Soar/Model-Testing,* from the Soar architecture. However, many of its tools would work equally well with other architectures, and we discuss here implications that transcend any particular architecture.

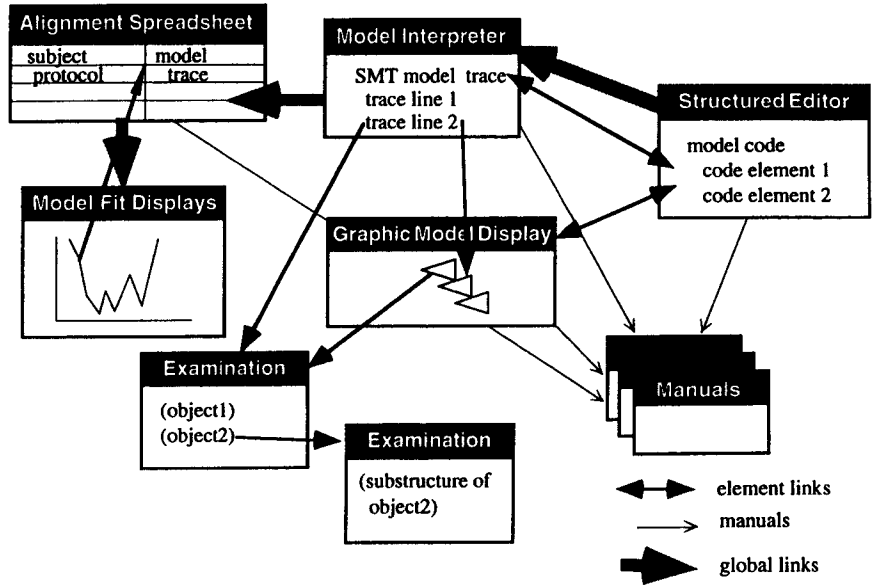
Figure 4 shows the interactive displays of the environment and their information links. Starting with Task 0 of the TBPA methodology, models are constructed in the Structured Editor and run in the Model Interpreter, which provides a trace and other diagnostic aids, such as the ability to examine model structures and substructures. Additionally, displays can summarize static and dynamic structures of the model. As specified for Task 1, the design for the trace of model actions facilitates alignment with sequential data. To begin Task 2, aligning data with the model trace, sequential data are entered or imported into the Alignment Spreadsheet, which also holds a trace of the model. An automatic algorithm and tools for semi-automatic alignment jointly support matching the data and trace. Task 3, assessing the performance of the model, uses the Model-Fit displays that summarize how well the static and dynamic structures of the model correspond to data. We discuss how the entire environment helps to suggest revisions for improving a model (Task 4). Additionally, a "pseudo-revision" process can suggest possible effects of a proposed revision without altering the model itself.

The SMT environment supports each of the TBPA tasks, as already discussed. However, SMT components are integrated, allowing ready movement between tasks. Many displays help manage the large amount of information intrinsic to TBPA by showing selected information, which the user can elaborate as needed. Manuals and on-line help facilities are linked to most of the displays. Information on how to obtain the SMT components is available from Frank E. Ritter.

5.1. Building a Model (Task 0)

A structured editor (Hucka, 1994; Ritter, Hucka, & McGinnis, 1992) supports creating and revising model code; provides commands to facilitate searching and manipulating the model; and includes templates for common code structures. Integration within the environment allows modifying and examining the underlying elements selected in the running model.

Figure 4. **Displays and information links in the SMT environment.**

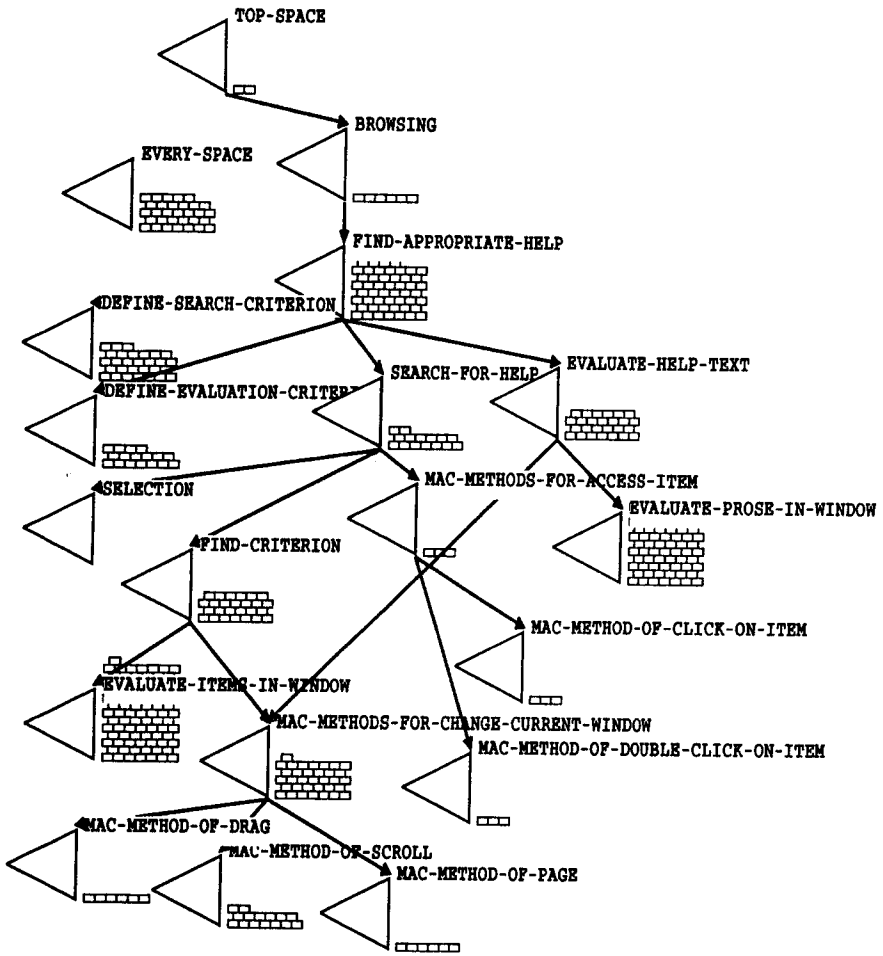


Creating a computational model of behavior and modifying it to better account for data require a deep and complete conception of how the model works. As all programmers know, reading the model code, or even reading a trace, is often insufficient to provide this conceptual structure. The SMT Graphic Model Display (Ritter & McGinnis, 1992) shown in Figure 5 builds from the model code a diagram showing spatially the elements of the model and their relations–in this case, Browser. The display is interactive and can show varying amounts of detail. The particular notation, triangles for methods (or problem spaces) and circles for actions (operators), comes from the community using the Soar modeling architecture. The bricks indicate individual model elements (productions)–reflecting the complexity of each method and giving access to these rules by clicking on them.

