John, B. E. (2003) Information processing and skilled behavior. In J. M. Carroll, (Ed.), *Toward a multidisciplinary science of human computer interaction.* Morgan Kaufman.

4
CHAPTER

Information Processing and Skilled Behavior

Bonnie E. John Carnegie-Mellon University

4.1 MOTIVATION FOR USING THE HUMAN INFORMATION PROCESSING THEORY IN HUMAN-COMPUTER INTERACTION[1]

Human performance can be enormously enhanced through the use of machines, but the interface connecting human and machine is subtle, and the success of the entire human-machine system turns on the design of this interface. Air traffic control systems, personal computers with word processors, portable digital appliances—all can make it possible for people to do new things, or through poor design of the human-machine interface, and can be rendered impractical or even dangerous. This lesson was learned early and often. For example, in recommending a program of research on human engineering for a national air-traffic control system, early researchers noted the following:

> The . . . disregard of physiological and sensory handicaps; or fundamental principles of perception; of basic patterns of motor coordination; of human limitations in the integration of complex responses, etc. has at times led to the production of mechanical monstrosities which tax the capabilities of human operators and hinder the integration of man and machine into a system designed for most effective accomplishment of designated tasks. (Fitts, 1951)

[1] Thanks to Stuart K. Card for his aid in placing human information processing in HCI in historical context.

S
R
L

Many of the early design issues revolved around perceptual and motor activities, and these are emphasized in Fitts' quotation. In psychological terms, these equated to *stimulus-response* and led to the study of stand-alone actions—the time to decode an aircraft display, or the errors made in selecting which aircraft fuel tank to shut off. To design for humans as a component of a larger system, it is useful to be able to describe both sides of the system in uniform terms. Treating the human as an information processor, albeit a simple stimulus-response controller, allowed the application of information theory and manual control theory to problems of display design, visual scanning, workload, aircraft instrument location, flight controls, air-traffic control, and industrial inspection, among others. Fitts' Law, which predicts the time for hand movements to a target, is an example of an information-processing theory of this era (see Chapter 3).

But the advent of computer systems emphasized the need for models beyond the stimulus-response-controller models. Models for analyzing human-computer interactions (HCI) needed to deal with sequential, integrated behavior rather than discrete actions. They also needed to deal with content of displays, not just their format. A promising set of developments in psychology and computer science was the information-processing models of Allen Newell and Herbert Simon (1961, 1972; Simon, 1969). These models were based on sequences of operations on symbols (as contrasted with the analog signal representations of many of the previous generation of models). The natural form of information-processing theories is a computer program, where a set of mechanisms is described locally, and where larger scale behavior is emergent from their interaction. The claim is not that all human behavior can be modeled in this manner, but that for tasks within their reach

> it becomes meaningful to try to represent in some detail a particular man at work on a particular task. Such a representation is no metaphor, but a precise symbolic model on the basis of which pertinent specific aspects of a man's problem solving behavior can be calculated. (Newell & Simon, 1972, p. 5)

Figure 4.1 is a generic representation of an information-processing system. At the center is some processing executive that acts on a recognize-act cycle. On each cycle, information available through the receptors and from internal memory is matched against a set of patterns, usually represented as a set of if-then rules called *productions*. This match triggers a set of actions (or *operators*) that can change the state of internal memory and/or change the external world through

S
R
L

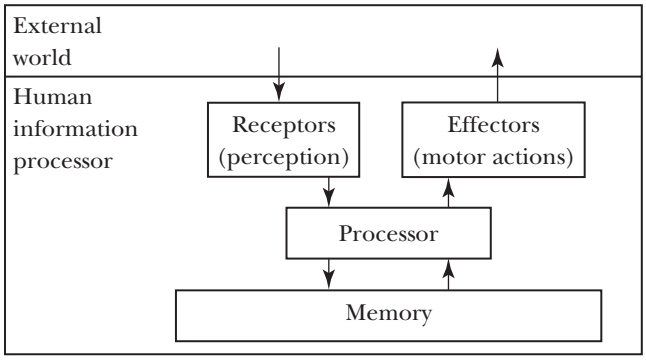


FIGURE 4.1 A generic human information-processing system.

the effectors. The cycle then repeats. For simplicity of use, some information processing models do not articulate this full structure, but it is implicit in their assumptions.

The need to apply human information processing to the design of computer interfaces led to a philosophy composed of task analysis, approximation, and calculation (Card, Moran, & Newell, 1983). These properties separate the models useful in engineering from some of the models used in general psychology. For engineering, it is important to have models that can make predictions from a technical analysis of a task without needing data fitted against an empirical study of users executing the task—zero-parameter models. This is necessary to allow the models to be used early in the design process. "Zero-parameter" models do have parameters, like typing rate or number of search targets, but these parameters can be determined from the task or previous studies in the psychological literature; they are not fit to data measured on the system and the task they are predicting. Note that, to be useful for engineering, the model parameters must be practical to supply at the time the model is to be used. Surprisingly, this requirement excludes some promising models from the psychological literature (Elkind et al, 1990).

Most science and engineering disciplines use approximation, both to make analysis more tractable and to simplify details that are not expected to have major effects on the result. Approximation, though, is a double-edged sword. It can be the key to making a model tractable, abstracting away from irrelevant inputs and approximating the information processing being used. On the other hand, it can also be accused of "looking under the lamp post" because, in making a

S
R
L

tractable prediction, relevant variables have been ignored in the name of approximation. In fact, a number of HCI techniques subsequent to the information-processing approach have reacted specifically against approximation, giving up the power of quantitative prediction in favor of rich qualitative information unfiltered by what is tractable to the model (e.g., Chapter 13). Indeed, many models in the information-processing vein probably did abstract away too much, assuming unrealistically, for instance, that all information available on a cluttered screen was instantaneously and reliably perceived and comprehended by the model (e.g., John & Vera's [1992] initial model of an expert playing a video game made that assumption, but a later model of a novice game-player by Bauer & John [1995] did not). Analysts need to carefully weigh the pros and cons before approximating away from details of the task environment and artifacts. The evaluation of these models is not how much they approximate or whether they are statistically different from the eventual behavior, but whether they predict the behavior to within a particular desired range and whether the mechanisms of the model are insightful for that behavior.

The original examples of human information processing models included problem-solving tasks and the commission of errors (e.g., crypto-arithmetic, Newell & Simon, 1972), but these were more descriptive than predictive, were difficult to build, and could not easily be used as design tools. Card, Moran, and Newell (1983) introduced the Model Human Processor (MHP), which provided a framework for expressing zero-parameter models, and which could successfully predict short tasks, like matching a symbol to memory, or isolated tasks, like determining the fastest that someone would be able to type on several different keyboards. But the MHP was not fully operationalized in a computer program, and thus predicting complex emergent behavior was beyond its scope; newer computational cognitive architectures replaced it in the ensuing decades and will be discussed in subsequent sections.

To produce models of human behavior whose zero-parameter quantitative predictions could be useful in system design, Card, Moran, and Newell concentrated on the common situation of skilled users performing tasks within their area of skill, approximating actual performance by error-free performance (Card et al., 1980a, 1980b; 1983). These models are called GOMS (Goals, Operators, Methods, and Selection rules) models, and they represent an important aspect of HCI design. Many systems are designed with the belief that people will become skilled in their use and will want efficient methods for accomplishing routine tasks; GOMS can predict the impact of design decisions on this important measure of success. Because this is an important aspect of design, because these models have proved successful at predicting performance, and because such modeling has received extensive attention in HCI, this chapter will focus on

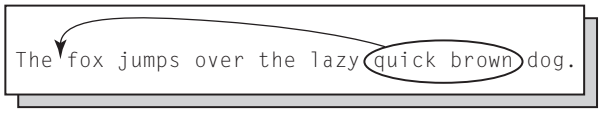


FIGURE 4.2 Example of a text-editing task.

GOMS modeling in examples and in the case study. It will return to the more general notion of human information processing when discussing the theoretical underpinnings, current status, and future work.

4.2 OVERVIEW OF GOMS

To briefly illustrate a GOMS model, consider the text-editing example in Figure 4.2. The figure shows a sentence that has been marked up for editing; the words "quick brown" must be moved from their current location to earlier in the sentence. We can use this example (which we will refer to throughout as the *fox task*) to make the concepts in GOMS concrete. GOMS is a method for describing a task and the user's knowledge of how to perform the task in terms of *G*oals, *O*perators, *M*ethods, and *S*election rules. Goals are simply the user's goals, as defined in layman's language. What does he or she want to accomplish by using the software? In the next day, the next few minutes, the next few seconds? In the case of our example, the highest-level goal is to edit the text, or, more specifically, to move "quick brown" to before "fox".

Operators are the actions that the software allows the user to take. With the original command-line interfaces, an operator was a command and its parameters, typed on a keyboard. Today, with graphic user interfaces, operators are just as likely to be menu selections, button presses, or direct-manipulation actions. Emerging technologies have operators that are gestures, spoken commands, and eye movements. Operators can actually be defined at many different levels of abstraction, but most GOMS models define them at a concrete level, like button presses and menu selections. The most common operators for analysis are at what is called the "keystroke level" and include moving the cursor, clicking the mouse button, and keying in information. In this example, the keystroke-level operators may involve both the keyboard (e.g., type CTRL-X to cut text) and the mouse (e.g., move mouse to word, double-click mouse button, move to menu, etc.).

S
R
L

Methods are well-learned sequences of subgoals and operators that can accomplish a goal. For our example, the following method exists in our word processor:

1. Highlight the text to be moved,
2. Type CTRL-X,
3. Move the cursor to the desired location,
4. Click the mouse button,
5. Type CTRL-V.

Notice that this expression of the method is a combination of operators (actions not readily decomposable) and subgoals (easily decomposable to more primitive actions). The operators already at the keystroke level include typing keyboard shortcuts, moving the cursor, and clicking the mouse button. Highlighting the text, however, is a subgoal that can be accomplished through several methods. For instance, the user could choose to mouse-down before "quick" and drag to the end of "brown" to highlight the text. Alternatively, the user could double-click on "quick" and then shift-click on "brown". Other methods for the entire task also exist, such as using the menus to invoke cut and paste through menu items, or deleting the text and retyping it in before "fox", or even deleting the whole sentence and retyping it correctly.

If there is more than one method to accomplish the same goal, then selection rules, the last component of the GOMS model, are required. Selection rules are the personal rules that users follow in deciding what method to use in a particular circumstance. For instance, in the example, if the text to be moved is one or two characters long, then a specific person may delete the text in the wrong location and retype it in the right location. If the text is three or more characters long, that person may cut and paste using keyboard shortcuts. Thus, that person's personal selection rule depends on the length of the word. Another user may have a different selection rule that depends on a different length of word, or on whether she can remember the keyboard shortcuts, or on other features of the task situation.

Thus, the fox task in Figure 4.2 has the elements in its GOMS model shown in Table 4.1. GOMS analysis applies to situations in which users will be expected to perform tasks that they have already mastered. In the psychology literature, this is called having a *cognitive skill*, that is, users are not problem solving and are not hunting around for what they need to do next. They know what to do in this task situation, and all they have to do is act. There are many different types of cognitive skill in human-computer interaction. For instance, there are many

Top-level goal	Edit manuscript, or more specifically, move "quick brown" to before "fox"
Subgoal	Highlight text
Operators	Move-mouse Click mouse button Type characters (keyboard shortcuts)
Methods	For the editing goal: 1. Delete-word-and-retype (*retype* method) 2. Cut-and-paste-using-keyboard-shortcuts (*shortcuts* method) 3. Cut-and-paste-using-menus (*menus* method) For the highlighting subgoal: 1. Drag-across text (*dragging* method) 2. Double-click first; shift-click last (*all-clicking* method)
Selection rules	For the editing goal: If text to be moved is one or two characters long, use *retype* method Else, if remember shortcuts, use *shortcuts* method Else, use *menus* method. For the highlighting subgoal: If text to be moved is not whole words, use *dragging* method Else, use *all-clicking* method

TABLE 4.1. Possible GOMS elements in the fox task.

single-user applications where the user tells the system what to do, then the system does it and tells the user what it has done. This is a user-paced, passive system, and GOMS has been shown to work very well for this situation GOMS has been applied to software such as text editors (Card, Moran & Newell, 1980a, 1983), spreadsheets (Lerch, Mantei, & Olson, 1989), information browsers (Peck & John, 1992), operating systems (Card, Moran, & Newell, 1983), ergonomic design systems (Gong & Kieras, 1994), CAD systems (Bhavanani & John, 1997), map digitizers (Haunold & Kuhn, 1994), flight-management computers in commercial airplanes (Irving, Polson, & Irving, 1994), oscilloscopes (Lee, Polson, & Bailey, 1989), programmable television sets (Elkerton 1993), and Web pages (John, 1995).

GOMS has also been shown to be valid in single-user, active systems, where the system changes in unexpected ways or other people participate in accomplishing the task. There are GOMS models, for instance, of radar monitoring (Rosenblatt & Vera, 1995) and of video games (John & Vera, 1992), where the system throws new situations at the user at a maniacal pace, and there are GOMS models of telephone operators interacting with customers (Gray, John, & Atwood, 1993). The knowledge gathered by a GOMS analysis is sufficient to predict what a skilled person will do in these seemingly unpredictable situations.

GOMS can be used both quantitatively and qualitatively. Quantitatively, it gives good predictions of absolute performance time and relative learning time. It can therefore be used to help in a purchasing decision (see the case study in Section 4.5), or to see if a proposed design meets quantitative performance requirements, or to compare one system's training time to another.

