The GOMS Family of User Interface Analysis Techniques: Comparison and Contrast

BONNIE E. JOHN
Carnegie Mellon University
and
DAVID E. KIERAS
University of Michigan

Since the publication of *The Psychology of Human-Computer Interaction*, the GOMS model has been one of the most widely known theoretical concepts in HCI. This concept has produced several GOMS analysis techniques that differ in appearance and form, underlying architectural assumptions, and predictive power. This article compares and contrasts four popular variants of the GOMS family (the Keystroke-Level Model, the original GOMS formulation, NGOMSL, and CPM-GOMS) by applying them to a single task example.

Categories and Subject Descriptors: H.1.2 [**Models and Principles**]: User/Machine Systems—*human information systems*

General Terms: Human Factors

Additional Key Words and Phrases: Cognitive modeling, GOMS, usability engineering

Work on this article by B. John was supported by the Office of Naval Research, Cognitive Science Program, under contract number N00014-89-J-1975N158, and the Advanced Research Projects Agency, DoD, and was monitored by the Office of Naval Research under contract N00014-93-1-0934. Work on this article by D. Kieras was supported by the Office of Naval Research, Cognitive Science Program, under contract number N00014-92-J-1173 NR 4422574, and the Advanced Research Projects Agency, DoD, and was monitored by the NCCOSC under contract N66001-94-C-6036. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Office of Naval Research, NCCOSC, the Advanced Research Projects Agency, or the U.S. Government.
Authors' addresses: B. E. John, Human-Computer Interaction Institute and Departments of Computer Science and Psychology, Carnegie Mellon University, Pittsburgh, PA 15213; email: bonnie.john@cs.cmu.edu; D. E. Kieras, Department of Electrical Engineering and Computer Science, University of Michigan, Advanced Laboratory Building, 1101 Beal Avenue, Ann Arbor, MI 48109-2110; email: kieras@eecs.umich.edu.
Permission to make digital/hard copy of part or all of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.
© 1996 ACM 1073-0516/96/1200–0320 $03.50

1. INTRODUCTION

Since the publication of *The Psychology of Human-Computer Interaction* [Card et al. 1983] (hereafter, CMN), GOMS analysis has been one of the most widely known theoretical concepts in HCI. The GOMS concept, that it is useful to analyze knowledge of how to do a task in terms of **G**oals, **O**perators, **M**ethods, and **S**election rules, provided the stimulus for much research that verifies and extends the original work. Today, there are several variants of the GOMS analysis technique, and many applications of the technique in real-world design situations [John and Kieras 1996].[1] However, the clear differences between these techniques can create confusion about how they relate to each other and to the original concept. The purpose of this article is to compare several of the popular variants, demonstrating and discussing their similarities and differences.

This article is *not* a tutorial in how to use any version of GOMS; that information is elsewhere in textbooks, handbooks, and tutorial notes [Card et al. 1983; John and Gray 1995; Kieras 1988; 1996]. It is also not a guide for deciding when to use the variants of GOMS in a particular design situation; that information is in the companion article [John and Kieras 1996]. This article presents how different GOMS techniques are related.

We will examine four variants of GOMS: the simplest version presented by Card, Moran, and Newell, called the Keystroke-Level Model (KLM); the original formulation of GOMS, which we will refer to as CMN-GOMS; a more rigorous version called NGOMSL; and a version that can model overlapping human activities, CPM-GOMS. To make the comparison, we analyze the same task in each of these variants and then discuss the qualitative and quantitative similarities and differences.

1.1 The Example Task

Throughout this presentation, we use a single example task and present how each GOMS technique represents this task. A GOMS model can, and should, start at a high level of a task such as collaboratively writing a research paper with a coauthor. At such a high level, the subtasks involve many different applications: a word processor to actually write the paper, a graphics application to make figures, bibliographies to look up related work, email to send the drafts back and forth, the operating system used to manage the files, and so forth. This wide inclusion of applications, many of which were not designed with the others in mind, gives a broad perspective on the knowledge people need to accomplish such a complex task. GOMS models can then show how knowledge transfers from one application to another or how much additional time is spent moving information between applications that do not fit together well. However, presenting such a broad task is impossible within the confines of this article, so we will present a very small part of it, editing a paragraph in a word processor (Figure 1), and make reference to the larger task as appropriate.