5.2. Running a Model to Produce a Trace (Task 1)

The Soar model interpreter was modified to produce a more compact and less ambiguous trace of the actions of the model. The top of Figure 6 shows a simplified version of this trace. Each line corresponds to a predicted subject action (e.g., move mouse to the hierarchical menu) or a mental decision (e.g., decide to evaluate the help text) that could match an

Figure 5. Structure of Browser code produced by the SMT Graphic Model Display.

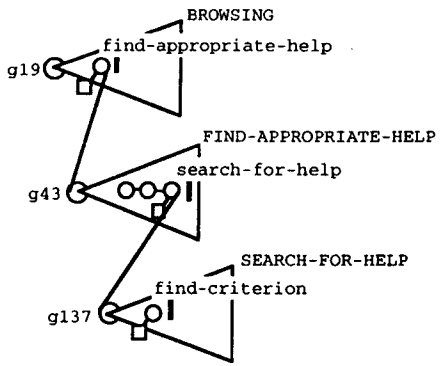


utterance or a mouse movement over text being evaluated. The line numbers correspond to the time units of the model. The indentations and dots reflect the hierarchical structure of actions and methods that implement them. The spacing and format allow the trace to fit directly and compactly into the spreadsheet and to be automatically aligned. (Os in the trace indicate Soar operators, which correspond to what we call *actions* in the Browser model. Ps are problem spaces, which correspond to methods.)

A trace is simply a log of main events. It usually gives little insight into how these events correspond to the internal structure of the model. To serve this need, SMT provides a dynamic graphic display of the current structure of the model. Figure 6 includes a graphic display corresponding

Figure 6. Simplified trace of model actions and graphical representation of dynamic model state.

```
 5  .   P: browsing
 6  .   S: s39 (To-be-found: unknown)
 7  .   O: find-appropriate-help                    Current
 8  .   => G: find-appropriate-help
 9  .   .   P: find-appropriate-help
10  .   .   S: s59
11  .   .   O: define-search-criterion
12  .   .   => G: g65
13  .   .   .   P: define-search-criterion
14  .   .   .   S: s79
15  .   .   .   O: generate-search-criterion
                    (to-be-found: write)
16  .   .   .   O: evaluate-search-criterion        Past
17  .   .   O: define-evaluation-criterion
18  .   .   => G: g103
19  .   .   .   P: define-evaluation-criterion
20  .   .   .   S: s117 (unknown)
21  .   .   .   O: generate-evaluation-criterion
22  .   .   .   O: evaluate-evaluation-criterion
23  .   .   O: search-for-help
24  .   .   => G: g137
25  .   .   .   P: search-for-help                  Current
26  .   .   .   S: s155 (to-be-found write)
27  .   .   .   O: find-criterion (keyword)
```



to the trace. In the trace, the "find-appropriate-help" method (P) arose from the action (O) of the same name. In this method, several actions and their methods have been used, and the model is now applying the "search-for-help" method still within the find-appropriate-help method. The graphical structure shows this current state directly. As the model runs, new lines appear in the trace, and the graphic display is updated, with new circles appearing for actions and new triangles for methods. When a method is complete, the triangle disappears. As an example of its utility, work using this display has suggested that many Soar models do not just do search in problem spaces and that operators are less concrete than first thought (Ritter, 1993).

5.3. Aligning User and Model Actions (Task 2)

To support the difficult task of aligning the data and trace, SMT provides a specialized spreadsheet (Nichols & Ritter, 1994). As shown by the representation of this tool in Figure 7, its format is quite different from the single-element displays often used in protocol-coding tools, reflecting a central difference between protocol coding and TBPA. In protocol coding, one wants available all information about the protocol element, formatted well, with space for categories and annotations. Many argue (e.g., Ericsson & H. A. Simon, 1993) that this coding should be done independent of context (i.e., without immediate access to the surrounding elements). In contrast, TBPA involves matching data and model-action sequences, and so the sequences must be visible. Additionally, this format produces meaningful, emergent features–the filled space of regions where the model and data match closely contrasting with blank space in either the data or model metacolumns indicating unmatched sequences.

Columns representing data and those representing the model trace are grouped together into two metacolumns (separated by a heavy line in Figure 7). The model metacolumn (on the right) includes the step number of each action (DC), the model trace, and related notes or annotations (not shown). The protocol metacolumn (on the left) shows the time (T) of the action, with separate columns for mouse actions (MSE), responses from the interface (INT), and verbal utterances (VBL). The type of protocol segment (m = mouse movement, b = mouse button action, v = verbal) appears in the ST column. The type of model–data match (mr = mouse movement required to do the task, mba = mouse button action, v = verbal utterance) appears in the TYP column. The column headed MC indicates the number of the trace action matching the protocol data element. This column is particularly useful when items match out of order (e.g., if Segment 21 matched Line 121) or when several items in one sequence match one item in another sequence. Metacolumn rows must stay together because each represents a single data or trace element. Therefore, during alignment, the spreadsheet moves entire rows in a metacolumn relative to the other metacolumn. The user can align two metacolumn rows simply by selecting them and executing a keystroke.

If there are well-defined criteria determining whether two elements match, an automatic algorithm (described in Figure 8) can align element sequences. The technique worked well with the Browser corpus, correctly aligning all 296 task actions (mouse clicks and movements to them). Even if only a fraction of the elements are aligned automatically, not only is part of the job done, but the aligned pairs subdivide the rest into smaller and thus simpler subsequences. For example, the Browser corpus has about 600 sec of protocol. The 296 automatically aligned task actions, on average, position the remaining data within 2 sec or about 20 lines of their final

Figure 7. **A simplified portion of an alignment spreadsheet.**

T	MSE	INT	VBL	ST	#	TYP	MC	DC	MODEL TRACE
58	M(-y) 1 line to 'scalex' in			m	19	mr	92	92	. . . O: move-mouse (to hier. win.)
	C("scalex Setting the Scales			b	20	mba	93	93	. . . O: click-button
		mouse pointer to watch							
59		'scalex' help text appears							
59			"Let's look at	v	21	v	94	94	O: evaluate-help-text
		mouse watch to pointer						95	->G: g725 (operator no-change)
								96	. P: evaluate-help-text
								97	. S: s741 accessed: scalex
								98	. O: focus-on-help-text
62			"scale X- and scaleY- commands"	v	22	v	99	99	. O: evaluate-current-window

alignment. If the algorithm proves untractable on the total corpus, the user may be able to identify parts suitable for automatic alignment and then work by hand to place these subparts within the whole.