Qualitatively, GOMS can be used to design training programs, help systems, and the system itself. Because a GOMS model is a careful description of the knowledge needed to perform a given task, it describes the content of task-oriented documentation. You need only tell the new user what the goals are and how to recognize them, what different methods could be used to achieve them, and when to use each method (selection rules). This approach has been shown to be an efficient way to organize help systems (Elkerton, Goldstein, & Palmiter, 1990; Elkerton & Palmiter, 1989; Elkerton & Palmiter, 1991), intelligent tutoring systems (Steinberg & Gitomer, 1993) and training programs (Irving, Polson & Irving, 1994) as well as user documentation (Gong & Elkerton, 1990). GOMS models can also be used qualitatively to redesign a system. When GOMS uncovers a frequent goal supported by a very inefficient method, then the design can be changed to include a more efficient method. If GOMS shows that some goals are not supported by any method at all, new methods can be added. GOMS may also reveal where similar goals are supported by inconsistent methods, a situation in which users are likely to have problems remembering what to do, and show how to make the methods consistent (Kieras, in Helander et al., 1997).

Since the 1980s, HCI researchers have very carefully tested and retested the predictions of GOMS models, and they have reported these results in refereed conferences and journals. Many studies give rigorous laboratory verification of the predictions made from GOMS models on a number of products. Several studies have used real-world data to verify performance-time predictions of GOMS models. There has also been work with realistic training situations that show the value of GOMS-inspired training programs and help systems (Irving, Polson & Irving, 1994; Gong & Elkerton, 1990) GOMS is one of the most validated methods in HCI, making it a trustworthy tool for user interaction (UI) practitioners.

4.3 SCIENTIFIC FOUNDATIONS UNDERLYING GOMS

The concepts associated with GOMS are a mixture of several types: task-analysis techniques from the human factors and system design literature, models of human performance on specific tasks, computational models of human cognitive

architecture, and loosely defined concepts about human cognition and information processing. Figure 4.3 displays the relationships among these ideas. The figure is a lattice; at the top is the general idea of task analysis, and at the bottom is a basic conceptual framework for HIP, the *stage model* (where information flows from the outside world through perception to memory, where it can be manipulated by cognition, and cognition then commands the motor system to act on the world). Thus the GOMS family consists of ideas for *analyzing and representing tasks in a way that is related to the stage model of human information processing*. This dependence on a psychological framework is the distinctive feature of the GOMS approach compared to many other concepts of task analysis in the human factors and system-design literature.

These fundamental assumptions—that both the task structure and the cognitive architecture are necessary to describe, and ultimately predict, people's behavior with computer systems—arose from a parable often referred to as "Simon's Ant."

> We watch an ant make his laborious way across a wind- and wave-molded beach. He moves ahead, angles to the right to ease his climb up a steep dunelet, detours around a pebble, stops for a moment to exchange information with a compatriot. Thus he makes his weaving, halting way back to his home. . . .
>
> He has a general sense of where home lies, but he cannot foresee all the obstacles between. He must adapt his course repeatedly to the difficulties he encounters and often detour uncrossable barriers. His horizons are very close, so that he deals with each obstacle as he comes to it; he probes for ways around or over it, without much thought for future obstacles. It is easy to trap him into deep detours.
>
> Viewed as a geometric figure, the ant's path is irregular, complex, hard to describe. But its complexity is really a complexity of the surface of the beach, not a complexity in the ant. On that same beach another small creature with a home at the same place as the ant might well follow a very similar path. . . .
>
> An ant, viewed as a behaving system, is quite simple. The apparent complexity of its behavior over time is largely a reflection of the complexity of the environment in which it finds itself. (Simon, 1969, 63–64, emphasis in original)

Simon asserts that another small creature might follow the same path home, but this applies only to a surface-walking creature; it is easy to imagine that a digging creature might take a subterranean path. The story of Simon's Ant shows

S
R
L

that, to predict the ant's path, it is not sufficient to understand the ant alone nor to map the beach alone, but one must understand both in relation to the other. Much psychological research during the next few decades after Simon wrote his parable concentrated on understanding the structure of and constraints on human information processing; that is, it focused on understanding the ant. On the other hand, task analysis, traditionally in the purview of human factors and industrial engineering, concentrated effort on developing techniques to represent the abstract structure inherent in tasks and task environments; that is, on mapping the beach. However, with the advent of widespread computer usage, the emphasis in the HCI branch of HIP research changed to understanding both the human *and* the environment, because, in essence, HCI is about *designing* the beach, not simply describing it. Some research paths embraced rich descriptions of the beach and, importantly, how the ant changes it through use (e.g., Chapter 13). GOMS also embodies Simon's insight, marrying task analysis (maps of the beach) to frameworks of human behavior (the workings of the ant).

Thus, in Figure 4.3, reading down from the top, the top layer consists of task analysis techniques, followed by explicit computational cognitive architectures, and, at the bottom, conceptual frameworks, which are informal statements about how humans can be modeled. As one reads down from the top, or up from the bottom, the ideas get more explicit and detailed; the middle contains approaches whose instantiations are running computer simulation models.

Three things should be noted about the diagram: (1) Because our primary purpose in this chapter is to discuss GOMS models that have been used and useful in HCI, Figure 4.3 emphasizes these techniques and the concepts directly related to them. (2) There are areas in the diagram that are not directly related to GOMS models, and these are indicated with italics. Thus, the diagram is certainly not exhaustive: many more nodes and arrows could be drawn to the literature. What is shown here is only what is central to our discussion of currently documented GOMS models. (3) The diagram shows only generic ideas and approaches, not specific instances of task modeling. Examples of specific instances of using these techniques will appear as needed throughout this chapter.

We will describe the entries in this lattice, starting with the conceptual frameworks, because they form the basis for all GOMS analyses; we will then work through the computational cognitive architectures up to the task-analytic approaches, which are the heart of the GOMS family.

4.3.1 Conceptual Frameworks

S
R
L

The *conceptual frameworks* are so named because they are informally stated assumptions about the structure of human cognition. The conceptual frameworks

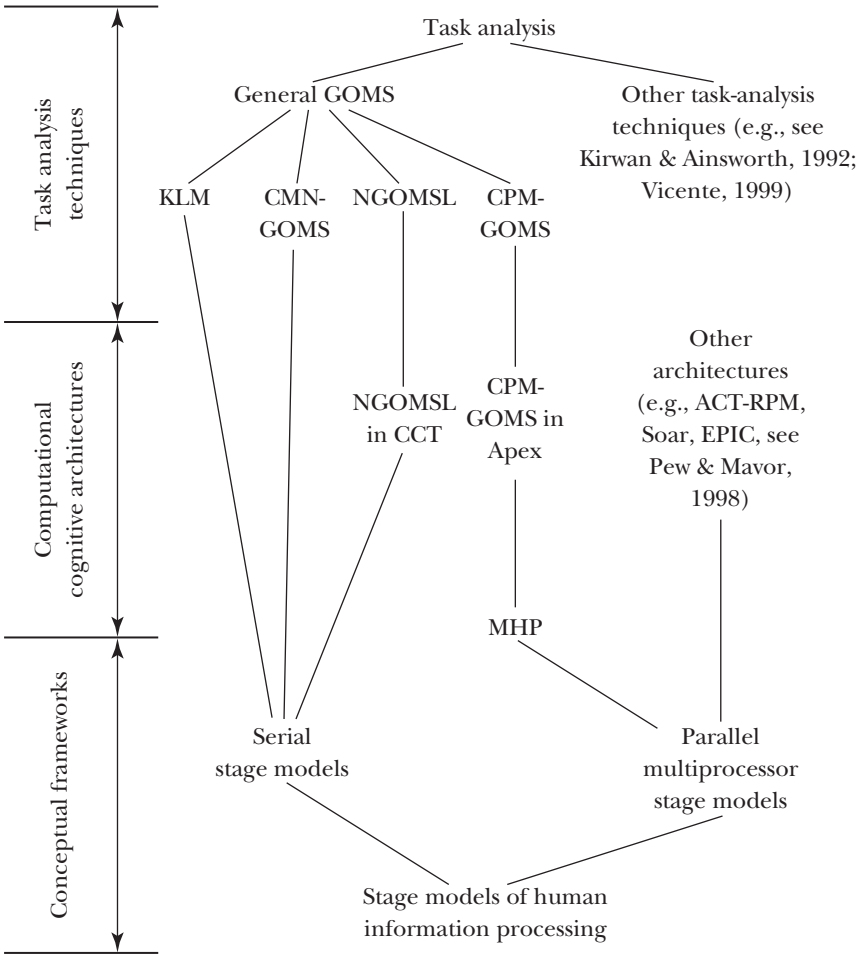


FIGURE 4.3 The GOMS family consists of task-analysis techniques that are related to models of human information processing.

shown are all based on a general assertion that human cognition and behavior is usefully analyzed in terms of stages. The conventional notion is that stimuli are first processed perceptually; the resulting information is passed to a central cognitive process that manipulates that information and eventually initiates some motor activity.

The cognitive process that manipulates information is often described as *search through a problem space*. A problem space is defined by a set of possible *states*, which include the information available to cognition internally (from memory)

or in the world (through perception). The search involves applying an *operator* to the current state to change it to a new state, and evaluating whether that new state is closer to a desired (goal) state. Operators can be both internal (e.g., mentally adding two numbers) or external (e.g., pulling down a menu to reveal its items). The difficulty of a problem can be assessed by the length and indirectness of the search. In one situation, many plausible operators might be applicable to each state and no knowledge to choose between them. This forces the user to explore, take false paths, and back up to old states. In another situation, there could be few applicable operators at any state (like in a wizard), or the user could have the knowledge of which operator to apply to move closer to the goal. The former situation is called *problem solving* and the latter *skilled behavior,* but in information-processing terms they are both on a continuum of behavior using the same mechanism of search through a problem space.

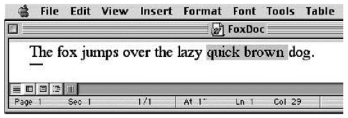
Figures 4.5 and 4.5 are two problem behavior graphs that show two different paths through a problem space for cutting "quick brown" in the example sentence. Figure 4.4 reflects the behavior of a person who does not know which menu contains the **Cut** command and must search until he or she finds it. This path contains one dead-end state and a back up to a previous state before finding the desired command. This behavior is typical of problem solving. Figure 4.5 reflects the behavior of a person who knows that the **Cut** command resides in the **Edit** menu, and search becomes trivial; this is characteristic of the skilled behavior that GOMS can model.

The stage model of HIP breaks out into two more specific forms. One is that the stages are performed serially, which is sufficient for describing laboratory experiments examining the single-action tasks mentioned in the Motivation section (e.g., decoding an aircraft display or switching off a fuel tank, in isolation from other tasks). The other is that the stages can be performed in parallel to some extent, since different kinds of processing are handled by separate mechanisms, or *processors.* The assumption of parallel operations seems necessary to understand certain types of human behavior, such as transcription typing a long paragraph where the eyes seem to stay several words ahead of the hands, as opposed to typing a single word as it is flashed on a screen.

The Model Human Processor (MHP) that by Card, Moran, and Newell (1983) introduced to HCI is a parallel architecture. Perceptual, cognitive, and motor processing are done by separate processing mechanisms, each with their own distinctive types and timing of activities, and with associated principles of operation. Card, Moran, and Newell's important insight was that the empirical human cognition and performance literature could be used to motivate and justify an *engineering model* of human information processing that could be used to predict performance in HCI situations.

S
R
L

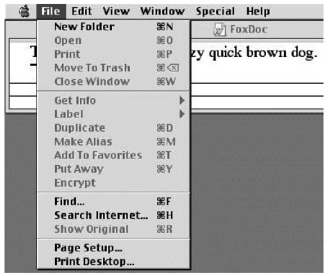
State 1:
Text to be cut is highlighted.
Many operators are applicable, e.g., one for pulling down each menu.
No knowledge about which operators to apply, so the first menu is chosen arbitrarily.



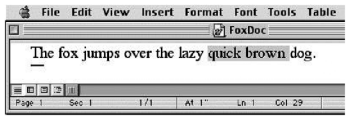
Apply Operator:
Pull down **File** menu,
Search for **Cut**.

Failure:
Back up to State1

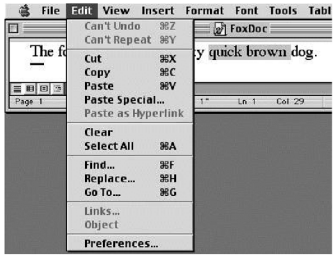
State 2:
File menu is visible.
Cut is not in this menu.



State1 transforms to State 1',
which now includes the knowledge that **Cut** is not in the **File** menu.
Many operators are still applicable and the second menu is chosen arbitrarily

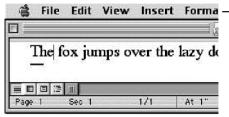


Apply Operator:
Pull down **Edit** menu,
Search for **Cut**.



State 3:
Edit menu is visible.
Cut is in this menu.
The knowledge to select the **Cut** item is applicable, so the task can continue.

Apply Operator:
Select the **Cut** menu item.



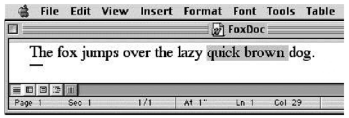
State 4:
Text to be cut has been cut successfully.