---

[1]See pages 287–319 in this issue.

In order to understand GOMS models that have arisen in the last decade and the relationships between them, an analyst must understand each of the components of the model (goals, operators, methods, and selection rules), the concept of level of detail, and the different computational forms that GOMS models take. In this section, we will define each of these concepts; in subsequent sections we will categorize existing GOMS models according to these concepts.

Fig. 1. The example task: Editing a marked-up manuscript.

Text editing was the original task used in the development of GOMS and is of course still an important task domain. However, it is incorrect to assume that GOMS is somehow limited to text editing; GOMS is much more general. In fact, nine cases presented in John and Kieras [1996] concern task domains radically different from text editing, as do other published works (e.g., Beard et al. [1996] and Vera and Rosenblatt [1995]). But this familiar domain, with task goals and typical procedures familiar to all readers, makes the best example context to present and compare the different GOMS techniques.

Before presenting the analyses, we define each of the components of the GOMS model and discuss an important distinction between two forms that GOMS models take.

1.2 Definitions of GOMS Components

1.2.1 *Goals.* Goals are what the user has to accomplish. The commonsense meaning of the term applies here; a goal is the "end towards which effort is directed."[2] In the collaborative writing example mentioned above, the highest-level goal is to write the paper. Goals are often broken down into subgoals; all of the subgoals must be accomplished in order to achieve the overall goal. Some subgoals for collaborative writing might be to format the bibliography, send the current draft to the second author, or incorporate marked-up comments into the text file (Figure 1). Expanding the latter, the subgoal could be *EDIT-MANUSCRIPT*, and its subgoals might be *MOVE-TEXT*, *DELETE-PHRASE*, and *INSERT-WORD*. All of the subgoals must be accomplished to accomplish the higher-level goal.

Goals and subgoals are often arranged hierarchically, but a strict hierarchical goal structure is not required. In particular, some versions of GOMS models allow several goals to be active at once, and some versions represent

[2]*Webster's New Collegiate Dictionary,* 1977, page 493.

extremely well practiced behavior in a "flattened" structure that does not contain an explicit hierarchy of subgoals.

1.2.2 *Operators.* An operator is an action performed in service of a goal. Operators can be perceptual, cognitive, or motor acts, or a composite of these. Operators can change the user's internal mental state or physically change the state of the external environment. The important parameters of operators, in particular execution time, are assumed to be independent of how the user or the system got into the current state (i.e., independent of the history of operators). Execution time may be approximated by a constant, by a probability distribution, or by a function of some parameter. For instance, the time to type a word might be approximated by a constant (e.g., the average time for an average word by an average typist) or a statistical distribution, or by a function involving the number of letters in the word and the time to type a single character (which could, in turn, be approximated by a constant or a distribution). The accuracy of execution time predictions obtained from a GOMS model depends on the accuracy of this assumption and on the accuracy of the duration estimates. In our text-editing example, with the goal hierarchy defined above, some operators could be MOVE-MOUSE, CLICK-MOUSE-BUTTON, SHIFT-CLICK-MOUSE-BUTTON, and HIT-DELETE-KEY.

1.2.3 *Methods.* Methods are sequences of operators and subgoal invocations that accomplish a goal. If the goals have a hierarchical form, then there is a corresponding hierarchy of methods. The content of the methods depends on the set of possible operators and on the nature of the tasks represented. One method for accomplishing the goal *DELETE-PHRASE* (in the text editor we are using to write this article) would be to MOVE-MOUSE to the beginning of the phrase, CLICK-MOUSE-BUTTON, MOVE-MOUSE to the end of the phrase, SHIFT-CLICK-MOUSE-BUTTON, and finally, HIT-DELETE-KEY (the *mark-and-delete* method).