5.4. Assessing the Model (Task 3)

The aligned corpus is typically complicated and lengthy, so we need summaries to show in what ways the model does and does not account for the data. Although the spreadsheet provides some informative local information, more powerful summaries are necessary. We have created two such displays. To aid in relating the summaries to the details of the spreadsheet, the graphical summaries are "live," in that clicking on a point displays further information about them. These assessment tools go beyond a single assessment number (e.g., goodness of fit) and show where to improve the model (Grant, 1962).

Dynamic Structure in the Model. The model-support display shows the dynamic behavior of the model. Thus, it is another tool, like those discussed in Section 5.1, for understanding the dynamic behavior of the model. We discuss it here because it can also serve as a basis for model–data comparison. This display is a graph of each model action as a function of when it occurred in the action sequence (see Figure 9). The horizontal axis shows the sequence number of an action; on the vertical axis are names of actions. The small black squares indicate that the model performed the action at the left at a location in the sequence indicated by the position on the horizontal axis. Lines through these points link them sequentially. Actions on the vertical axis are ordered and indented to correspond roughly to their place in the action–method hierarchy. Figure 9 shows one short browsing episode or about one 15th of the Browser corpus. Peck and John (1992) showed a similar hand-drawn display.

Emergent features in this display reflect dynamic features of the model. For example, in Figure 9, the small periodic cycles occur as the model repeatedly reads the contents of a scrolling menu and scrolls the menu to

Figure 8. **SMT's automatic alignment algorithm.**

Using as a basis the alignment algorithm of Card, Moran, and A. Newell (1983, chap. 5), we developed the following algorithm for use in SMT, which extends theirs in four ways:

Allowing equivalent symbols. By providing pairs of regular expressions, the user defines equivalencies between the data and trace sequences, allowing different tokens in the two sequences. Thus, "click" in the data can match "operator-click-button" in the trace.

Computing an actual sequence of forward matched pairs. Both algorithms start by creating a matrix corresponding to String 1 along the top and String 2 along the side. A simple walk across the matrix and comparison of the string elements corresponding to each cell build up a measure of the maximum cumulative matches up to that cell (but not what they are). The value of the last cell holds the length of the maximum common subsequence. Our algorithm then walks across the matrix again, using the match numbers to identify corresponding pairs.

Four patterns can appear in the matrix: (a) If the next symbols in the streams match (now indicated by the cumulative match values), then the walk moves diagonally, adding a pair of matched tokens and their locations. A completely matching subsequence thus corresponds to a diagonal path across the matrix; (b) if the next two symbols are completely unmatchable, the algorithm moves diagonally, inserting two pairs of tokens and blanks, indicating that these tokens are to be unaligned; (c & d) the cumulative match cells may indicate that the longest match required is obtained by not aligning elements in one of the streams—that is, moving vertically or horizontally to find another matched pair. In this case, an element from a single trace is noted as matching a blank. If one sequence is completely matched, then the remaining elements in the other sequence are simply matched to "blank."

Updating the spreadsheet. The user is first offered the opportunity to examine and modify the pairs found by the algorithm. These correspondences and their locations are then used to align the elements matched by the algorithm through shifting the metacolumns in the spreadsheet.

Flexible modification. The matched sequence found by this algorithm depends on the starting location of its "walk" and its route through the matrix. Generally, starting at the beginning—where the model and trace are most likely to match and the match will be most stable—is a good heuristic. However, in the Browser corpus, there often is a series of mouse movements that ultimately place the cursor over a menu item, with this last movement the one that corresponded to the movement of the model. We modified the algorithm so that it adjusts which of the several best subpaths it uses to include the last matchable item of a series of equivalent items in the data sequence.

see more. These and larger cycles illustrate what Peck and John (1992) called "routine" behavior–a constant cycling through actions and methods that implement them, without interruption or search for alternatives.

Empirical Support for Model Structures. The model-support display in Figure 9 uses the behavior of the model as a backdrop for viewing the

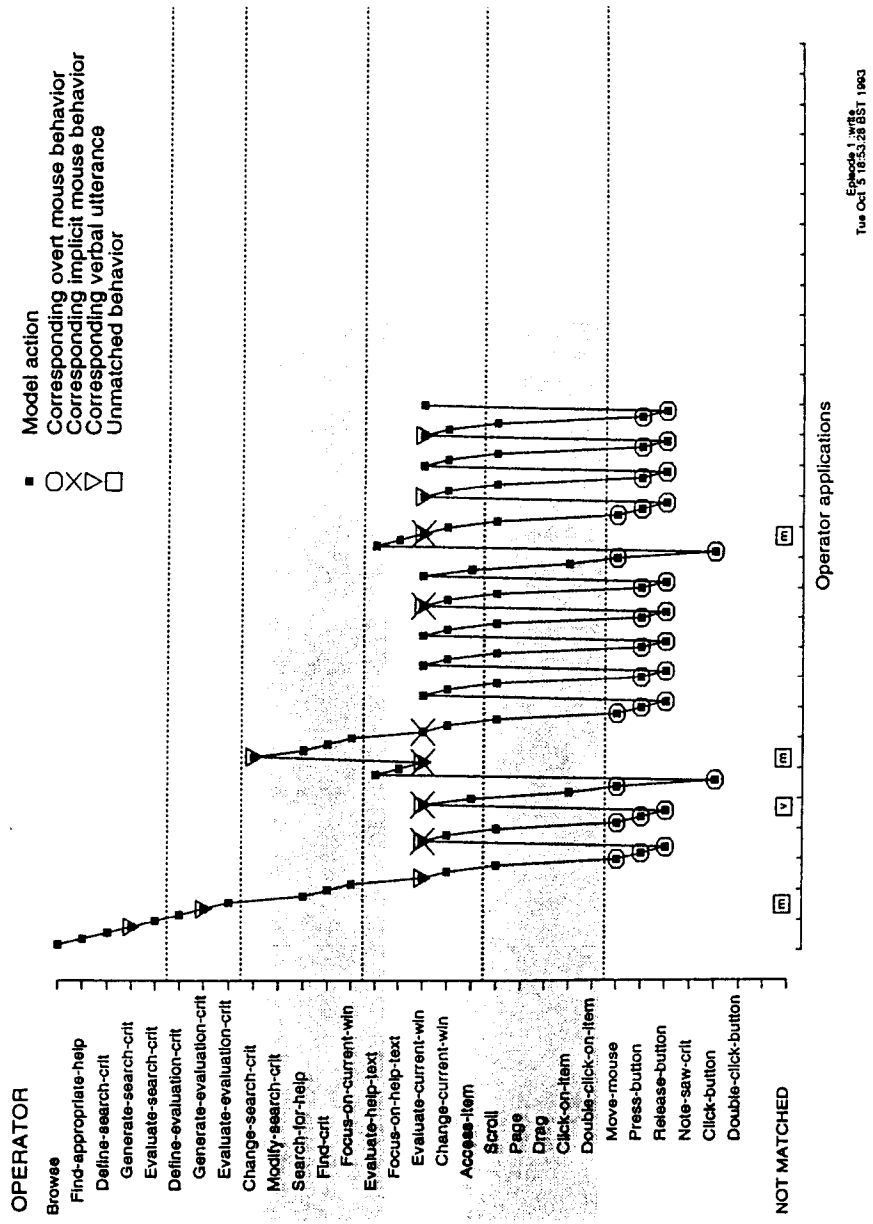


Figure 9. **Model-support display showing model actions, organized by method, annotated with corresponding subject actions.**

behavior of the subject and so assessing how well they correspond. Each small, black square is a model action, and annotations indicate corresponding elements in the protocol–each annotation reflecting a protocol–model match in the alignment spreadsheet. Unmatched model actions remain as unannotated dark squares, and unmatched protocol actions appear in temporal sequence at the bottom of the graph.