The task continues...

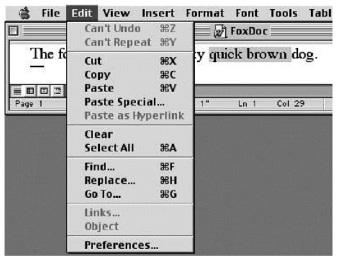
FIGURE 4.4 The problem behavior graph showing search through a problem space for cutting "quick brown" when the person does not know which menu contains the **Cut** command.

L R S

State 1:
Text to be cut is highlighted.
Many operators are applicable, e.g., one for pulling down each menu.
This time, the user has knowledge about which operator to apply, so he chooses to pull down the **Edit** menu.

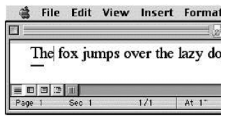


Apply Operator:
Pull down **Edit** menu,
Search for **Cut**.



State 3:
Edit menu is visible.
Cut is in this menu.
The knowledge to select the **Cut** item is applicable, so the task can continue.

Apply Operator:
Select the **Cut** menu item.



State 4:
Text to be cut has been cut successfully.

The task continues...

FIGURE 4.5 The problem behavior graph showing trivial search through a problem space for cutting "quick brown" when the person does know that the **Edit** menu contains the **Cut** command.

Although the MHP is inherently parallel, only 1 of the 19 examples of reasoning from the MHP presented in Card, Moran, and Newell (1983, Example 7) depend on this fact. The parallel operation of the MHP was made clear in the TYPIST model of transcription typing (John, 1988, 1996;), in which the processors could operate in a "pipeline" mode, with information moving through perceptual, cognitive, and motor stages continuously. This model accounted for important properties of skilled typing performance and shows that parallelism can greatly influence the structure and performance of a task.

Although Card, Moran, and Newell provide many examples of how the MHP can be applied to predict performance in some well-understood simple task situations similar to the experimental paradigms in the human performance literature, and they provide simple real-world analogs of these tasks, they did not provide an explicit method for applying the MHP to complex, realistic tasks (the CPM-GOMS methodology to be discussed later provides this explication). In Figure 4.3, the MHP is shown at the border between conceptual architectures and computational cognitive architectures because, while it is more specified than the simple stage concepts, it is not as fully explicit and computationally represented as the ideas in the next level up in the diagram.

4.3.2 Computational Cognitive Architectures

The level called *computational cognitive architectures* in Figure 4.3 lists proposals for how to represent human information processing in terms explicit enough to run as a computer program (such architectures are also called *unified theories of cognition;* Newell, 1990). Representation as a computer simulation is a preferred research tactic in cognitive psychology, based on the assumption that a computational model has "empirical content"—that is, a properly constructed and interpreted model can make predictions of human performance, and these predictions can be empirically confirmed. There are several such architectures under development in cognitive psychology and artificial intelligence, several of which have been applied to topics in HCI, such as ACT-RPM and its predecessors (Anderson, 1976, 1983, 1993; Byrne & Anderson, 2001), Construction-Integration (Kintsch, 1988, 1992), Soar (Newell, 1990), and EPIC (Kieras & Meyer, 1994; Meyer & Kieras, 1994). Each of these architectures makes different assumptions about how cognitive processes such as working memory management, flow of control, learning, and problem solving are handled by the architecture, and testing the empirical content of these assumptions is an active area of psychological research. In principle, all of these architectures could be used to implement a particular GOMS task analysis in a computational model (e.g., John

& Vera, 1992, and Peck & John, 1992, implemented GOMS models in Soar; Gray & Sabnani, 1994, implemented a GOMS model in ACT-R).

However, two computational architectures have a more direct relationship to GOMS and deserve special mention. Cognitive complexity theory (CCT) (Bovair, Kieras, & Polson, 1988, 1990; Kieras & Polson, 1985), a production-rule architecture based on the serial stage model, has been used as the basis for a specific GOMS technique, NGOMSL, which incorporates CCT's assumptions about working memory management, flow of control, and other architectural mechanisms. For brevity, CCT and NGOMSL will not be discussed further, because their mechanisms and contributions are well represented in other HCI literature (e.g. Kieras, in Helander et al., 1997). Apex (Freed, 1998) is an architecture for modeling human behavior in complex dynamic environments such as piloting a Boeing 757 or in air-traffic control; its foundation is the class of AI architectures called *reactive planners* (Firby, 1989). It combines reactive planning with the concept of limited resources (e.g., cognition, right-hand, left-hand, visual perception, etc.) to model human performance. Its concepts and programming environment have been used to build CPM-GOMS models (Section 4.4.3), where the architectural concepts dictate how low-level HCI skills (e.g., typing or using a mouse) can be interleaved to maximize parallelism in skilled performance (John et al., 2002).

4.3.3 Task-Analysis Techniques

At the top of the GOMS family tree in Figure 4.3, under the overall node of *task analysis,* which includes many techniques that generally "map the beach" (e.g., hierarchical task analysis, link analysis, operational sequence diagrams, timeline analysis in Kirwan & Ainsworth, 1992; Vicente's cognitive work analysis, 1999). The node labeled *general GOMS* represents the concept that it is useful to analyze knowledge of how to do a task in terms of goals, operators, methods, and selection rules. Thus, it is a form of task analysis that describes the procedural, "how-to-do-it" knowledge involved in a task. The result of a GOMS-based task analysis will be some form of description of the components of the goals, operators, methods, and selection rules.

There are three critical restrictions on the kinds of tasks that GOMS models can be used for. The first is that the task in question must be usefully analyzed in terms of the "how to do it" or *procedural* knowledge required rather than other aspects of knowledge about the system, like mental simulations of an internalized device model, or analogical reasoning (see Kieras & Polson, 1985, for more discussion). The italicized area to the right under *task analysis* represents other

S
R
L

existing and potential approaches to task analysis that capture other forms of task knowledge. For example, work on electronics troubleshooting (see Gott, 1988) incorporates the person's knowledge of electronic components and the structure and function of the system under investigation, in addition to various kinds of procedural knowledge. Work in analogical reasoning has been applied to understanding consistency in operating systems (Rieman, Lewis, Young, & Polson, 1994).

The second restriction is that the GOMS family can represent only *skilled behavior,* which consists of procedural knowledge that may originally derive from problem-solving activity, or from instruction, but with practice has taken the form of a routinely invocable sequence of activities that accomplishes the goals (see Card et al., 1983, Chap. 11). That is, the search through the problem space to achieve a goal has become trivial. At each state, there are only a few operators already known to move closer to the goal state, and selection between them has also been well learned. Of course, users often engage in problem solving, exploration, and other nonroutine activities while using a computer, and other cognitive modeling approaches and task analysis techniques can be used to investigate these behaviors (e.g., the Cognitive Walkthrough technique [Wharton, Rieman, Lewis & Polson, 1994] applies to exploratory behavior by novice users).

We emphasize, however, that most tasks have some element of skilled behavior. Composing a research paper requires the skill of text editing, charting data requires the skill of entering information into spreadsheets, architectural design with a computer-aided design (CAD) system requires routine window manipulation, and so on. Even if the primary task is creative or otherwise not routine, those aspects of the task that are routine are amenable to analysis with GOMS techniques. Applying GOMS to improve the routine aspects of a complex task will reduce the effort necessary to master and perform those routine aspects, getting them "out of the way" of the primary creative task.

The third restriction is that, in all GOMS analysis techniques, the designer or analyst must *start* with a list of top-level tasks or user goals. GOMS analyses and methods do not provide this list; it must come from sources external to GOMS; Typically, this list of goals can be obtained from other task-analysis approaches (e.g., see Diaper, 1989), such as interviews with potential users, observations of users of similar or existing systems, or, in the worst case, simple intuitions on the part of the analyst. Once this list is assembled, GOMS analyses can help guide the design of the system so that the user can accomplish the given tasks in an efficient and learnable way. However, except for possibly stimulating the analyst's intuitions, the subsequent GOMS analysis will not identify any new top-level user goals or tasks that the analyst overlooked, nor will it correct a misformulation of the user goals.

The next level down in the diagram in Figure 4.3 consists of specific proposals for how to carry out a task analysis within a GOMS orientation. It is at this level that the differences appear between different versions of GOMS analyses. Note that the general GOMS concept merely asserts that it is useful to analyze a task in terms of the user's goals, methods, operators, and selection rules. It does not specify any particular technique for doing such an analysis. A particular technique requires (1) more specific definitions of the GOMS components, especially the operators, and (2) guidance and a procedure for constructing the methods in terms of these more specific definitions. These additional details will be presented for three types of GOMS models in the next section.

4.4 DETAILED DESCRIPTION OF GOMS MODELS

GOMS can provide different types of information for the design of an interactive system. John and Kieras (1996a) review its contributions to assessing the coverage and consistency of functionality, and its predictions of operator sequences, execution time, training time, and error-recovery support. The strength of GOMS is in its quantitative predictions of performance time and its qualitative explanations for those predictions. Therefore, this chapter will demonstrate those aspects of GOMS and direct the reader to John and Kieras (1996a) for the others. In addition, there are many different versions of GOMS. Four of them, detailed and related to each other by John and Kieras (1996a, 1996b), appear in Figure 4.3, but others exist with varying degrees of penetration in the HCI community. For instance, QGOMS (Beard, Smith, & Denelsbeck, 1996) is a variant embodied in a graphical tool that substitutes probabilities for selecting a method for the symbolic semantics favored by Card, Moran, and Newell. This section will use three versions of GOMS: the Keystroke-Level Model (KLM), traditional GOMS introduced by Card, Moran, and Newell (CMN-GOMS), and a parallel version, CPM-GOMS, to analyze the expected performance time on the text-editing task shown in Figure 4.2. (For a similar analysis that includes NGOMSL as well as the three detailed here, see John & Kieras, 1996a).

4.4.1 KLM

The Keystroke-Level Model (KLM) is the simplest GOMS technique; it was originally described in Card, Moran, and Newell (1980b) and later in Card, Moran, and Newell (1983, Chap. 8). The KLM makes several simplifying assumptions

that make it a restricted version of GOMS. In particular, the analyst must specify the method used to accomplish the particular task of interest, which typically entails choosing specific task instances. Other GOMS techniques discussed later predict the method given the task situation and the knowledge of methods and selection rules, but the KLM does not. Furthermore, the specified method is limited to containing only a small set of preestablished keystroke-level primitive operators. Given the task and the method, the KLM uses duration estimates of these keystroke-level operators to predict the time a skilled user will need to execute the task.

The original KLM included six types of operators: K to press a key or button; P to point with a mouse to a target on a display; H to home hands on the keyboard or other device; D to draw a line segment on a grid; M to mentally prepare to do an action or a closely related series of primitive actions; and R to represent the system-response time during which the user has to wait for the system. Each of these operators has an estimate of execution time, either a single value, a parameterized estimate (e.g., K is dependent on typing speed and whether a key or mouse button click, press, or release is involved), or a simple approximating function (e.g., Fitts' Law estimates for P). The KLM also includes a set of five heuristic rules for placing mental operators to account for mental preparation time.

Subsequent research has refined these six primitive operators, improving the time estimates or differentiating between different types of mental operations (Olson & Olson, 1990), and practitioners often tailor these operators to suit their particular user group and interface requirements (e.g., Haunold & Kuhn, 1994). In addition, the heuristics for placing mental operators have been refined for specific types of subtasks (e.g., for making a fixed series of menu choices, see Lane, Napier, Batsell, & Naman, 1993). In particular, since the original heuristic rules were created primarily for command-based interfaces, they need to be updated for direct manipulation interfaces (John & Kieras, 1996a).

Figure 4.6 provides a sample KLM with computation of execution time for the fox task shown in Figure 4.2, using the operator times supplied in Card, Moran, and Newell (1983, p. 264). Quantitatively, the KLM makes the prediction that this task will take about 15 seconds. Qualitatively, the analyst can use the model to highlight several ideas. The subgoal structure is not explicit in the KLM itself, but an analyst can see it in the model (as annotated) and use it to look for recurring subprocedures that might be combined or shortened. For instance, the analyst has made an annotation to consider a MOVE command instead of CUT and PASTE. A KLM for MOVE would show what time savings this would provide, which could then be weighed against other considerations like users' prior knowledge or other functionality (e.g. the ability to paste multiple copies).

Description	Operator	Duration (sec)
Mentally prepare by Heuristic Rule 0	M	1.35
Move cursor to "quick"	P	1.10
(no M by Heuristic Rule 1)		
Double-click mouse button	K	0.40
		1.10
		0.40
		1.35
		1.10
(no M by Heuristic Rule 1)		
Click mouse button	K	0.20
Move cursor to Cut menu item	P	1.10
(no M by Heuristic Rule 1)		
Click mouse button	K	0.20
Mentally prepare by Heuristic Rule 0	M	1.35
Move cursor to before "fox"	P	1.10
(no M by Heuristic Rule 1)		
Click mouse button	K	0.20
Mentally prepare by Heuristic Rule 0	M	1.35
Move cursor to Edit menu	P	1.10
(no M by Heuristic Rule 1)		
Click mouse button	K	0.20
Move cursor to Paste menu item	P	1.10
(no M by Heuristic Rule 1)		
Click mouse button	K	0.20
TOTAL PREDICTED TIME		14.90

This Figure 4.6 is incomplete; it does not show the hand markings a designer might put on the KLM.
See Figure 4.6 on the next page for a better representation of a KLM as it could be used in real work.