1.2.4 *Selection Rules.* There is often more than one method to accomplish a goal. Instead of the *mark-and-delete* method just described, another method for accomplishing the *DELETE-PHRASE* goal in Figure 1 would be MOVE-MOUSE to the end of the phrase, CLICK-MOUSE-BUTTON, and HIT-DELETE-KEY 11 times (the *delete-characters* method). If there is more than one method applicable to a goal, then selection rules are necessary to represent the user's knowledge of which method should be applied. Typically such rules are based on specific properties of the task instance. Selection rules can arise through a user's personal experience with the interface or from explicit training. For example, a user may have a rule for the DELETE-PHRASE goal that says if the phrase is more than eight characters long, then use the *mark-and-delete* method; otherwise use the *delete-characters* method.

1.2.5 *Goals versus Operators: Level of Detail.* It is important to clarify a common point of confusion about goals and operators. The distinction is strictly one of the required level of detail:

> *The difference between a goal and an operator in a GOMS analysis is merely a matter of the level of detail chosen by the analyst. For a goal, the analyst provides a method that uses lower-level operators to specify the details of how it is to be accomplished; in contrast, operators are not broken down any further.*

That is, an analyst will decide that certain user activities do not need to be "unpacked" into any more detail and thus will represent them as operators, while other activities do need to be considered in more detail, so the analyst will represent these in terms of goals with their associated methods. Thus, any particular GOMS analysis assumes a certain grain of analysis, a "stopping point" in the level of detail, chosen to suit the needs of the analysis. Continuing the text-editing example, a GOMS analysis could have only one goal (*EDIT-MANUSCRIPT*) and a few high-level operators (e.g., MOVE-TEXT, DELETE-PHRASE, and INSERT-WORD). Or, if the design situation required a finer level of detail, the analysis could have four goals (*EDIT-MANUSCRIPT*, with *MOVE-TEXT*, *DELETE-PHRASE*, and *INSERT-WORD* as subgoals) and finer-grained operators like MOVE-CURSOR, CLICK-MOUSE-BUTTON, DOUBLE-CLICK-MOUSE-BUTTON, SHIFT-CLICK-MOUSE-BUTTON, and HIT-DELETE-KEY to accomplish these goals.

In principle, the goals and operators of a task could be described at much higher levels (e.g., collaboratively writing a paper) or ever-deeper levels of detail, down to muscle group twitches. However, at any stopping point, the analyst must be sure that it is reasonable to assume that the execution times of the lowest-level operators (*primitive operators*) are constant regardless of the surrounding context (or are a constant function of some given parameter). The times can be estimated from data, often from previous similar tasks found in the literature, and used to predict performance on new tasks. The dual constraints that primitive operators be context free and already estimated leads most GOMS models to stop at the command or keystroke level.

It is not necessary to bring all parts of an analysis down to the same level of primitive operators. In many design situations, different parts of a system or different user tasks may require different levels of scrutiny, and GOMS allows such selective detail of analysis. Starting from the high-level user goals, the analyst expands only those parts of the goal hierarchy as necessary for the questions at hand. Other parts can be expanded later as other questions arise. For instance, in collaborative writing, a GOMS analyst might first chose to expand all the search-functions in the different applications (word-processor, bibliography, email) and weeks later expand on the spell-checking functions. Thus, decomposing goals into subgoals and primitive operators is a very flexible analysis tool that suits many design situations.

1.3 Form of a GOMS Model

Different GOMS models in the literature differ substantially in the basic form and appearance of their methods. There are two basic forms: the *program form* and the *sequence form*.