As discussed by Ericsson and H. A. Simon (1993), Figure 9 shows that most model actions are not matched by data. Verbal protocols are always sparse compared to a process model. People do not report everything that is part of their processing. In the Browser corpus, the use of motor actions augments the data considerably, but the evidence for a model must still rest on consistency between the model and those data points we have.

For the episode shown in Figure 9, two action groups (associated with "double-click-on-item" and "focus-on-help-text") receive strong support. Protocol elements correspond to many or most of the model actions implementing these methods. In contrast, the "access-item" method receives no direct support from these data. The only evidence for it is the support for an action used to implement it, "double-click-on-help-text."

Temporal Match Between Model and Data. Sequential data, if they includes timing information, allow testing of temporal predictions of a process model. The SMT processing-rate graph (Figure 10) shows subject time on the horizontal axis and model time on the vertical axis. Points on the graph are protocol–trace pairs, with the symbol indicating the type of element, as in Figure 9. Unmatched protocol actions again appear at the bottom. Again we use one episode from the Browser corpus, but, because the Browser model was never intended to make temporal predictions, this post hoc analysis serves merely as an illustration of our techniques. Sakoe and Chiba (1978) used similar displays to understand speech-recognition models.

If the temporal predictions of the model were completely accurate, and the model and subject are executing comparable processes, all the points would lie on a straight line with intercept 0 and slope 1. Other patterns indicate particular inconsistencies with the way the model fits the data. Specifically:

1. A straight line with slope unequal to 1 indicates that the unit of model time is different from the unit of protocol time–a suggestion to recalibrate the times associated with basic processes in the model. Regions with differing slopes indicate that different structures of the model may each perform tasks with a time proportional to that of the subject, but the "model time" units–the time associated with some basic processes–are inconsistent between the structures.

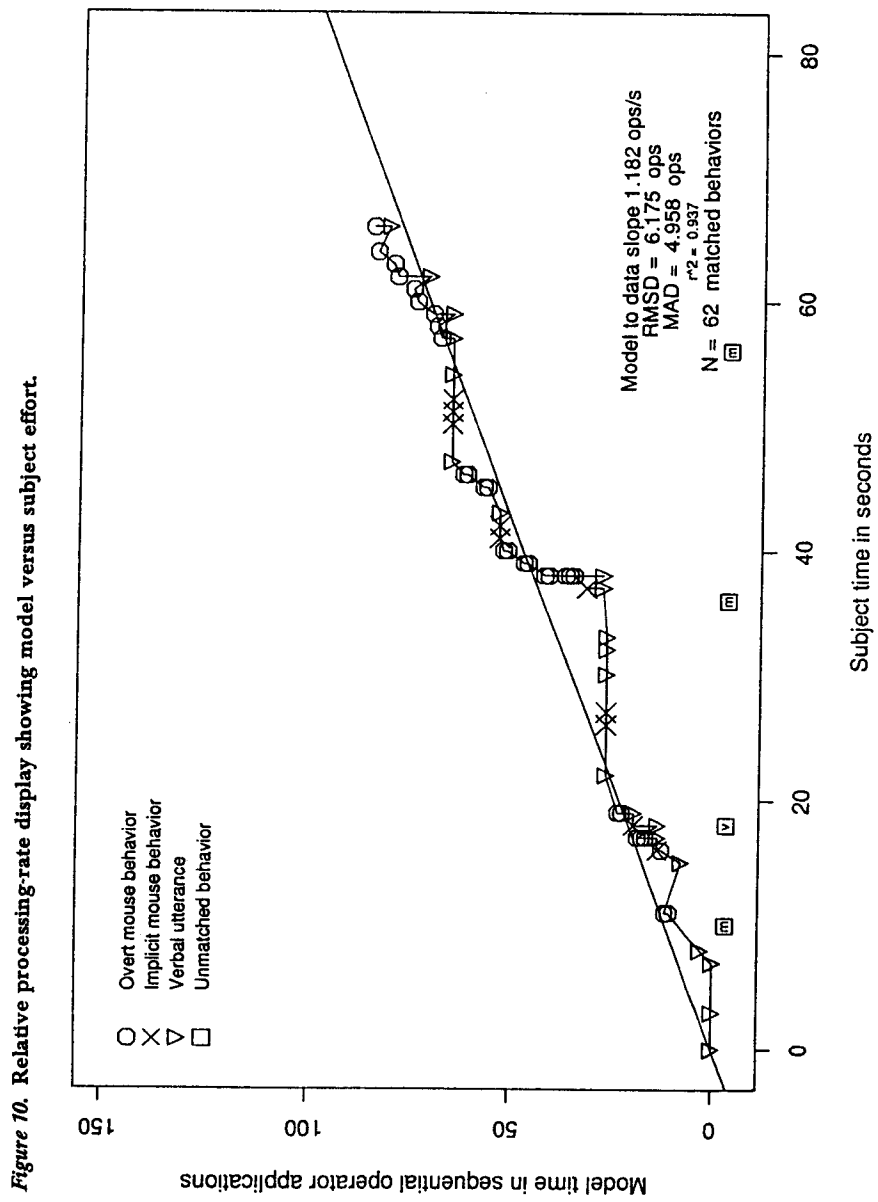


Figure 10. **Relative processing-rate display showing model versus subject effort.**

2. During nearly horizontal regions, the subject performs actions slowly relative to the model. Perhaps the model performs the task in an overly simple manner. The unmatched protocol elements might suggest more realistic mechanisms for this subtask.

3. During nearly vertical regions, the subject performs actions rapidly relative to the model. Perhaps the subject is acting automatically, whereas the model executes more sequential steps. The model may need the ability to perform these actions simply or directly, without using submethods.

4. A downward concave line suggests that the processing rate of the model is decreasing relative to that of the subject, perhaps because the subject was learning more than the model. An upward concave line suggests the opposite.