FIGURE 4.6 A keystroke-level model for moving the text in Figure 4.2 using the CUT-AND-PASTE-USING-MENUS method for the entire task, and the ALL-CLICKING method for highlighting the text to be moved.

In terms of underlying architecture, KLM does not need a computational representation because the methods are supplied by the analyst and are expressed as a sequence of operators; all the information-processing activity is assumed to be contained in the primitive operators, including internal cognitive actions, which are subsumed by black-box mental operators. Thus the underlying conceptual framework is simply the serial stage model.

S
R
L

Description	Operator	Duration (sec)
Mentally prepare by Heuristic Rule 0	M	1.35
Move cursor to "quick"	P	1.10
(no M by Heuristic Rule 1)		
Double-click mouse button	K	0.40
Move cursor to "brown"	P	1.10
(no M by Heuristic Rule 1)		
Shift-click mouse button	K	0.40
Mentally prepare by Heuristic Rule 0	M	1.35
Move cursor to Edit menu	P	1.10
(no M by Heuristic Rule 1)		
Click mouse button	K	0.20
Move cursor to Cut menu item	P	1.10
(no M by Heuristic Rule 1)		
Click mouse button	K	0.20
Mentally prepare by Heuristic Rule 0	M	1.35
Move cursor to before "fox"	P	1.10
(no M by Heuristic Rule 1)		
Click mouse button	K	0.20
Mentally prepare by Heuristic Rule 0	M	1.35
Move cursor to Edit menu	P	1.10
(no M by Heuristic Rule 1)		
Click mouse button	K	0.20
Move cursor to Paste menu item	P	1.10
(no M by Heuristic Rule 1)		
Click mouse button	K	0.20
TOTAL PREDICTED TIME		14.90

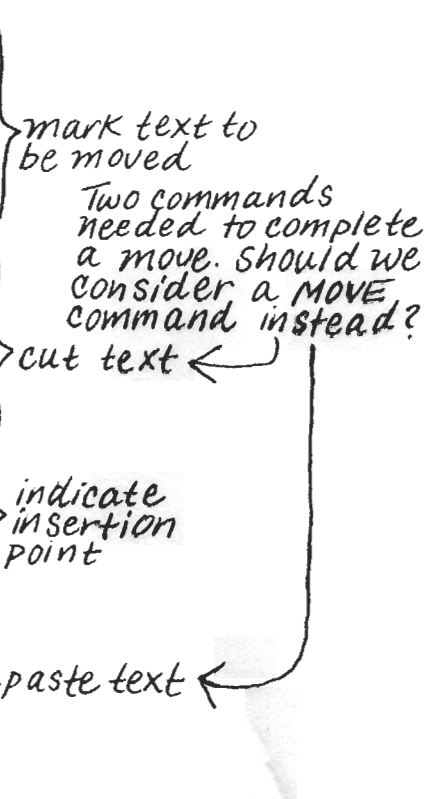


FIGURE 4.6 A Keystroke-Level Model for moving the text in Figure 4.2 using the CUT-AND-PASTE-USING-MENUS method for the entire task, and the ALL-CLICKING method for highlighting the text to be moved.

The primary advantage of KLM technique is that it allows a rapid estimate of execution time with an absolute minimum of theoretical and conceptual baggage. In this sense, it is the most practical of the GOMS techniques; it is the easiest to apply in actual interface design practice, and it is by far the simplest to explain and justify to computer software developers. This simple estimate of execution times can be used to compare design ideas on benchmark tasks, to do parametric evaluation to explore the space defined by important variables (e.g., the length of filenames in a command language), and to do sensitivity analyses on the assumptions made (e.g., user's typing speed) (Card et al., 1980b; Card et al., 1983).

4.4.2 CMN-GOMS

CMN-GOMS is the term used to refer to the form of GOMS model presented in Card, Moran, and Newell (Card et al., 1980a, 1980b; 1983, Chap. 5). CMN-GOMS is slightly more specified than general GOMS; there is a strict goal hierarchy, operators are executed in strict sequential order, and methods are represented in an informal pseudo-code-like notation that can include submethods and conditionals.

In the context of the Card, Moran, and Newell work, it would appear that the CMN-GOMS model is based on the MHP, but in fact Card, Moran, and Newell do not make a tight linkage. In particular, in presenting the CMN-GOMS formulation, they provide no description of how the MHP would represent and execute CMN-GOMS methods. Furthermore, the GOMS concept itself cannot be derived from the MHP as presented by Card, Moran, and Newell, but is only loosely based on two of the MHP principles of operation, the Rationality Principle and Problem Space Principle both well developed in the problem-solving theoretical literature (e.g., Newell & Simon, 1972; see Card et al., 1983, Chap. 11). Thus, Figure 4.3 shows that the CMN-GOMS model is based only on the serial stage model, not the MHP.

Card, Moran, and Newell do not describe the CMN-GOMS technique with an explicit "how-to" guide, but present nine models at different levels of detail that illustrate a breadth-first expansion of a goal hierarchy until the desired level of detail is attained. They report results in which such models predicted operator sequences and execution times for text-editing tasks, operating-systems tasks, and the routine aspects of computer-aided VLSI layout tasks. These examples are sufficiently detailed and extensive that independent researchers have been able to develop their own CMN-GOMS analyses (e.g., Lerch, Mantei, & Olson, 1989).

S
R
L

Figure 4.7 is an example of a CMN-GOMS model at the keystroke-level for the fox task shown in Figure 4.2, including details for the MOVE-TEXT goal. Moving is accomplished by first cutting the text and then pasting it. Cutting is accomplished by first highlighting the text, and then issuing the **Cut** command. As specified by a selection rule set, highlighting can be done in two different ways, depending on the nature of the text to be highlighted. Finally, pasting requires moving to the insertion point, and then issuing the **Paste** command.

Quantitatively, CMN-GOMS models predict the operator sequence and execution time. Qualitatively, CMN-GOMS models focus attention on methods to accomplish goals; similar methods are easy to see, unusually short or long methods jump out and can spur design ideas. In this example, the analyst has noticed that issuing commands via the menus will occur often and suggests keyboard shortcuts. In addition, the annotations indicate that this analyst has observed that the VERIFY operator explicitly records points of feedback to the user.

Comparing Figure 4.7 with Figure 4.6, the relationship between the CMN-GOMS technique and the KLM technique is evident. (Note that the expansion of the MOVE-TEXT goal in Figure 4.7 represents the same behavior as the KLM in Figure 4.6.) For instance, there is a one-to-one mapping between the physical operators in the CMN-GOMS model and the Ks and Ps in the KLM. The CMN-GOMS model has other operators at this level: VERIFY-LOCATION and VERIFY-HIGHLIGHT, which have no observable physical counterpart (they could be observed with an eye-tracker, but this instrument has only recently become available and affordable and is not often used in any but the most detailed HCI research). The KLM has no explicit goals or choices between goals, whereas the CMN-GOMS model represents these explicitly. Roughly, the VERIFY operators, goal hierarchies and selection rules of the CMN-GOMS model are represented as the M operators in the KLM. That is, operators such as VERIFY and goals and selections appear in the CMN-GOMS model in groups that roughly correspond to the placement of Ms in the KLM. This is only approximately the case, as the VERIFY operators sometimes occur in the middle of a group of physical operators, but the approximation is close.

A major difference between the KLM and the CMN-GOMS models is that CMN-GOMS is in program form, so therefore the analysis is general and executable. That is, any instance of the described class of tasks can be performed or simulated by following the steps in the model, which may take different paths depending on the specific task situation. Goals and method selection are predicted by the model given the task situation, and they need not be dictated by the analyst as they must for the KLM.

Given the task specified by the manuscript in Figure 4.2, this model would predict the trace of operators shown with the estimates of operator times in the

S
R
L

```
GOAL: EDIT-MANUSCRIPT
.       GOAL: EDIT-UNIT-TASK...repeat until no more unit tasks
.       .   GOAL: ACQUIRE UNIT-TASK...if task not remembered
.       .   .   GOAL: TURN-PAGE...if at end of manuscript page
.       .   .   GOAL: GET-FROM-MANUSCRIPT
.       .   GOAL: EXECUTE-UNIT-TASK...if a unit task was found
.       .   .   GOAL: MODIFY-TEXT
.       .   .   . [select: GOAL: MOVE-TEXT*...if text is to be moved
.       .   .   .          GOAL: DELETE-PHRASE...if a phrase is to be deleted
.       .   .   .          GOAL: INSERT-WORD]...if a word is to be inserted
.       .   .   . VERIFY-EDIT                                          1.35

* Expansion of MOVE-TEXT goal
GOAL: MOVE-TEXT
.       GOAL: CUT-TEXT
.       .   GOAL: HIGHLIGHT-TEXT
.       .   .   [select**: GOAL: HIGHLIGHT-PHRASE-COMPOSED-OF-WORDS
.       .   .              . MOVE-CURSOR-TO-FIRST-WORD                 1.10
.       .   .              . DOUBLE-CLICK-MOUSE-BUTTON                 0.40
.       .   .              . MOVE-CURSOR-TO-LAST-WORD                  1.10
.       .   .              . SHIFT-CLICK-MOUSE-BUTTON                  0.40
.       .   .              . VERIFY-HIGHLIGHT                          1.35
.       .   .              GOAL: HIGHLIGHT-ARBITRARY-TEXT
.       .   .              . MOVE-CURSOR-TO-BEGINNING-OF-TEXT
.       .   .              . PRESS-MOUSE-BUTTON
.       .   .              . MOVE-CURSOR-TO-END-OF-TEXT
.       .   .              . RELEASE-CLICK-MOUSE-BUTTON
.       .   .              . VERIFY-HIGHLIGHT]
.       .   GOAL: ISSUE-CUT-COMMAND
.       .   .   MOVE-CURSOR-TO-EDIT-MENU                               1.10
.       .   .   CLICK-MOUSE-BUTTON                                     0.20
.       .   .   MOVE-CURSOR-TO-CUT-ITEM                                1.10
.       .   .   VERIFY-HIGHLIGHT                                       1.35
.       .   .   CLICK-MOUSE-BUTTON                                     0.20
.       GOAL: PASTE-TEXT
.       .   GOAL: POSITION-CURSOR-AT-INSERTION-POINT
.       .   .   MOVE-CURSOR-TO-INSERTION-POINT                         1.10
.       .   .   CLICK-MOUSE-BUTTON                                     0.20
.       .   .   VERIFY-POSITION                                        1.35
.       .   GOAL: ISSUE-PASTE-COMMAND
.       .   .   MOVE-CURSOR-TO-EDIT-MENU                               1.10
.       .   .   CLICK-MOUSE-BUTTON                                     0.20
.       .   .   MOVE-CURSOR-TO-PASTE-ITEM                              1.10
.       .   .   VERIFY-HIGHLIGHT                                       1.35
.       .   .   CLICK-MOUSE-BUTTON                                     0.20
```

Is all this feedback in order?

Issuing commands will be used a lot! Can we shorten this procedure? Consider keyboard shortcuts.

TOTAL TIME PREDICTED (SEC) 16.25

**Selection Rule for GOAL: HIGHLIGHT-TEXT:

If the text to be highlighted is a phrase made up of words,
use the HIGHLIGHT-PHRASE-COMPOSED-OF-WORDS method,
else use the HIGHLIGHT-ARBITRARY-TEXT method.

FIGURE 4.7 Example of CMN-GOMS text-editing methods showing the task hierarchy and a selection rule.

far-right column. The estimates for the physical operators are identical to the ones in the KLM. The VERIFY-EDIT, VERIFY-HIGHLIGHT and VERIFY-POSITION operators are assigned 1.35 sec, the same value as the KLM's M operator because this is Card, Moran, and Newell's best estimate of mental time in the absence of other information.[2] Thus, the CMN-GOMS model produces nearly the same estimate for task completion as the KLM. The CMN-GOMS model has one more M-like operator in that it verifies the success of the entire task with a VERIFY-EDIT operator in the EXECUTE-UNIT-TASK goal. Notice that the CMN-GOMS technique assigns time only to operators, not to any "overhead" required to manipulate the goal hierarchy. In their results, Card, Moran, and Newell found that time predictions were as good with the simple assumption that only operators contributed time to the task as they were when goal manipulation also contributed time, but they suggested that at even more detailed levels of analysis such cognitive activity might become more important. Also notice that where the KLM puts Ms at the beginning of subprocedures, the CMN-GOMS model puts the mental time in VERIFY operators at the end of subprocedures. Since mental time is observable only as pauses between actions, it is difficult to distinguish between these two techniques empirically; only appeals to more detailed cognitive architectures can explain the distinction. Pragmatically, however, this difference is irrelevant in most design situations.

4.4.3 CPM-GOMS

CPM-GOMS is a version of GOMS based directly on the MHP, and thus on the parallel multiprocessor stage model of human information processing. It does not make the assumption that operators are performed serially, that is, perceptual, cognitive, and motor operators at the level of MHP processor cycle times can be performed in parallel as the task demands. Research has shown that CPM-GOMS models reflect an even higher level of skill than KLM or CMN-GOMS models (Baskin & John, 1998; John et al., 2002). CPM-GOMS uses a *schedule chart* (or *PERT chart,* familiar to project managers, see Stires & Murphy, 1962) to represent the operators and dependencies between operators. The acronym *CPM* stands for both the *c*ognitive-*p*erceptual-*m*otor analysis of activity, and also

[2] Some design situations may require, or provide opportunity for using better estimates of specific types of mental operators. Analysts can look at the additional empirical work of Card, Moran, and Newell (1983) in Chapter 5 where they measure many specific mental times, or other HCI empirical work (e.g. John & Newell, 1987 for estimates of time to recall command abbreviations, Olson & Olson, 1990, for mental preparation in spreadsheet use).