1.3.1 *Program Form.* A GOMS model in program form is analogous to a parameterized computer program. The methods take any admissible set of task parameters and will execute the corresponding instance of the task correctly. For example, if the *mark-and-delete* method described above was represented in program form, it would take as task parameters the starting and ending locations of the to-be-deleted phrase and when executed would move the mouse to the corresponding locations. Thus, a GOMS model in program form describes how to accomplish a general class of tasks, with a specific instance of the class being represented by a set of values for the task parameters. Typically, such a model will explicitly contain some form of conditional branching and invocations of submethods to accomplish subgoals. The procedural knowledge represented in program form is fixed, but the execution pathway and sequence of operators through the task will depend on the specific properties of the task instance. Once the model is defined, all of the possible tasks can be covered by different execution pathways through the model. Thus, a program form model is a compact, generative[3] description that explicitly represents the knowledge of what features of the task environment the user should attend to and how the user should operate the system to accomplish the task goals.

The program form has the advantage that all procedural knowledge is visible to the analyst. In addition, if many task instances need to be analyzed, the generative nature of the program form allows those tasks to be instantiated quickly, especially if implemented in a running computer program. However, program form has two disadvantages. First, the only way to determine the sequence of operators used in a task instance is to run the model (either by hand or machine) and obtain a trace of the method execution steps. Second, defining and expressing a complete and accurate program form model can be quite time consuming, especially if it is represented as a machine-executable model.

1.3.2 *Sequence Form.* In contrast, the methods in a sequence-form GOMS model contain a fixed sequence of operators for accomplishing a particular task instance. There may be some conditionality and parameters included in the sequence model. For instance, in the text-editing example above, listing the exact operators necessary to delete the phrase indicated in Figure 1 is a GOMS model in sequence form (e.g., MOVE-MOUSE, CLICK-MOUSE-BUTTON, 11 *HIT-DELETE-KEY). A more general sequence model would take the number of characters in the phrase as a parameter and contain an implicit iteration. For example, for the *delete-*

[3]The term *generative* is used analogously to its sense in formal linguistics. The syntax of a language can be represented compactly by a generative grammar, a set of rules for *generating* all of the grammatical sentences in the language.

characters method, there would be a MOVE-MOUSE operator, a CLICK-MOUSE-BUTTON operator, and then the HIT-DELETE-KEY operator would be repeated until there were no more characters in the phrase.

The advantages and disadvantages of the sequence form are the inverse of the program form. That is, the analyst does not have to explicitly define the procedural knowledge for every possible task situation in program-like detail, and the sequence of operators is clearly visible to the analyst. But there may be more information about the structure of the methods than can be captured by the operator sequences for a set of task instances; such unrepresented aspects will not be inspectable. Finally, even though listing the operator sequence for an individual task instance is usually easy, if a large number of task instances are involved, it could be time consuming to construct and evaluate the corresponding large number of sequence form models.

2. COMPARISON OF GOMS TASK ANALYSIS TECHNIQUES

We now apply each technique to the example task, discuss the underlying architectural basis and the ensuing constraints for each technique, and compare and contrast the analysis with that of the other GOMS variants.

2.1 The Keystroke-Level Model

The Keystroke-Level Model (KLM) is the simplest GOMS technique [Card et al. 1980a; 1983, Ch. 8]. To estimate execution time for a task, the analyst lists the sequence of operators and then totals the execution times for the individual operators. In particular, the analyst must specify the method used to accomplish each particular task instance. Other GOMS techniques discussed below predict the method given the task situation, but the KLM does not. Furthermore, the specified methods are limited to being in sequence form and containing only keystroke-level primitive operators. Given the task and the method, the KLM uses preestablished keystroke-level primitive operators to predict the time to execute the task.

The original KLM presentation 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. As presented in CMN, the KLM technique includes a set of five heuristic rules for placing mental operators to account for mental preparation time during a task that requires several physical operators. For example, Rule 0 reads "Insert **M**s in front of all **K**s that are not part of argument strings proper (e.g., text or numbers). Place **M**s in front of all **P**s that select commands (not arguments)" [Card et al. 1983, p. 265].