5.5. Revising the Model (Task 4)

The function of model-assessment displays is less to demonstrate the global adequacy of a model than to suggest ways in which the model might misrepresent processes and thus might be improved. Although it is not possible to specify an algorithm for assessing particular model deficits indicated by a collection of results, we discuss at the end of the Section 5.4 how the processing-rate display might suggest model improvements. We continue that list here, indicating how the various other SMT displays might suggest model revisions.

5. Blank regions in the protocol metacolumn of the spreadsheet, and many unmatched protocol actions in the summary displays, indicate where and how the model is incomplete. The content and function of the unmatched protocol material might guide expanding the model–suggesting new actions, methods, or mental structures.

6. Long matched sequences with out-of-order substructures suggest problems with the fine-control structure of the model or with the conditions associated with major methods.

7. Unannotated methods in the model-support display suggest that the model is doing too much. Methods could be removed or could have their conditions specialized to avoid the problematic situations.

Pseudo-Revision. Because exploring model improvements is a central task, SMT provides a "pseudo-revision" facility to test some implications of model revision without recoding the model. Specifically, users can delete (or add) tasks and methods and propagate these changes through the summary displays as if they were produced by a running model. Thus, simple changes can be assessed roughly but quickly. For example, in the Browser corpus, we "removed" the intermediate Macintosh methods (e.g., "access-item"), implementing higher actions directly with mouse actions.

Propagating these changes through the summary displays showed that the density of the support increased in the model-support displays and that the relative processing-rate display did not appreciably change. (However, B. E. John, personal communication, interpreted these changes differently.)

5.6. Exploratory Data Analysis With TBPA: An Illustration

TBPA is an exploratory data analysis technique, and the SMT environment provides powerful tools for this exploration. When one applies such tools, sometimes unexpected results appear. This is as it should be with exploration.

In the processing-rate graph (Figure 10), there are regions that have a sawtooth appearance because the verbal utterances (∇) are systematically lower (later) than the surrounding motor actions (circles). This indicates that the subject's utterances appear later than one would expect from the surrounding motor actions. This difference raises the question: How are verbal utterances generally related to their nearby motor actions? (However, recall that the Browser model was not designed to predict times. Therefore, the following discussion is purely an exploration of our techniques and of possible implications from the Browser corpus.)

One might expect that the answer relates to processing load–giving a verbal protocol is essentially a secondary task. If processing load from the primary task is high, then verbal utterances may be delayed, as with any secondary task. If this view is correct, it could aid HCI designers in the difficult task of assessing the processing load imposed by an interface design by providing a more principled measure of task load.

In principle, a detailed process model could predict processing load by direct assessment of the internal processing required. For example, if the time allotted by the external environment is too short to perform the necessary internal processing, a high processing load (or at least a sense of stress) is likely to result. Alternatively, the model might process a lot of information internally, without any way of storing intermediate results externally. Both situations create high processing load and would predict corresponding slowness to respond to a secondary task. (For efforts to model processing load, see Gray et al., 1993; Olson & Olson, 1990; Payne et al., 1992.) Even with a less detailed model, we can regard verbalization as a secondary task and use the predictions of the model as a base from which to measure verbalization delay. This may give us a continuous picture of processing load through the course of the task. Alternatively, lags that make no sense may indicate where to improve the model.

Examining these lags is also a way of testing Ericsson and H. A. Simon's (1993) assumption that verbalization order corresponds to the order in which verbalization content enters working memory. If this assumption is correct, most verbalizations will be current–that is, the structures they

refer to should be in sequence with nearby motor actions or should follow them closely.

We use task actions (A) as a "clock" with which to measure lags of verbalizations (V) of matching model structures. Task actions, because they are unambiguously matched, form a reliable baseline. If the alignment shows the sequence A1–V–A2 for both the model and the protocol, then we define the verbalization V to have a lag of 0 (within the grain size of the model). If, however, the model shows A1–V–A2–A3, and the protocol shows A1–A2–A3–V (for Vs matched in content), then verbal production is lagging behind other operations. We define *lag* in this case as the model time of the preceding task action (A3) minus the model time of V.

In the Browser corpus are 195 aligned verbal utterances. For 145 of them, there is no lag–the verbalization occurred in sequence with task actions. For 46 of the remaining 50, the lag was within 100 units of model time, roughly 10 sec (A. Newell, 1990). One verbalization occurred earlier than predicted, and 3 were significantly delayed (300 to 400 model cycles). Upon further inspection, in each of these "delayed" verbalizations, the subject was reviewing an earlier part of the search. Thus, these events seem roughly consistent with the current model but suggest a review process the current model does not include. Originally, two utterances occurred earlier than their predicted time. One was a coding error. The second occurs in a long menu-examination sequence. Study of the processing-rate display indicates that the model is slow compared to the subject. (The model reads every item; the subject skims.) Thus, the baseline is probably too slow, rather than the utterance being too far ahead.

Implications for Verbal Protocols. In the Browser corpus, all verbal utterances were sequential with respect to other utterances, and most of them (145 of 195) were in sequence with associated motor actions. This supports Ericsson and H. A. Simon's (1993) assumption that verbal protocols reflect current working-memory content and are reported in the order they enter working memory. The Soar architecture links model time to human time (about 10 cycles/sec). This gives us a rough measure of the lag of concurrent verbal protocol under minimal task demands–roughly 1 sec (9 model cycles was the average lag). (However, recall that this exploration is based on a model never intended to make time prediction.)

5.7. Expandability

SMT has been built with available, largely free tools in the Unix environment. This allows a knowledgeable user to extend it to meet specific needs (e.g., different architecture or different summary displays). The Graphic Display was built with Garnet (Myers et al., 1990) and the Soar5 interpreter (Laird, Congdon, Altmann, & Swedlow, 1990). The Modified trace relies solely on the Soar interpreter. The Summary Displays were built with S (Becker, Chambers, & Wilks, 1988), using S-mode (Bates, Kademan, Ritter, & D. Smith, 1990) within the GNU-Emacs editor (Stallman, 1984). The model editors and the alignment spreadsheet are also based on the GNU-Emacs editor.

6. CONCLUDING REMARKS

Process models characterize sequential data through an algorithm that can replicate those data. The precision and dynamic information of such models have much to offer, but the models have not been used as extensively as one would expect given their benefits. The major reasons may be that both the models and associated data are complex and voluminous, the methodology for developing them has been ill-defined, and the effort involved in manipulating them can be overwhelming (especially without good computational tools).

6.1. TBPA and Its Computational Support

We have explicated TBPA–a methodology for developing process models to account for sequences of human behavior. In essence, the methodology involves (a) building a computational model capable of producing a sequence of actions that can be compared with the actions in a sequential data set, (b) aligning and matching the model output with data, (c) interpreting which mechanisms of the model succeed and which fail in accounting for features of the data, and (d) using this information to revise and improve the model. Analyzing the requirements and difficulties of each step contributed to the design of computational aids to make the TBPA methodology both tractable and more informative. The requirements and difficulties for each step were noted in some detail so that they could be supported computationally.