S
R
L

*c*ritical *p*ath *m*ethod, since the *critical path* in a schedule chart provides a simple prediction of total task time.

To build CPM-GOMS models, the analyst begins with a CMN-GOMS model of a task with operators at a level similar to those in Figure 4.7. These operators are then expressed as goals and implemented with methods of MHP-level operators. Researchers have developed templates of the combinations of MHP-level cognitive, perceptual, and motor operators that implement many different HCI activities like moving a mouse to a target and clicking, reading information from the screen, or typing (e.g., Gray & Boehm-Davis, 2000; John & Gray, 1992, 1994, 1995). Each operator in the templates is associated with a duration estimate, or a set of estimates that also depend on task conditions. For instance, visually perceiving and comprehending a six-character word is assigned a duration of 290 ms (John & Newell, 1989), whereas visually perceiving and comprehending that a symbol is merely present or absent (e.g., the presence of highlighting) is assigned a duration of 100 ms (Card et al., 1983).

CPM-GOMS models were traditionally constructed using project-management software (i.e., MacProject). Templates were preestablished and stored in a library file. The analyst would copy appropriate templates for a task into a blank canvas, then join them together serially to represent the operations necessary to accomplish the task. Finally, the analyst would interleave the templates to take advantage of the parallelism of the underlying conceptual architecture. Recently, a modeling tool built by NASA, Apex (Freed, 1998), has been used create CPM-GOMS models automatically from CMN-GOMS expressed in a procedure description language (PDL), and by-hand manipulation of operators in project-management software is no longer necessary (John et al., 2002). The schedule chart of the fox task, shown in Figure 4.8, was produced with that tool by encoding the CMN-GOMS model in Figure 4.7 in PDL (Remington et al., 2002).

Quantitative predictions of performance time can be read off the schedule-chart representation of the CPM-GOMS model. Qualitative analysis of what aspects of a design lead to what portions of the performance time are quite easy once the models are built, as are subtask profiling, sensitivity and parametric analyses, and playing "what-if" with suggested design features (e.g., Baskin & John, 1998; Chuah, John, & Pane, 1994; Gray & Boehm-Davis, 2000; Gray, John, & Atwood, 1993; John et al., 2002).

Continuing the fox example of Figure 4.2, Figure 4.8 shows the beginning of a CPM-GOMS model. For brevity, the model covers only the portion of the procedure involved with highlighting the text to be moved and pulling down the **Edit** menu; the model continues beyond the figure until it ends at time=4598 msec. Before discussing this model in detail, however, it is important to note that

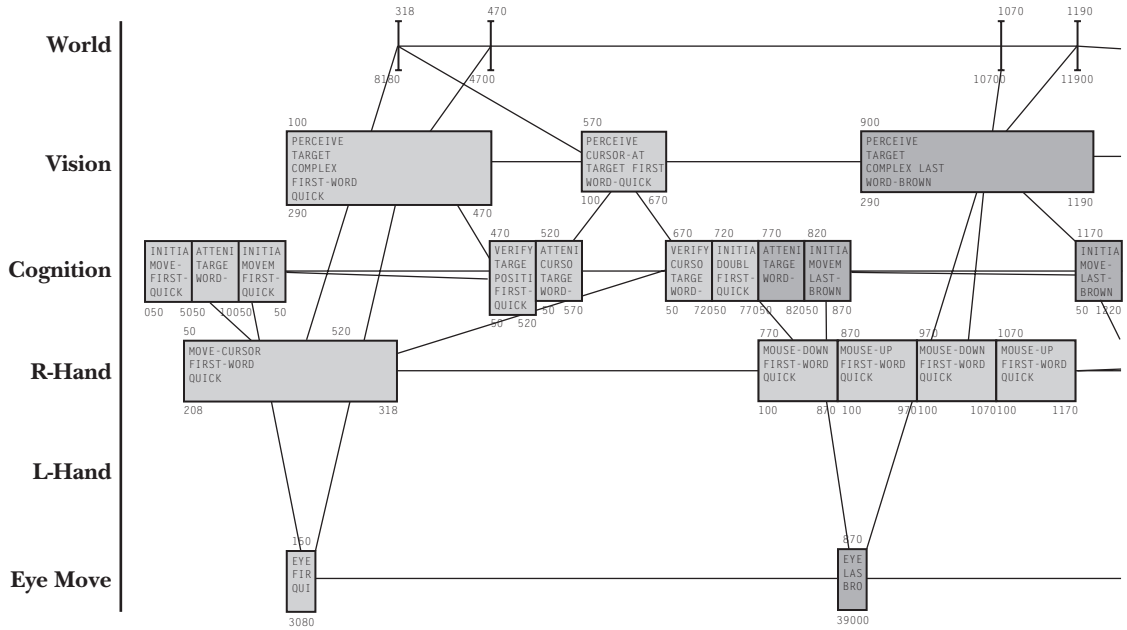
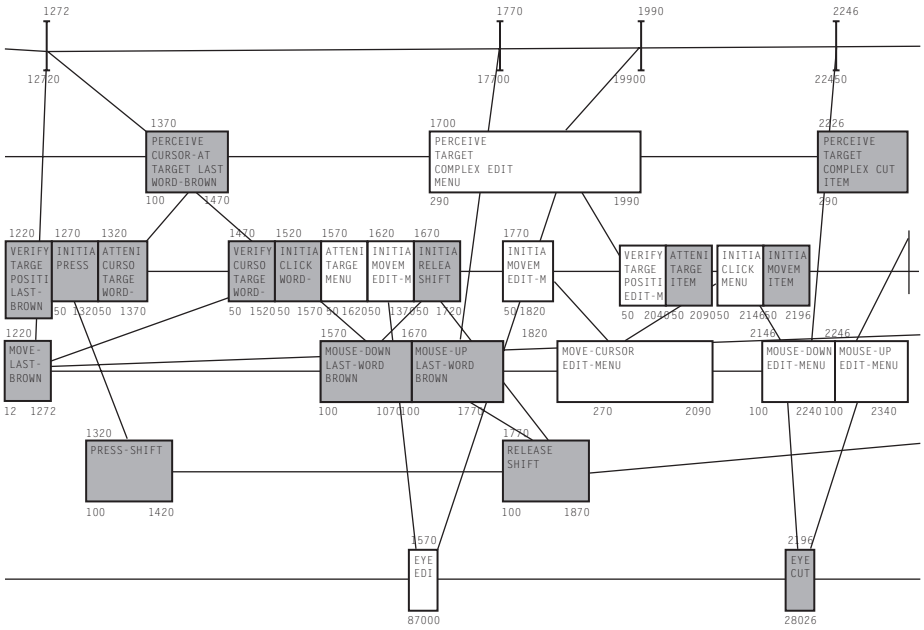


FIGURE 4.8

The beginning of the schedule chart representing the CPM-GOMS model of the fox-task editing example. This part of the chart includes highlighting the phrase to be moved and starting to issue the **Cut** command. Each operator is represented as a box, with its start time in the upper-left corner, its duration in the lower-left corner and its end time in the lower-right corner, in msec. The width of each box is proportional to its duration, so the schedule chart is also a timeline. The yellow boxes (see the color insert) are the cognitive, perceptual, and motor operators involved in double-clicking on the word "quick". The blue boxes are those involved in shift-clicking on the word "brown". The white boxes are those involved in pulling down the **Edit** menu. The purple boxes are the first four operators involved in selecting the **Cut** menu item. The schedule chart continues beyond this figure, until the end of the task at time=4598 msec. (See Plate 4 of the color insert for a color version of this figure.)

text editing is not a good application of the CPM-GOMS technique, and we present it here only so that similarities and differences to the other GOMS variations are clear. Text-editing is *usefully approximated* by serial processes, which is why the KLM and CMN-GOMS have been so successful at predicting performance on text editors. The CPM-GOMS technique is overly detailed for such primarily serial tasks and, as will become clear, can underestimate the execution time. For examples of tasks for which a parallel-processing model is essential, and where the power of CPM-GOMS is evident, see the telephone-operator task



that appears in the case-study section of this chapter (Gray, John & Atwood, 1993), transcription typing (John, 1988; John & Newell, 1989), and a highly interactive version of drawing in a CAD system (Baskin & John, 1998).

Although text editing is not the best task to display the advantages of CPM-GOMS, there are several interesting aspects of the model in Figure 4.8 compared to the example models of the text-moving task in the preceding sections. First, there is a direct mapping from the CMN-GOMS model to the CPM-GOMS model, because all CPM-GOMS models start with CMN-GOMS and the particular model in Figure 4.8 was built with reference to the one in Figure 4.7. For example, the CMN-GOMS operators MOVE-CURSOR-TO-FIRST-WORD and DOUBLE-CLICK-MOUSE-BUTTON in Figure 4.7 become goals in the CPM-GOMS model that are expanded into operators (yellow boxes) in Figure 4.8; these boxes represent the perceptual, cognitive, and motor operators necessary to move a mouse to a target and double-click on it.

As with the KLM, selection rules are not explicitly represented in the schedule chart because the schedule chart is merely the trace of the predicted behavior. The selection rules are represented in the PDL code, just as they are in CMN-GOMS. The duration times for the various operators, shown in the lower left corner of the boxes in the schedule chart, are based on the durations estimated by John and Gray (1992, 1994, 1995).

S
R
L

Parallelism is illustrated throughout the model. For instance, in the yellow set of operators the eye-movement and perception of information occur in parallel with the cursor being moved to the word QUICK. The information-flow dependency lines between the operators ensure that the eyes must get to the word and perceive it before the new position of the cursor can be verified to be at the right location.

Several goals can be active at one time in CPM-GOMS models, and this is illustrated in Figure 4.8 whenever operators of two colors occur at the same time, For example, the blue operators in service of shift-clicking on BROWN begin before the yellow operators in service of the double-click on QUICK complete. Later, the white operators in service of pulling down the EDIT-MENU intermingle with both the blue operators in service of the shift-click and the purple ones in service of selecting the CUT-MENU-ITEM. This interleaving represents a very high level of skill on the part of the user.

Reading the end time of the final item in the schedule chart (not shown) gives a total execution time through the fox task to be 4.598 sec. Totaling the execution time over the same steps in the other models gives 14.90 sec for the KLM and 16.25 sec for the CMN-GOMS model, making the quantitative prediction of the CPM-GOMS model much shorter than the estimates from the other models. The primary source of the discrepancy between the GOMS variants is the basic assumption in the commonly used form of the CPM-GOMS technique that the user is extremely experienced and executes the task as rapidly as the MHP architecture permits. It should be kept in mind that this particular example task is not really suitable for CPM-GOMS, but it is presented to facilitate comparison with the other techniques, and it shows how CPM-GOMS can represent parallel activities in the same editing task. Some discussion of why the CPM-GOMS technique predicts an execution time that is so much shorter than the others will help clarify the basic assumptions of this form of GOMS analysis.

One aspect of the extreme-expertise assumption is that the example model assumes that the user knows exactly where to look for the to-be-moved-phrase. This means that the model needs only one eye movement to find the beginning and one to find the end of the words to be moved, and that the mouse movements to these points can be initiated prior to the completion of the eye movements. In some real-world tasks, like telephone operators handling calls (see the case study later in this chapter, or Gray, John, & Atwood, 1993), the required information always appears at fixed screen locations, and, with experience, the user will learn where to look. But in a typical text-editing task like our example, the situation changes from one task instance to the next, and so visual search may be required to locate the target phrase. The CPM-GOMS has been used to

S
R
L

model visual search processes (Chuah, John, & Pane, 1994), but for brevity we did not include this complexity in our example.

A second aspect of the assumed extreme expertise is that the example does not include any substantial cognitive activity associated with selection of methods or complex decisions. Such cognitive activity is represented in the other GOMS variants with M-like operators of about a second in duration. In contrast, in Figure 4.8, the method selection is implicit in the trace produced in the schedule chart, but it does not appear as an operator itself. Likewise, VERIFY-POSITION operators are included in the CPM-GOMS model, but they represent much more elementary recognitions that the cursor is indeed in the location where the model is already looking rather than complex verifications that a text modification has been done correctly as required in CMN-GOMS model. Thus, Figure 4.8 represents the absolute minimum cognitive activity, which is an unreasonable assumption for a normal text-editing task. However, in an experiment by Card, Moran, and Newell (Card et al., 1983, pp. 279–286), the performance time of an expert user on a novel editing task was well predicted by the KLM; after 1100 trials on the *same task instance,* however, the performance time decreased by 35%, largely because the M operators became much shorter. It is this type of extreme expertise that our example CPM-GOMS model represents. Subsequent research (Baskin & John, 1998; John et al., 2002) has suggested that skilled human performance is well predicted by CPM-GOMS models after about 100 trials of the same task instance, while KLM and CMN-GOMS models predict about the fifth trial quite well. A more elaborate CPM-GOMS model could represent complex decisions as a series of MHP-level operators performing minute cognitive steps serially, as in the earlier work on recalling computer command abbreviations (John, (1988); John and Newell, (1989); John, Rosenbloom, & Newell, 1985). However, the technique for modeling complex decisions in CPM-GOMS models is still a research issue, and so these models currently should be used only for tasks in which method selection is based on obvious cues in the environment and in which decisions can be represented very simply.