Subsequent research has refined these six primitive operators, improving the time estimates or differentiating between different types of mental operations [Olson and Olson 1990]. Practitioners often tailor these operators or define new ones to suit their particular user group and interface requirements (e.g., Haunold and 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 [Lane et al. 1993]). Since the original heuristic rules were created primarily for command-based interfaces, they had to be updated for direct-manipulation interfaces. Thus, heuristic Rule 0 should be expanded to "Insert **M**s in front of all **K**s that are not part of argument strings proper (e.g., text or numbers). Place **M**s in front of all **P**s that select commands (not arguments) or that begin a sequence of direct-manipulation operations belonging to a *cognitive unit*."[4]

2.1.1 *Architectural Basis and Constraints.* The KLM is based on a simple underlying cognitive architecture: a serial stage model of human information processing in which one activity is done at a time until the task is complete. All of the human information-processing activity is assumed to be contained in the primitive operators, including internal perceptual and cognitive actions, which are subsumed by black-box Mental (**M**) operators. This restricts the KLM to tasks that can be usefully approximated by a series of operators, with no parallel activities, no interruptions, and no interleaving of goals. Luckily, many single-user computer tasks are usefully approximated with these restrictions. However, these restrictions, along with primitive operators defined to be at the keystroke level, make the KLM impractical for representing an entire high-level task like collaboratively writing a research paper. The next two GOMS variants are more able to handle that task.

2.1.2 *Example KLM.* Figure 2 provides a sample KLM for moving the circled phrase in Figure 1. To construct this model, we used heuristics for placing **M**s that have been updated for mouse-based interfaces [Card et al. 1983, p. 265 and above] and the original operator types and times supplied by Card et al. [1983, p. 264]. Figure 2 also includes illustrative observations that an analyst might make about the model.

Quantitatively, the KLM makes the prediction that this task will take about 14 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). Considering the subgoal structure is an

[4]The concept of a cognitive unit is discussed in Card et al. [1983, p. 268].

Moving text with the *MENU-METHOD*

Description	Operator	Duration (sec)
Mentally prepare by Heuristic Rule 0	M	1.35
Move cursor to beginning of phrase	P	1.10
(no M by Heuristic Rule 1)		
Click mouse button	K	0.20
(no M by Heuristic Rule 0)		
Move cursor to end of phrase	P	1.10
(no M by Heuristic Rule 1)		
Shift-click mouse button		
(one average typing K)	K	0.28
(one mouse button click K)	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)		
Press mouse button	K	0.10
Move cursor to Cut menu item	P	1.10
(no M by Heuristic Rule 1)		
Release mouse button	K	0.10
Mentally prepare by Heuristic Rule 0	M	1.35
Move cursor to insertion point	P	1.10
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)		
Press mouse button	K	0.10
Move cursor to Paste menu item	P	1.10
(no M by Heuristic Rule 1)		
Release mouse button	K	0.10
TOTAL PREDICTED TIME		14.38

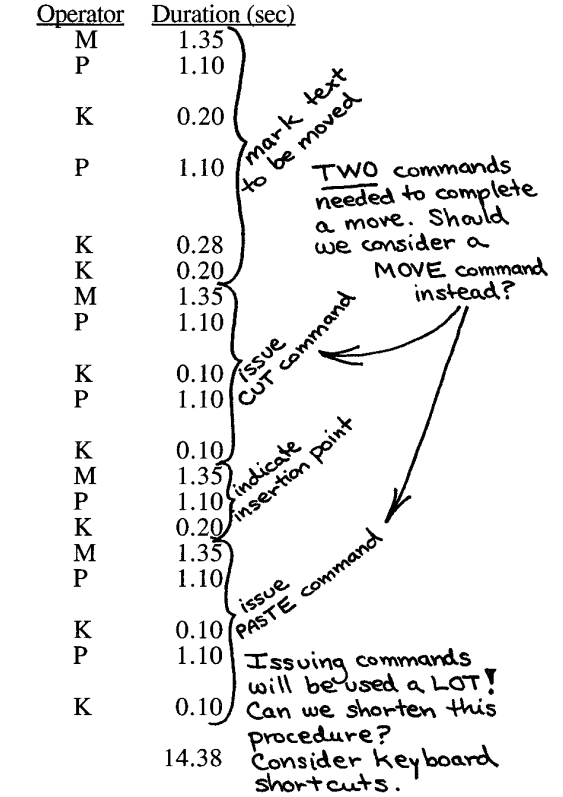

Fig. 2. A Keystroke-Level Model for moving the text in Figure 1. The notes on the right represent handwritten notes an analyst might add to the KLM to highlight ideas.

important use of all GOMS versions, and the next two variants will make it explicit in the model itself.