Using our specification of TBPA and a review of the software used to perform some of its tasks, we identified the following needs: (a) complementary automation and user support, (b) interactive displays, with information selected and spatially organized to facilitate users' tasks, (c) support within a single, highly linked environment, and (d) expandability by the user. These features appear to be important to many types of sequential data analysis as well.

Our prototype integrated environment, SMT, illustrates the following functionalities, which we see as central to the best use of process models: (a) representations of the model, including its static structure and the dynamic structures that emerge when the model executes, (b) automatic and manual tools for interpreting sequential data and aligning them with the predictions of the model, and (c) displays and measures indicating how

functional structures of the model account for data. Applying this prototype to a model and corpus of data illustrate, as is to be discussed, how good tools allow more extensive benefits from a process model. The evolution of the environment is ongoing, as further models stretch its capabilities (Ritter, 1994) and the underlying architecture (Soar) changes.

6.2. Benefits of Computationally Supported TBPA

Principled Analysis of User Behavior. Creating a process model requires specifying knowledge sufficient to perform the task. This knowledge specification can be used to measure task load, to explain why HCI designs are successful (e.g., why a novice can easily use a particular help system), and to ensure consistency and completeness in manuals as well as in the interface itself. Once created, a process model can represent a potential user at any time. Given a new interface or, more realistically, a modification of an interface, the model should show what a user will do. Other theories of interface use are approximations of this level of understanding. For example, design guidelines attempt to steer the design toward interface designs that will produce useful patterns of behavior, but they do not characterize actual behavior. Process models can make quantitative predictions (e.g., about time and knowledge-based errors in using a particular interface). The processing-rate display can help develop models in this direction. The tentative measure of how long verbalizations typically lag coordinated motor actions (about 1 sec) contributes to the science of psychology as a foundation of HCI and demonstrates that mouse and verbal actions can give a consistent picture of user activity.

Improved Communication. In both computer science and psychology, it is difficult to formulate good model summaries that are both simple enough to communicate and faithful to the important details. The SMT displays have been popular in papers and presentations in the Soar community and in workshops introducing Soar models (Lehman et al., 1994). This popularity seems to reflect the ability of the display to show the dynamic behavior of the theoretical objects in the model. Similar dynamic direct-manipulation displays should be useful in describing the structure of any process model.

We illustrated the general utility of several types of summaries of sequential behavior: (a) The model itself provides an active summary that can both replicate human behavior and be analyzed or modified to suggest behavior in new situations; (b) the degree of correspondences between model actions and data actions in the alignment spreadsheet summarizes, in a simple way, the quality of the model; and (c) graphic displays, such as those we presented, both highlight visually the larger scale regularities in human behavior (e.g., periodicity) and how they can characterize the model–data match. The displays also illustrate where the model does well (or poorly), so that one knows where to trust its predictions.

Time Savings and Its Implications. An obvious advantage of appropriate computational tools is that they save time. For example, we were able to replicate the original analyses for each Browser episode in about 90 min, including creating both graphic displays. In contrast, the original analysis (not including graphic displays) required at least 10 hr. A model support graph requires about 6 hr with a drawing package or about 10 min with SMT.

These savings go beyond merely getting the job done faster–they allow a qualitative change in how one works with a model and data. When the examination of one protocol episode requires a day or more, the analyst becomes immersed in the details and loses track of general questions and issues. Exploring an odd idea about the data is hardly an option if it requires 3 weeks. Tools like SMT let analysts routinely examine how well a model accounts for data and what changes might improve this account. This approach brings to large amounts of sequential nonnumerical data some of the ability to examine and explore the data that cell means and sketched graphs provide for quantitative data.

We hope that these ideas and illustrations will encourage additional depth and rigor in use of sequential data and will allow wider use of process models as action summaries. In the Browser case, for example, we were readily able to examine all the data with all the tools and displays–thus supporting stronger statements about how well the general features of the model really characterize all the data.

More Complete and Varied Analysis. We were able to explore a variety of analyses and displays, selecting those that proved most informative. In this environment, we illustrated the general utility of several types of summaries of sequential behavior: (a) The model itself provides an active summary that can both replicate human behavior and be analyzed or modified to suggest behavior in new situations or used to predict behavior with a changed interface; (b) the degree of correspondences between model actions and data actions in the alignment spreadsheet summarizes, in a simple way, the quality of the model; and (c) graphic displays, such as those we presented, both highlight visually the larger scale regularities in the model–data match (e.g., periodicity) and indicate how well various parts of the data are supported by data.

Opportunity to Explore. A small feature of one of our displays (the jagged segments in the relative processing-rate displays) suggested new implications of previously analyzed data. With good tools, we could readily explore this possibility–arriving at an unexpected result on the

processing load in the task–and test several theoretical statements about verbal protocol data. Letting the data tell you things you never thought of is a crucial part of science, particularly of ESDA, and has often been obscured for process-model work by the inability to stand back and summarize the sequential data and predictions that are at the heart of the method.

NOTES

Acknowledgments. This article is based on Frank E. Ritter's thesis, co-advised by Allen Newell and Jill H. Larkin. Allen Newell contributed valuable comments to the design of this work. Bonnie John and Virginia "Gin" Peck generously provided us with the Browser model and associated data and worked with us during the development of the new analyses. This article has been improved by the comments of the special issue editors, Ellen Bass, Tim Croudace, Nigel Major, and three anonymous reviewers. Development of this methodology and work has benefited from comments from the Soar group and is partially a general product of the cognitive science community at Carnegie Mellon University.

Support. Funding for this work was provided by the Air Force Office of Scientific Research through fellowship to Frank E. Ritter and is currently supported by U.K. Joint Council Initiative in HCI and Cognitive Science Grant SPG 9018736 and by the Economic and Social Research Council's Centre for Research in Development, Instruction, and Training.

Authors' Present Addresses. Frank E. Ritter, Department of Psychology, University of Nottingham, Nottingham, NG7 2RD, England. E-mail: ritter@psyc.nott.ac.uk; Jill H. Larkin, Department of Psychology, Carnegie Mellon University, Pittsburgh, PA 15213. E-mail: jhl@andrew.cmu.edu.

HCI Editorial Record. First manuscript received April 6, 1993. Revisions received November 11, 1993 and February 15, 1994. Accepted by Penelope M. Sanderson and Carolanne Fisher. Final manuscript received May 24, 1994. – *Editor*

REFERENCES

ACM Special Interest Group on Artificial Intelligence. (1989). Special issue on knowledge acquisition. *SIGART Newsletter,* 108.