A final contributor to the short predicted time is that the mouse movements in CPM-GOMS are calculated using Fitts' Law specifically for the particular target size and distance in this situation, yielding much shorter times than Card, Moran, and Newell's 1.10 sec estimate of average pointing time used in the other models. The 1.10 sec used in the KLM and CNM-GOMS model is the average value suggested by Card, Moran, and Newell for large-screen text editing tasks. But Gong (1993) found that many of the mouse movements involved in using a Macintosh interface, such as making menu selections and activating windows, were faster than 1.10 sec, and that Fitts' Law estimates (see Card et al., 1983,

S
R
L

p. 55) were much more accurate. Thus, Fitts' Law values based on the actual or typical locations of screen objects should probably be used whenever possible in all of the techniques. For CPM-GOMS, moving the cursor to point to an object is a combination of cognitive operators, motor operators, and perceptual operators (see Figure 4.8). The duration of the mouse-movement motor operator itself is calculated using Fitts' Law.

Thus, the CPM-GOMS technique allows one to represent the overlapping and extremely efficient pattern of activity characteristic of expert performance in a task. The main contrasts with the other techniques is that CPM-GOMS models constructed with the current technique do not include the time-consuming M-like operators that the other models do, and that would be expected to disappear with considerable practice if the system interface holds the relevant aspects constant. Furthermore, CPM-GOMS tends to use Fitts' Law for estimates of mouse-movement times (although any GOMS variant could use Fitts' Law, and CPM-GOMS could use 1.100 seconds, as was done by Baskin & John, 1998). Analysts need to consider whether their users are likely to attain the extreme level of skill modeled by CPM-GOMS when deciding which GOMS variant to use. The following case study illustrates a task where this assumption was reasonable and held true.

4.5 CASE STUDY: PROJECT ERNESTINE

Many instances of GOMS modeling have contributed to the design and evaluation of computer systems (see John & Kieras, 1996a, for a description of 11 examples). Perhaps the most economically successful is the case of Project Ernestine (Gray, John, & Atwood, 1993). In 1988, NYNEX, the telephone company then serving New York and New England, considered replacing the workstations used by toll and assistance operators (TAOs, who handled calls such as collect calls and person-to-person calls) with a new workstation. A major factor in making the purchase decision was determing how quickly the expected decrease in average work time per call would offset the capital cost of making the purchase. Because an average decrease of one second in work time per call would save an estimated $3 million per year, the decision was economically significant.

To evaluate the new workstations, NYNEX conducted a large-scale field trial. At the same time, Wayne Gray (at NYNEX) and I (at Carnegie Mellon University) used CPM-GOMS models to predict the outcome of the field trial. We worked with expert telephone operators to construct models for the current workstation for a set of benchmark tasks. We then modified these models to

reflect the differences in design between the two workstations; these included different keyboard and screen layout, keying procedures, and system response time. This modeling effort took about two person-months, but this time included making extensions to the CPM-GOMS modeling technique to handle this type of task and teaching NYNEX personnel how to use CPM-GOMS, compared to the 18-month elapsed time and the scores of people involved in the field trial. The models produced quantitative predictions of expert call-handling time for each benchmark task on both workstations, which, when combined with the frequency of each call type, predicted that the new workstation would be an average of 0.63 seconds slower than the old workstation. Thus the new workstation would not save money, but would cost NYNEX an additional $2 million a year to operate.

This was a counter-intuitive prediction. The new workstation had many technically superior features. The workstation used more advanced technology to communicate with the switch at a much higher speed. The new keyboard placed the most frequently used keys closer together. The new display had a graphic user interface with recognizable icons instead of obscure alphanumeric codes. The procedures were streamlined, sometimes combining previously separate keystrokes into one keystroke, sometimes using defaults to eliminate keystrokes from most call types, with a net decrease of about one keystroke per call. Both the manufacturer and NYNEX believed that the new workstation would be substantially faster than the old one—by one estimate, as much as 4 seconds faster per call. Despite the intuition to the contrary, when the empirical field-trial data were analyzed, they supported the CPM-GOMS predictions. The new workstation was 0.65 seconds slower on average than the old workstation.

In addition to predicting the quantitative outcome of the field trail, the GOMS models explained *why* the new workstation was slower than the old workstation, something which empirical trials typically cannot do. The simple estimate that the new workstation would be faster was based on the greater speed of the new features considered in isolation. But the execution time for the whole task depends on how all of the components of the interaction fit together, and this is captured by the critical path in the CPM-GOMS model. Because of the structure of the whole task, the faster features of the new workstation failed to shorten the critical path.

The pragmatic result of both the empirical results and the CPM-GOMS modeling was that NYNEX decided not to buy the new workstations. The scientific result was that CPM-GOMS modeling was shown to predict real-world performance of skilled users extremely well and it was shown to provide explanations for their behavior that were both reasonable and easy to communicate to users, developers, and managers. The following section will provide some more detail about

the task and the models to illustrate the power that CPM-GOMS brought to NYNEX's buy/no-buy decision.[3]

4.5.1 Details of Project Ernestine's CPM-GOMS Modeling Effort

Predicting skilled execution time is a strength of GOMS modeling. Such predictions are successful when the users are experts performing a routine cognitive skill and making few errors. These conditions were satisfied in Project Ernestine, making this task a suitable candidate for GOMS modeling. A TAO handles hundreds of calls each day, and many stay at the job for years (even decades). TAOs recognize each call situation and execute well-practiced methods, rather than engage in problem solving. As for errors, the call-handling system is designed to preclude many types of errors (for example, the workstation will not release a call unless all necessary information is entered), and experienced TAOs make few errors of any type.

Successful GOMS models also depend on the task having clearly identifiable goals, operators, methods, and (if necessary) selection rules. The task of a TAO has these characteristics, as described with reference to the following example.

In 1988, a TAO was the person to whom customers spoke when they dialed "0" (many of the functions handled by TAOs are now automated). A TAO's job was to assist a customer in completing calls and to record the correct billing. Among other duties, TAOs handled person-to-person calls, collect calls, calling-card calls, and calls billed to a third number. (TAOs did not handle directory assistance calls.) For example, consider the situation where a person directly dials a number and has the TAO bill the call to his or her calling card. The dialog is sparse, but typical:

Workstation:	B"Beep"
TAO:	"New England Telephone, may I help you?"
Customer:	"Operator, bill this to 412–555–1212–1234."
TAO:	Keys information into the workstation.
TAO:	"Thank you."
TAO:	Releases the workstation to accept the next incoming call by pressing the RELEASE key.

[3] Much of the following section is adapted from Gray, John, and Atwood, 1993, where more detail, including a complete model of a telephone call and details of the empirical study, can be found.

TAOs were trained to answer three questions in the course of a call; these questions correspond to three goals in a GOMS analysis. For each call, a TAO had to determine (1) who should pay for the call, (2) what billing rate to use, and (3) when the connection is complete enough to terminate interaction with the customer. In the above calling-card call, (1) the owner of the calling card pays for the call, (2) the operator-assisted station rate is applied, and (3) the connection is complete when the calling-card number is approved. It was not necessary for the call to actually go through because, if the call did go through, then the correct billing would be applied; if it did not go through, the calling card would not be charged.

To accomplish these goals, TAOs converse with the customer, key information into a workstation, and read information from the workstation screen. In some cases, they also write notes to themselves (primarily to remember callers' names for collect calls). Thus, the GOMS operators for accomplishing the goals include listening, talking, reading, keying, writing, and the cognitive activities necessary to assimilate information and determine action. An additional complexity, beyond the sheer variety of activities, is that TAOs appear to perform many of them simultaneously. The TAO types information into the workstation while listening to the customer and scanning the screen for information relevant to the call. Thus, this task was particularly suited to CPM-GOMS modeling.

Many of these activities had been successfully modeled in GOMS research prior to Project Ernestine. Heuristics for modeling the perception of words on a display screen and for modeling typing had already been developed (John, 1988; John & Newell, 1989; John, Rosenbloom, & Newell, 1985). To handle the variety of activities the TAO must perform on the current workstation, we made extensions to CPM-GOMS to model auditory perception, verbal responses, eye movements to get information from well-known positions on a CRT screen, and system response time. These extensions were made early in the project (John, 1990) and their derivation and general use is discussed in more detail elsewhere (Gray, John, & Atwood, 1993; John & Gray, 1992).

The TAOs accomplished their goals using a dedicated workstation, and the details of the workstation design influenced the CPM-GOMS models. The differences in screen and keyboard layout, keying procedures, and system response times between the current and proposed workstations affected the methods used to process a call, as well as the duration of specific operators within a method (for example, the duration of the horizontal movements to some keys changed between workstations). However, expressing the differences between workstations in the models was straightforward; no extensions to CPM-GOMS were required to express these differences.

S
R
L

Building the CPM-GOMS Models

Figure 4.9 shows the traditional GOMS goal hierarchy and the beginning of the CPM-GOMS model that implements it. We built 36 such CPM-GOMS models, one for each workstation for 18 different benchmark calls selected for frequency of occurrence and importance to NYNEX (e.g., emergency calls are infrequent but important to accomplish in the minimum time necessary). Fifteen of these benchmarks covered more than 80% of the calls handled by TAOs and are included in the quantitative predictions discussed in the next section. The durations of the CPM operators were determined in one of three ways: observed in videos of the benchmark calls, from previous psychological literature, or estimated by the proposed workstation manufacturer.

The durations for all pausing and speaking, by both the TAOs and the customers, were measured from videotapes of the benchmark calls handled by expert TAOs with the current workstation. These included complex auditory perceptual operators, such as the real-time perception and comprehension of the customer's phrase "make this collect," as well as the duration for the TAO's motor operator to say "New England Telephone, may I help you?" Since the same conversations appeared in the benchmark calls for both the current and proposed workstations, these times observed on the current workstation were used in both sets of models. The duration of hand movements for pressing keys on the current workstations are also set from measurements of the videotapes, as are all the system response times for the current workstation (but not the proposed workstation, discussed later).

Normative estimates for unobservable operators were obtained from the psychological literature and can be considered off-the-shelf estimates of how long an average person requires to perform a particular operator. Cognitive operators are assumed to be of equal duration, 50 msec (John & Newell, 1990). The motor operator that makes an eye movement to a known screen location was 30 msec (Russo, 1978).[4] Binary visual perception operators, used when the TAO must detect only the presence or absence of a visual signal, are assumed to be 100 msec

[4] Early papers about Project Ernestine used 180 msec for the duration of the eye movement. This was derived from the 1983 Card et al., estimate of 230 msec for an eye movement subdivided into a cognitive operator, 50 msec, to initiate the movement and the movement action itself, 180 msec. We discovered later that the eye movement estimated by Card et al. included some amount of perception and comprehension, which should be separated from the actual eye movement time in CPM-GOMS. Card et al. referred to Russo (1978) for the travel time alone being 30 msec, much shorter than 180 msec. The eye movements never appeared on the critical path of the Ernestine models, and shortening them would therefore not affect the time predictions. All CPM-GOMS models after 1992 use 30 msec for eye-movement time.

S
R
L

GOMS goal hierarchy	Observed Behavior
goal: handle-calls	
. **goal: handle-call**	
. . **goal: initiate-call**	
. . . **goal: receive-information**	
. . . . listen-for-beep	Workstation: Beep
. . . . read-screen(2)	Workstation: Displays source information
. . . **goal: request-information**	
. . . . greet-customer	TAO: "New England Telephone, may I help you?"
. . **goal: enter-who-pays**	
. . . **goal: receive-information**	
. . . . listen-to-customer	Customer: Operator, bill this to 412-555-1212-1234
. . . **goal: enter-information**	
. . . . enter-command	TAO: hit F1 key
. . . . enter-calling-card-number	TAO: hit 14 numeric keys
. . **goal: enter-billing-rate**	
. . . **goal: receive-information**	
. . . . read-screen(1)	Workstation: previously displayed source information
. . . **goal: enter-information**	
. . . . enter-command	TAO: hit F2 key
. . **goal: complete-call**	
. . . **goal: request-information**	
. . . . enter-command	TAO: hit F3 key
. . . **goal: receive-information**	
. . . . read-screen(3)	Workstation: displays credit-card authorization
. . . **goal: release-workstation**	
. . . . thank-customer	TAO: "Thank you"
. . . . enter-command	TAO: hit F4 key

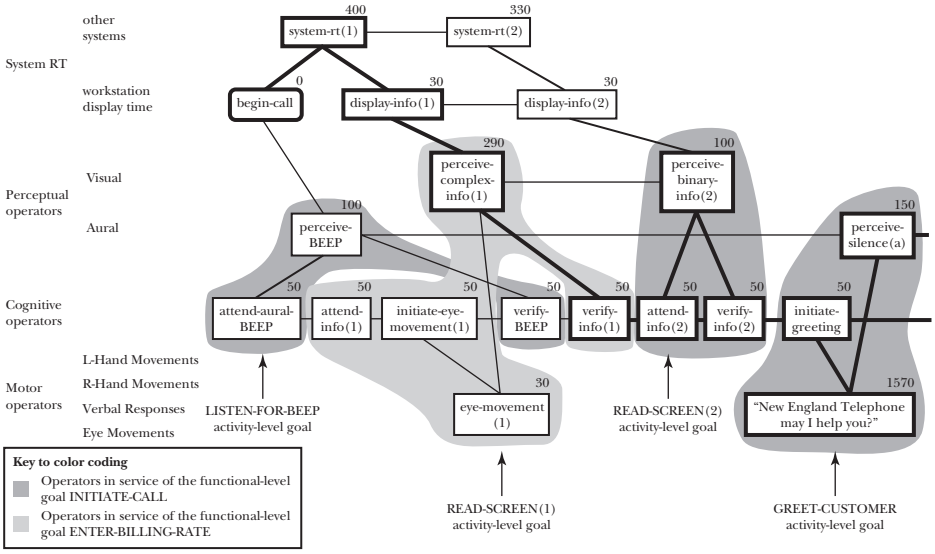


FIGURE 4.9 GOMS hierarchy down to the activity level, and CPM-GOMS implementation of that hierarchy. Shading in the hierarchy corresponds to the background shading of the operators in the CPM-GOMS model. (Adapted from John, 1990, figure 1, pp. 110–111, and Gray, John, & Atwood, 1993, figure 7, p. 254.)