2.2 Card, Moran, and Newell GOMS (CMN-GOMS)

CMN-GOMS is the term we use to refer to the form of GOMS model presented in Card et al. [1983, Ch. 5] and Card et al. [1980b]. CMN-GOMS has a strict goal hierarchy. Methods are represented in an informal program form that can include submethods and conditionals. A CMN-GOMS model, given a particular task situation, can thus predict both operator sequence and execution time.

Card et al. [1983] do not describe the CMN-GOMS technique with an explicit "how to" guide, but their presentation of nine models at different levels of detail illustrates a breadth-first expansion of a goal hierarchy until the desired level of detail is attained. Card et al. 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. These examples are sufficiently detailed and

extensive that researchers have been able to develop their own CMN-GOMS analyses (e.g., Lerch et al. [1989]).

2.2.1 *Architectural Basis and Constraints.* In the context of the CMN book, it would appear that CMN-GOMS is based on the Model Human Processor (MHP), a simple conventional model of human information processing with parallel stages described by Card et al. [1983, Ch. 2] and summarized in Section 2.6.1. But in fact Card et al. do not establish this relationship and do not derive the GOMS concept from the specific properties of the MHP.

Rather, CMN-GOMS is based on two of the MHP "Principles of Operation," the *Rationality Principle* and the *Problem Space Principle*, both of which developed in the problem-solving theoretical literature (e.g., Newell and Simon [1972] and Card et al. [1983, Ch. 11]). The Problem Space Principle postulates that a user's activity can be characterized as applying a sequence of actions, called *operators*, to transform an initial state into a goal state. With experience, the sequence of operators to accomplish a goal no longer has to be inferred; rather the sequence, termed a *method*, can be routinely recalled and executed when the same goal situation is recognized [Card et al. 1983, Ch. 11]. The Rationality Principle asserts that users will develop methods that are efficient, given the structure of the task environment (i.e., the design of the system) and human processing abilities and limitations. Thus, human activity with a computer system can be viewed as executing methods to accomplish goals, and because humans strive to be efficient, these methods are heavily determined by the design of the computer system. This means that the user's activity can be predicted to a great extent from the system design. Thus, constructing a GOMS model based on the task and the system design can predict useful properties of the human interaction with a computer.

CMN-GOMS, like the KLM, is based on the simple serial-stage architecture, and even though it has program methods and goal structure, no further assumptions are made about how these methods are executed or represented. Consequently, CMN-GOMS is easy to write, but the lack of an explicit description of the method representation and mechanisms involved in task execution means that CMN-GOMS models are relatively vague and unspecified compared to the next two GOMS techniques.

2.2.2 *Example CMN-GOMS Model.* Because the goal hierarchy is explicitly represented, a CMN-GOMS model could start at the level of collaboratively writing a research paper, with subgoals like SEND-DRAFT-TO-CO-AUTHOR, FORMAT-BIBLIOGRAPHY, or EDIT-MANUSCRIPT. Figure 3 displays only those goals and operators at and below the EDIT-MANUSCRIPT subgoal. It includes details for the *MOVE-TEXT* subgoal and illustrative analyst annotations. Moving is accomplishing by first cutting the text and then pasting it. Cutting is accomplished by first selecting the text and then issuing the CUT command. As specified by a selection rule, selecting of text can be done in two different ways, depend-