ACM Special Interest Group on Artificial Intelligence. (1991). Special section on integrated cognitive architectures [Whole issue]. *SIGART Bulletin, 2*(4).

Anderson, J. R. (1993). *Rules of the mind.* Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Anderson, J. R., Farrell, R., & Sauers, R. (1981). Learning to program in Lisp. *Cognitive Science, 8,* 87–129.

Anzai, Y., & Simon, H. A. (1979). The theory of learning by doing. *Psychological Review, 86,* 124–140.

Bainbridge, L., & Sanderson, P. (in press). Verbal protocol analysis. In J. Wilson & E. Corlett (Eds.), *Evaluation of human work* (2nd ed.). London: Taylor & Francis.

Bates, D., Kademan, E., Ritter, F. E., & Smith, D. (1990, Fall). *S-mode for GNU Emacs.* (Rev. 1991, 1992; available from Statlib Software Archive, statlib@lib.stat.cmu.edu)

Baylor, G. W. (1971). *A treatise on the mind's eye: An empirical investigation of visual imagery.* PhD thesis, Carnegie Mellon University, Department of Psychology, Pittsburgh.

Becker, R. A., Chambers, J. M., & Wilks, A. R. (1988). *The new S language.* Pacific Grove, CA: Wadsworth & Brooks/Cole.

Bentler, P. M. (1980). Multivariate analysis with latent variables: Causal modeling. *Annual Review of Psychology, 31,* 419–456.

Bhaskar, R., & Simon, H. A. (1977). Problem solving in semantically rich domains: An example from engineering thermodynamics. *Cognitive Science, 1,* 193–215.

Bovair, S., Kieras, D. E., & Polson, P. G. (1990). The acquisition and performance of text-editing skill: A cognitive complexity analysis. *Human–Computer Interaction, 5,* 1–48.

Bree, D. S. (1968). The understanding process as seen in geometry theorems. *Dissertation Abstracts International, 30,* 1675A. (University Microfilms No. 69–18, 588)

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

Carley, K. (1988). Formalizing the social expert's knowledge. *Sociological Methods and Research, 17,* 165–232.

Casner, S., & Larkin, J. H. (1989). Cognitive efficiency considerations for good graphic design. *Proceedings of the 11th Annual Conference of the Cognitive Science Society,* 275–282. Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Dillard, J. F., Bhaskar, R., & Stephens, R. G. (1982). Using first-order cognitive analysis to understand problem solving behavior: An example from accounting. *Instructional Science, 11*(1), 71–92.

Ericsson, K. A., & Simon, H. A. (1993). *Protocol analysis: Verbal reports as data* (2nd ed.). Cambridge, MA: MIT Press.

Feldman, J. (1962). Computer simulation of cognitive processes. In H. Borko (Ed.), *Computer aapplications in the behavioral sciences* (pp. 336–359). Englewood Cliffs, NJ: Prentice-Hall.

Gascon, J. (1976). *Computerized protocol analysis of the behavior of children on a weight seriation task.* Doctoral dissertation, Université de Montréal, Departement de Psychologie.

Grant, D. A. (1962). Testing the null hypothesis and the strategy and tactics of investigating theoretical models. *Psychological Review, 69,* 54–61.

Gray, W. D., John, B. E., & Atwood, M. E. (1993). Project Ernestine: Validating a GOMS analysis for predicting and explaning real-world task performance. *Human–Computer Interaction, 8,* 237–309.

Greeno, J. G. (1978). A study of problem solving. In R. Glaser (Ed.), *Advances in instructional psychology* (Vol. 1). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Heise, D. R. (1991). Event structure analysis: A qualitative model of quantitative research. In N. G. Fielding & R. M. Lee (Eds.), *Using computers in qualitative research* (pp. 136–163). London: Sage.

Howes, A., & Young, R. M. (1991) Predicting the learnability of task-action mappings. *Proceedings of CHI '91 Conference on Human Factors in Computer Systems,* 113–118. New York: ACM.

Hucka, M. (1994). *Soar development environment.* Ann Arbor: University of Michigan, Soar Group.

John, B. E. (1994). *A database for analyzing "think-aloud" protocols and their associated cognitive models* (Technical Report CMU-HCII-94-101). Pittsburgh: Carnegie Mellon University, Human–Computer Interaction Institute, Pittsburgh.

Johnson, P. E., Durán, A. S., Hassebrock, F., Moller, J., Prietula, M., Feltovich, P. J., & Swanson, D. B. (1981). Expertise and error in diagnostic reasoning. *Cognitive Science, 5,* 235–283.

Kaplan, C. (1987). *Computer simulation: Separating fact from fiction* (CIP Report 498). Pittsburgh: Carnegie Mellon University, Department of Psychology.

Kieras, D. E. (1982). A model of reader strategy for abstracting main ideas from simple technical prose. *Text, 2*(1–3), 47–81.

Koedinger, K. R. (1991). *Tutoring concepts, percepts, and rules in geometry problem solving.* Doctoral dissertation, Carnegie Mellon University, Department of Psychology, Pittsburgh.

Koedinger, K. R., & Anderson, J. R. (1990). Abstract planning and perceptual chunks: Elements of expertise in geometry. *Cognitive Science, 14,* 511–550.

Kowalski, B., & VanLehn, K. (1988). Cirrus: Inducing subjective models from protocol data. *Proceedings of the Tenth Annual Conference of the Cognitive Science Society,* 623–629. Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Laird, J. E., Congdon, C. B., Altmann, E., & Swedlow, K. (1990). *Soar user's manual: Version 5.2* (Technical Report CSE-TR-72-90). Ann Arbor: University of Michigan, Department of Electrical Engineering and Computer Science.

Langley, P., & Ohlsson, S. (1984). Automated cognitive modeling. *Proceedings of AAAI–84,* 193–197. Los Altos, CA: Morgan Kaufman.

Larkin, J. H., McDermott, J., Simon, D. P., & Simon, H. A. (1980). Models of competence in solving physics problems. *Cognitive Science, 4,* 317–345.

Larkin, J. H., & Simon, H. A. (1987). Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science, 11,* 65–99.

Lehman, J. F., Newell, A., Newell, P., Altmann, E., Ritter, F., & McGinnis, T. (1994). *The Soar video.* Pittsburgh: Carnegie Mellon University, Soar Group.

Luger, G. F. (1981). Mathematical model building in the solution of mechanics problems: Human protocols and the MECHO trace. *Cognitive Science, 5,* 55–77.

MacWhinney, B. (1991). *The CHILDES Project: Tools for analyzing talk.* Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Miwa, K., & Simon, H. A. (1993). Production system modeling to represent individual differences: Tradeoff between simplicity and accuracy in simulation of behavior. *Prospects for Artificial Intelligence: Proceedings of AISB '93,* 158–167. Amsterdam: ISO.