S
R
L

(minimal perceptual operator in the MHP; Card et al., 1983). For example, in Figure 4.9, the perception of info(2) is a binary perception; information-processing analysis revealed that the TAO need only detect that a code appears in that spot on the screen, not what the code actually says (because it always says the same thing). In contrast, perceiving info(1) in Figure 4.9 is a complex visual perception because information-processing analysis revealed the TAO must perceive and comprehend the semantics of the code displayed, as well as the presence of the code, to get sufficient information to continue with the call. The complex visual perceptions required of the TAO are all of small words, alphanumeric codes, or numbers, and they are assumed to take 290 msec because they are of similar character to the small-word recognition tasks used to estimate that duration (this is derived from the John & Newell [1989] estimate of 340 msec for the perception and encoding of a short word subdivided into a 290 msec perceptual operator and a 50 msec cognitive operator to verify expectations). Binary auditory perceptual operators, such as detecting the "beep" that signals an incoming call, are also set at 100 msec (minimal perceptual operator of the MHP; Card et al., 1983). The perception of an auditory silence that signals turn taking in conversation is estimated at 300 msec (this is the 400 msec mean interspeaker pause found by Norwine and Murphy [1938] subdivided into a 300 msec perceive-silence operator followed by a 50 msec cognitive operator to verify the silence and a 50 msec cognitive operator to initiate the spoken response). Finally, for the horizontal hand movements required by the proposed workstation, we used Fitts' Law to predict the time to execute that movement. We considered this a "normative" estimate because Fitts' Law is a well-established empirical regularity (see Chapter 3).

To complete the models, the manufacturer of the proposed workstation supplied estimates for the expected response time of the proposed system to various TAO actions. These estimates were used in the CPM-GOMS models of the proposed workstation whenever the workstation had to perform an action like displaying information or looking up a number in a database.

In summary, the models of the current workstation used both normative estimates of durations and measurements taken from videotaped observations of the current workstation. In contrast, the models of the proposed workstation used only preestablished estimates from videotapes of the current workstation, normative estimates from the literature, and estimates of system response time supplied by the manufacturer. The proposed workstation was never observed in operation and might as well have not yet even existed. The models of the proposed workstation could have just as easily been created from a specification of that workstation.

S
R
L

Accuracy of the Models' Time Predictions

The 30 models of the 15 most frequent calls produced quantitative predictions of call-completion time. The most important measure to NYNEX was the difference between the current workstation and the proposed workstation averaged across call categories, but we also examined the absolute and relative predictions for each call category.

When each model was weighted by the frequency of occurrence of its call category, CPM-GOMS predicted that the proposed workstation would be an average of 0.63 seconds slower than the current workstation. For comparison, when the empirical data were weighted by the frequency of call occurrence, the proposed workstation was 0.65 seconds slower than the current one. This overall prediction was the most important one to NYNEX. Pragmatically, with an average decrease of one second per call saving $3 million per year in operating costs, the ability to quantitatively predict performance on the mixture of calls that NYNEX TAOs handled was the most prized prediction. This small difference in work time would cost NYNEX an estimated $2 million per year more than the current workstations' operating costs. The CPM-GOMS models predicted the overall outcome of the field trial with remarkable accuracy. As for predictions of workstation difference for each individual call category, CPM-GOMS predicted the direction of the difference for all but 3 of the 15 call categories.

Looking at the absolute time predictions, the CPM-GOMS models for the current workstation—when weighted by call category frequency—underpredicted the trial data by an average of 4.35%. This underprediction was continued by the models of the proposed workstation, with these models predicting a weighted worktime 4.31% faster than the trial data. These weighted predictions are well within the 20% error limit that previous work (John & Newell, 1989) has argued is the useful range of an engineering model. Because these underpredictions were consistent at about 4%, the relative prediction of the two sets of CPM-GOMS models (0.63 seconds predicted versus 0.65 seconds found in the empirical data) is more accurate than the predictions of absolute call time themselves.

Across call categories unweighted by frequency, the average percent difference between the CPM-GOMS models and the observed calls was 11.30% for the current workstation and 11.87% for the proposed workstation. The correlation between the CPM-GOMS predictions and the trial data was significant with an r^2 of 0.71 for the current workstation and of 0.69 for the proposed workstation. For each workstation and call category, the z scores show that for 14 of the 15 call categories the CPM-GOMS prediction is within one standard deviation of the trial mean for both current and proposed workstations. These data support the

S
R
L

conclusion that the CPM-GOMS models predict the trial data in the aggregate with acceptable accuracy.

The individual predictions of worktime per call category were less accurate than the aggregate measures. The percent difference between predicted and observed time per call category for the current workstation ranged from −63% to +49%, with eight call categories more than 20% away from their observed times. Likewise, the percent difference for the proposed workstation ranged from −54% to +49%, with the same eight call categories being more than 20% away from the observed times. These general results—that the overall prediction of work time (both weighted by call frequency and unweighted) is very good while the individual predictions of call category is not as good—are a statistical fact of life. If the individual predictions vary more or less randomly around the actual call times, some being too short and some being too long, aggregate measures will involve some canceling-out of these predictions. Since the aggregate measures are of primary importance to NYNEX, this fluxuation at the level of individual call types is interesting to examine but not too important to the results of the modeling effort. The lesson to be learned for future modeling efforts is to be sure to model a substantial suite of benchmark tasks as opposed to depending on one or two.

Explaining the Differences Between Workstations

Beyond predicting performance time, the CPM-GOMS models provide explanations for their predictions and thereby for the empirical data. Despite its improved technology and ergonomically superior design, performance with the proposed workstation was slower than with the current workstation. A high-order look at the critical paths shows the task to be dominated by conversation and system response time. Seldom is the TAO's interaction with the workstation on the critical path. This pattern is so strong that it was found in our initial model of just one call category (Gray et al., 1989), and so consistent that we declared it confirmed (Gray et al., 1990) after modeling five call categories. Thus, the top-order prediction of the CPM-GOMS analyses is that the design of the workstation should have little, if any, effect on the length of calls.

We can look at the details of the models to understand why the proposed workstation is actually slower than the current workstation. The workstations differ in their keyboard layout, screen layout, keying procedures, and system response time, each of which may effect call duration. Here we examine just the keying procedures to illustrate the explicative power of the CPM-GOMS models. The other effects are examined in Gray, John, and Atwood (1993).

S
R
L

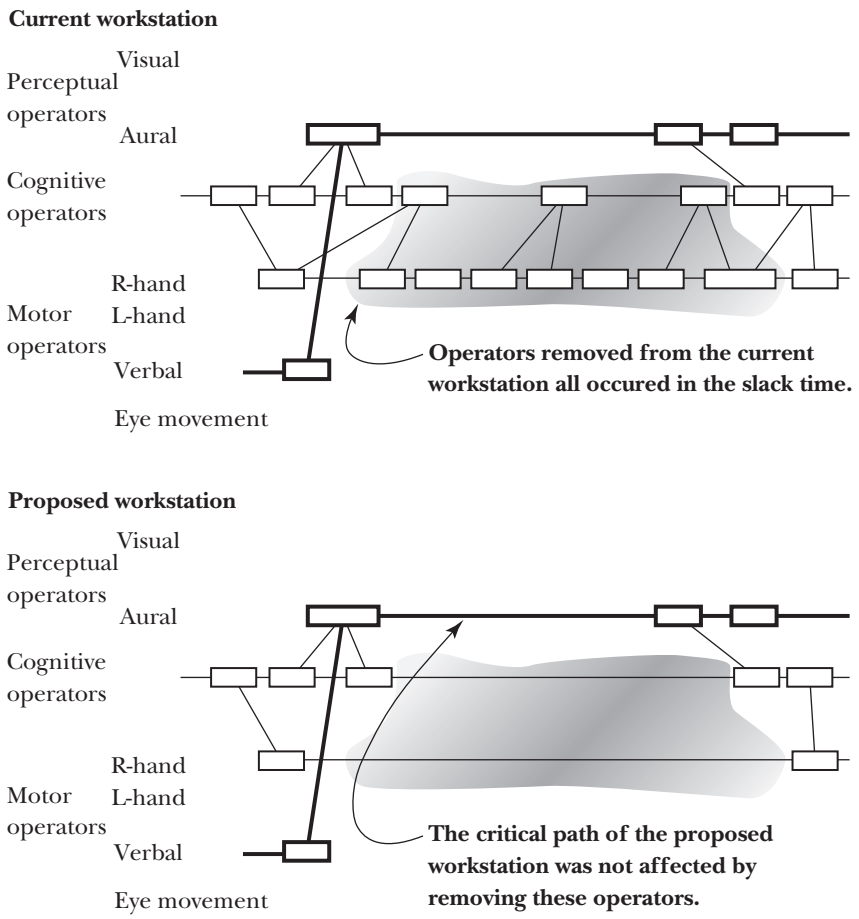


FIGURE 4.10 Section of the CPM-GOMS model from near the beginning of the call. (Adapted from Gray, John, Stuart, Lawrence, & Atwood, 1990, figure 1, p. 32; Gray, John, & Atwood, 1992, figure 1, p. 310; Gray, John, & Atwood, 1993, figure 19, p. 284; Atwood, Gray, & John, 1996 Figure 1, p. 108.)

For several calls, the keying procedures for the proposed workstation eliminated keystrokes. In some of these calls, this decrease in keystrokes was an advantage for the proposed workstation. However, because of the complex interaction of parallel activities in the TAOs' task, merely eliminating keystrokes is not necessarily an advantage. For example, Figure 4.10 and Figure 4.11 show the first and last segments of a CPM-GOMS analysis for a calling-card call where new

S
R
L

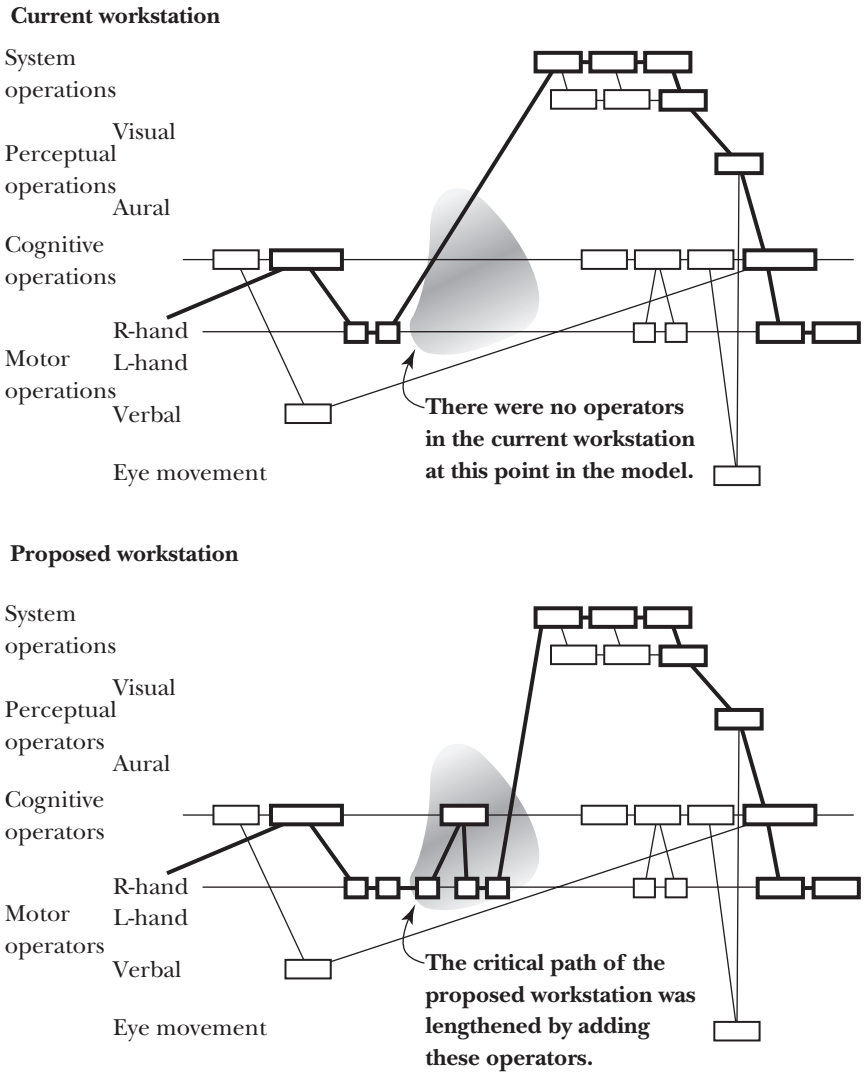


FIGURE 4.11 Section of the CPM-GOMS model from the end of the call. (Adapted from Gray, John, Stuart, Lawrence, & Atwood, 1990, figure 2, p. 32; Gray, John, & Atwood, 1992, figure 2, p. 310; Gray, John, & Atwood, 1993, figure 10, p. 285; Atwood, Gray, and John, 1996 figure 2, p. 109.