Moran, T. P. (1973). *The symbolic imagery hypothesis: An empirical investigation of human behavior via a production system simulation of human behavior in a visualization task.* PhD thesis, Carnegie Mellon University, Department of Computer Science, Pittsburgh.

Motta, E., Eisenstadt, M., Pitman, K., & West, M. (1988). Support for knowledge acquisition in the Knowledge Engineer's Assistant (KEATS). *Expert Systems, 5,* 6–28.

Myers, B. A., Giuse, D. A., Dannenberg, R. B., Vander Zanden, V., Kosbie, D. S., Pervin, E., Mickish, A., & Marchal, P. (1990). Garnet: Comprehensive support for graphical, highly-interactive user interfaces. *IEEE Computer, 23*(11), 71–85.

Neuwirth, C. M., Kaufer, D. S., Chandhok, R., & Morris, J. H. (1990). Issues in the design of computer support for co-authoring and commenting. *Proceedings of the Conference on Computer-Supported Cooperative Work,* 183–195. New York: ACM.

Newell, A. (1977). On the analysis of human problem solving protocols. In P. J. Johnson-Laird & P. C. Wason (Eds.), *Thinking: Readings in cognitive science* (pp. 46–61). Bath, England: Pitman.

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

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

Nichols, S., & Ritter, F. E. (1994). *A theoretically motivated tool for automatically generating command aliases* (Technical Report 20). Nottingham, England: University of Nottingham, Department of Psychology, CREDIT. (To appear in *Proceedings of the Conference on Human Factors in Computing Systems,* 1995, in press)

Ohlsson, S. (1982). *Problem solving strategies in a spatial reasoning task* (Technical Report 340). Uppsala, Sweden: University of Uppsala, Department of Psychology.

Ohlsson, S. (1990). Trace analysis and spatial reasoning: An example of intensive cognitive diagnosis and its implications for testing. In N. Frederiksen, R. Glaser, A. Lesgold, & M. G. Shafto (Eds.), *Diagnostic monitoring of skill and knowledge acquisition* (pp. 251–296). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Olson, J. R., & Olson, G. M. (1990). The growth of cognitive modeling in human–computer interaction since GOMS. *Human–Computer Interaction, 5,* 221–265.

Payne, D., Cohen, M., & Pastore, R. (1992). Computer-based task representation: A methodology for improving system design. *Interacting With Computers, 4,* 267–288.

Peck, V. A., & John, B. E. (1992). The Browser model: A computational model of a highly interactive task. *Proceedings of the CHI '92 Conference on Human Factors in Computer Systems,* 165–172. New York: ACM.

Polk, T. A. (1992). *Verbal reasoning.* Doctoral dissertation, Carnegie Mellon University, School of Computer Science, Pittsburgh.

Polson, P. G., & Lewis, C. H. (1990) Theory-based design for easily learned interfaces. *Human–Computer Interaction, 5,* 191–220

Ritter, F. E. (1993). *TBPA: A methodology and software environment for testing process models' sequential predictions with protocols* (Technical Report CMU-CS-93-101).

Pittsburgh: Carnegie Mellon University, School of Computer Science, Soar Project.

Ritter, F. E. (1994). Testing the sequential predictions of a model that learns [Slides]. *Proceedings of the Thirteenth Soar Workshop.* Columbus: Ohio State University.

Ritter, F. E., Hucka, M., & McGinnis, T. F. (1992). *Soar-mode manual* (Technical Report CMU-CS-92-205). Pittsburgh: Carnegie Mellon University, School of Computer Science.

Ritter, F. E., & McGinnis, T. F. (1992). *Manual for SX: A graphical display and interface for Soar in X windows.* Pittsburgh: Carnegie Mellon University, School of Computer Science, Soar Project.

Sakoe, H., & Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing, 26*(1), 43–49.

Sanderson, P. M. (1993). Designing for simplicity of inference in observational studies of process control: ESDA and MacSHAPA. In E. Hollnagel & M. Lind (Eds.), *Proceedings of the Fourth European Conference on Cognitive Science Approaches to Process Control* (pp. 20–47). Copenhagen: Symbian Conference Service, Copenhagen Science Park.

Sanderson, P. M., & Fisher, C. A. (1994). Exploratory sequential data analysis: Foundations. *Human–Computer Interaction, 9,* 251–317. [Included in this Special Issue.]

Singley, M. K. (1987). *Developing models of skill acquisition in the context of intelligent tutoring systems.* Doctoral dissertation, Carnegie Mellon University, Department of Psychology, Pittsburgh.

Siochi, A. C., & Hix, D. (1991). A study of computer-supported user interface evaluation using maximal repeating pattern analysis. *Proceedings of the CHI '91 Conference on Human Factors in Computer Systems,* 301–305. New York: ACM.

Sleeman, D., Hirsh, H., Ellery, I., & Kim, I. (1990). Extending domain theories: Two cases studies in student modeling. *Machine Learning, 5,* 11–37.

Smith, J. B., Smith, D. K., & Kupstas, E. (1993). Automated protocol analysis. *Human–Computer Interaction, 8,* 101–145.

Stallman, R. M. (1984). EMACS: The extensible, customizable, self-documenting display editor. In D. R. Barstow, H. E. Shrobe, & E. Sandewall (Eds.), *Interactive programming environments* (pp. 300–325). New York: McGraw-Hill.

Stone, P. J., Dunphy, D. C., Smith, M. S., & Ogilvie, D. M., with associates. (1966). *The General Inquirer: A computer approach to content analysis.* Cambridge, MA: MIT Press.

VanLehn, K. (1989). Learning events in the acquisition of three skills. *Proceedings of the 11th Annual Conference of the Cognitive Science Society,* 434–441. Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

VanLehn, K. (1991) Rule acquisition events in the discovery of problem-solving strategies. *Cognitive Science, 15*(1), 1–47.

VanLehn, K., & Jones, R. M. (1993). Learning by explaining examples to oneself: A computational model. In S. Chipman & A. L. Meyrowitz (Eds.), *Foundations of knowledge acquisition: Cognitive models of complex learning* (pp. 25–82). Boston: Kluwer.

Waltz, D. L. (1987). Applications of the connection machine. *IEEE Computer, 20*(1), 85–97.

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

Waterman, D. A., & Newell, A. (1973). Pas-II: An interactive task-free version of an automatic protocol analysis system. *Proceedings of the Third International Joint Conference on Artificial Intelligence,* 431–445. Menlo Park, CA: Stanford Research Institute.

Young, R. M., & Whittington, J. E. (1990). Using a knowledge analysis to predict conceptual errors in text-editor usage. *Proceedings of the CHI '90 Conference on Human Factors in Computer Systems,* 91–97. New York: ACM.