S
R
L

procedures eliminated two keystrokes from the beginning of the call and added one keystroke to the end of the call, for a net decrease of one keystroke. For each figure, the top chart represents the call using the current workstation, and the bottom shows the CPM-GOMS analysis for the same call using the proposed.

Figure 4.10 has two striking features. First, the model for the proposed workstation has 10 fewer boxes than the model for the current workstation, representing two fewer keystrokes. Second, none of the deleted boxes are on the critical path; all are performed in slack time. At this point in the task, the critical path is determined by the TAO greeting and getting information from the customer. The CPM-GOMS model predicts that removing keystrokes from this part of the call will not affect the TAO's work time. Work time is controlled by the conversation, not by the keystrokes and not by the ergonomics of the keyboard.

The middle of the model, not shown (the activities between those shown in Figure 4.10 and Figure 4.11), is identical for both workstations and essentially shows that the critical path is driven by how fast the customer says the 14-digit number to which the call should be billed. TAOs are taught to "key along" with the customer. While a rapidly speaking customer could force the critical path to be determined by the TAO's keying speed, both workstations use the standard numeric keypad, so the critical path (and resulting speed of keying in numbers) would be the same for both workstations in the middle of the call.

If the proposed keying procedures simply eliminated the two keystrokes required by the current workstation in the beginning of the call, then CPM-GOMS would predict equivalent performance. However, for the proposed workstation, the procedure has been changed so that one of the keystrokes eliminated at the beginning of the call now occurs later in the call (four extra boxes in the bottom of Figure 4.11). In this model, this keystroke goes from being performed during slack time at the beginning of the call, to being performed on the critical path at the end of the call. The cognitive and motor time required for this keystroke now add to the total time required to process this call. Thus, the net elimination of one keystroke actually increased call time because of the complex interaction between parallel activities shown in the critical-path analysis. The CPM-GOMs models showed similar clear reasons for each design decision having either no effect on total time, slightly decreasing total time, or, as in this case, increasing the total time of the call.

Value-Added of the CPM-GOMS Models

A simple, seemingly reasonable calculation can be done to predict worktime differences between the current and proposed workstations without cognitive modeling. Such a calculation was made before Project Ernestine, set NYNEX's initial

expectations of improved performance with the proposed workstation, and justified the expense of the field trial. Here we work through such a calculation and compare its accuracy to the CPM-GOMS predictions to evaluate the value-added of the human information processing approach in the form of CPM-GOMS.

The benchmark tasks are also used in the non-HIP estimate of workstation differences. First, the proposed workstation changed the keying procedure to eliminate keystrokes for several call categories. From Card, Moran, and Newell, we get an estimate of 280 msec per keystroke for an average, 40 wpm, nonsecretary typist (Card et al., 1983, Figure 9.1, p. 264). For each call category, this time was subtracted for each keystroke that the manufacturer's procedures eliminated. Four keystrokes were eliminated from one benchmark call; two keystrokes from two calls; one keystroke from each of seven calls; zero keystrokes from four calls; and one keystroke was added to one call. Second, the manufacturer estimated that the proposed workstation would be 880 msec faster than the current workstation to display a screenful of information. We subtracted this estimate from every benchmark call because every call displays a screenful of information. From these two facts, we would predict an average advantage for the proposed workstation of 5.2%. When call categories are weighted by their frequency of occurrence, the predicted advantage becomes 18.6% (4.1 sec) for an estimated *savings* in annual operating costs of $12.2 million.

In contrast, the CPM-GOMS models predicted, and the field trial confirmed, that the proposed workstation would actually be about 3% *slower* than the current workstation. Thus, the seemingly reasonable calculation based on the benchmarks and manufacturer's procedures and response-time estimates is wrong in both magnitude and sign. It is important to remember that the non-HIP prediction is more than just a strawman. Large-scale empirical trials such as Project Ernestine are expensive to conduct; this one involved dozens of workstations, scores of people, and many months of planning, training, data collection, and analysis. Expectations based upon such a calculation led NYNEX to commit to the time and expense required to conduct the empirical trial.

Why were the CPM-GOMS predictions so much more accurate than the noncognitive predictions? Two reasons are apparent: (1) Building CPM-GOMS models requires that the analyst understand the details of information flow between the workstation and the TAO, which were overlooked by the non-HIP predictions, and (2) CPM-GOMS models incorporate the complex effects of parallel activities that were important to this task.

For example, the non-HIP predictions assumed that each time a screenful of information was displayed, the proposed workstations' faster system response time would reduce the time of the call. However, the more detailed analysis

S
R
L

required to build CPM-GOMS models revealed that the TAO does not have to see the entire screen to initiate the greeting, just the first line, and therefore the TAO is conversing with the customer while the rest of the visual information is being displayed on the screen. Hence, comparisons of how fast the two workstations display an entire screen of information are largely irrelevant. Likewise, the noncognitive model assumes that every keystroke contributes to the length of the call. However, as we discussed above, CPM-GOMS shows that removing a keystroke only speeds the task if that keystroke is on the critical path.

Thus, CPM-GOMS disciplines the analyst to incorporate the right level of detail to evaluate such tasks and correctly calculates the effects of parallel activities to produce accurate quantitative predictions. A non-HIP approach based on benchmarks and design changes alone does not predict as accurately. In addition to producing more accurate quantitative predictions, CPM-GOMS models can provide qualitative explanations for the quantitative results and can also be used as a tool in workstation design (Gray, John, & Atwood, 1993). Clearly, CPM-GOMS adds value over non-HIP predictions.

4.6 CURRENT STATUS

4.6.1 GOMS in Particular

GOMS modeling may be the most active area of the information-processing approach in HCI applications. A search for "GOMS" in the HCI bibliography returns more than a hundred entries, including a 1996 paper in Association for Computing Machinery Transactions on Computer Human Interaction (ACM ToCHI) that profiles 11 cases of GOMS used in industry and government projects (John & Kieras, 1996a). Papers using GOMS appear every year in the ACM Computer-Human Interaction (CHI) conference, and research continues into extending GOMS and making it more accessible. Birds-of-a-Feather meetings at the CHI conference fill the room with scores of interested practitioners and researchers. GOMS appears in many HCI textbooks—Brinck, Gergle, & Wood (2002); Dix, Finlay, Abowd, & Beale (1998); Eberts (1994); Helander, Landauer, & Prabhu (1997); Newman & Lamming (1995); Preece, Rogers, Sharp, Benyon, Holland, & Carey (1994); Raskin (2000); Shneiderman (1998)—and is taught in HCI classes.

Although community interest is high, it is fair to say that few HCI practitioners consider GOMS a ready-to-hand tool to be used routinely in their design

practice. One reason may be the lack of readily-available tool support for the techniques. A review of several GOMS tools, GLEAN (Kieras et al., 1995), CAT-HCI (Williams, 1993), and Cognet (Zachary, 1989), highlighted the progress that had been made in this area and the needs still unfulfilled (Baumeister et al., 2000). Other tools are being explored in research labs, such as CRITIQUE (Hudson et al., 1999) and Apex (John et al., 2002), but no widely used commercial-quality tool has emerged at this writing.

4.6.2 Human Information Processing in General

The human information processing (HIP) approach, however, is broader than GOMS. HIP can be used to model more than the routine cognitive skill to which GOMS is restricted. It can be used to model more complex human behaviors like problem solving, learning, and group interaction, which are also crucial for the design of complex systems. Work in these areas of HIP, and in the area of integrating it all within the computational cognitive architectures discussed in Section 4.3.2, lived mainly in the realm of psychology research for several decades (1970s through 1990s), with steady progress in understanding its capabilities and limitations, but with little application to real-world HCI for much of that time.

However, a few highly successful applications of HIP did emerge from research in the late 1980s and early 1990s that dramatically demonstrated the power and potential of the HIP approach for applications. For instance, Project Ernestine demonstrated that the HIP approach could be taken out of the laboratory and used to predict performance on a real-world task with large financial significance.

Intelligent tutoring systems to teach high school mathematics were developed using the ACT-R architecture (Anderson et al., 1995) deployed in real classrooms, and were shown to produce significant improvement in mathematics standardized tests for real students under real conditions. These tutoring systems contain a cognitive model of the problem-solving knowledge students are to acquire. The tutors compare student behavior and model behavior step-by-step, to estimate the student's growing knowledge state, to individualize the problem sequence, and to provide advice as needed. The success of these cognitive tutors demonstrated that HIP could have a significant impact for education. These systems have been commercialized and are currently used by more than 100,000 high school students *(www.carnegielearning.com)*.

Finally, the HIP approach embodied in the Soar architecture was used in developing intelligent forces (IFors) for use in simulated theatres of war (STOWs)

(Tambe et al., 1995). These IFors flew simulated airplanes and helicopters in multiday STOWs where only a fraction of the tens of thousands of participants were human. They accepted instructions from commanding officers and planned missions in the same amount of time as human pilots. They communicated with each other to coordinate activities. They flew missions following the plan, but they also reacted to unexpected occurrences as human pilots would. These systems demonstrated that the HIP approach was stable enough and flexible enough to interact for long periods of time, under realistic conditions, and in coordination with each other and with humans. This approach has also been commercialized, and Soar Technology, Inc. produces IFors for several branches of the U.S. Department of Defense *(www.soartech.com)*.

The progress in the research labs and the successes in the field led to an examination of the state of the science of modeling human behavior by a panel of experts convened by the U.S. National Research Council's Committee on Human Factors. The resulting book, *Modeling Human and Organizational Behavior* (Pew & Mavor, 1998), is dominated by the HIP approach (although other approaches are discussed). As well as detailing the state of the science circa 1998, the book maps a route for future research and application. It calls for basic research, application with feedback to research, data collection for validation, and an infrastructure that facilitates scientific progress.

Subsequent to the publication of *Modeling Human and Organizational Behavior,* many new or expanded research and application opportunities have arisen. Several branches of the U.S. armed forces have commissioned HIP modeling efforts both in basic and applied research. Comparisons between different cognitive architectures are being conducted (Gluck & Pew, 2001). New areas of research are being supported, such as modeling the effects of environmental or emotional stressors on human performance. New arenas of application have opened up, such as modeling the effects of using interactive devices (cell phones, navigation systems, etc.) while driving an automobile (Salvucci & Macuga, 2002) and providing cognitively plausible software agents as video game opponents (Laird, 2001). New communities are also arising around the world, congregating at the International Conference on Cognitive Modeling begun in the late 1990s *(www.hfac.gmu.edu/~iccm/),* as well as having a continued presence in older communities like Human Factors and Ergonomics Society, ACM SIGCHI, and the Computer Generated Forces and Behavioral Representation conference. In short, thirty years after human information processing emerged as a respectable approach in psychology for researching human behavior, it enjoys a vibrant, exciting, and productive role in the creation of human-machine systems.

S
R
L

4.7 FURTHER READING

Readers interested in pursuing the topics covered in this chapter in more depth are referred to the following publications.

4.7.1 Seminal Text in Human Information Processing

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

4.7.2 Human Information Processing in HCI

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

John, B. E., & Kieras, D. E. (1996a). Using GOMS for user interface design and evaluation: Which technique? *ACM Transactions on Computer-Human Interaction 3*(4), 287–319.

John, B. E., & Kieras, D. E. (1996b). The GOMS family of user interface analysis techniques: Comparison and contrast. *ACM Transactions on Computer-Human Interaction 3*(4), 320–351.

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

4.7.3 Human Information Processing Embodied in Computational Cognitive Architectures

Introduces the concept of unified theories of cognition (computational cognitive architectures) and provides the Soar architecture as an example.

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

Describes the ACT-R and ACT-RPM (with perceptual and motor modules) architecture.

Anderson, J. R. (1983). *The Architecture of Cognition*. Cambridge, MA: Harvard University Press.

Anderson, J. R. (1993). *Rules of the Mind*. Hillsdale, NJ: Erlbaum.

Anderson, J. R., & Lebiere, C. eds. (1998) *The atomic components of thought,* Mahwah, NJ: Erlbaum

Describes the EPIC architecture.

Kieras, D. & Meyer, D. E. (1997). An overview of the EPIC architecture for cognition and performance with application to human-computer interaction. Human-Computer Interaction., 12, 391–438.

Meyer, D. E., & Kieras, D. E. (1997a). A computational theory of executive control processes and multiple-task performance: Part 1. Basic Mechanisms. *Psychological Review,* 104, 3–65.

Meyer, D. E., & Kieras, D. E. (1997b). A computational theory of executive control processes and human multiple-task performance: Part 2. Accounts of Psychological Refractory-Period Phenomena. *Psychological Review.* 104, 749–791.

Reviews the state of the art of human performance modeling. Chapter 3 reviews 11 different computational cognitive architectures.

Pew, R. W, & Mavor, A. S. (1998). Modeling Human and Organizational Behavior: Application to Military Simulations. Panel on Modeling Human Behavior and Command Decision Making: Representations for Military Simulations. Washington, DC: National Academy Press.

S
R
L