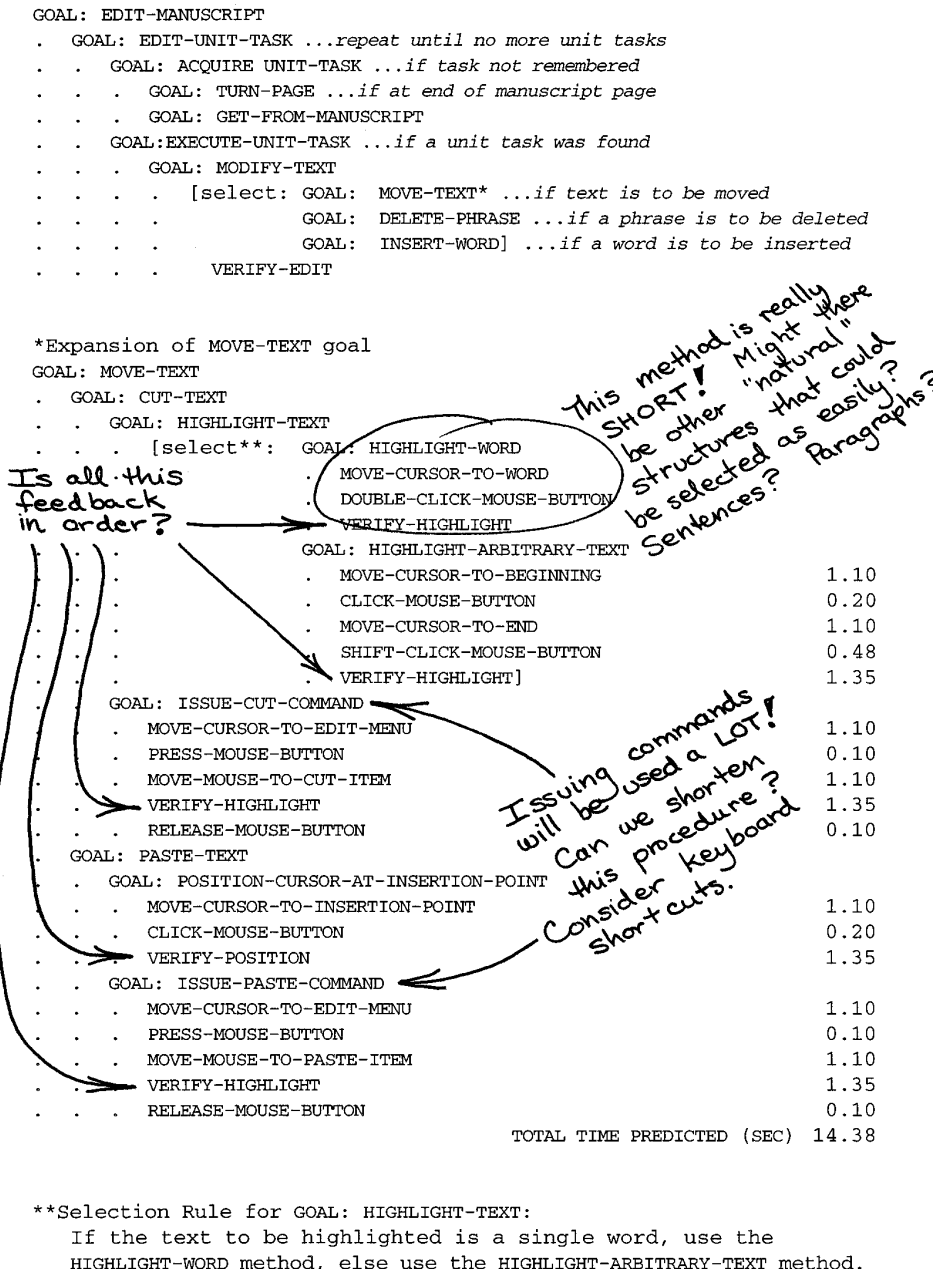

Fig. 3. Example of CMN-GOMS text-editing methods showing the top-level unit-task method structure, an expansion of one method, and a selection rule.

ing on the nature of the text to be selected. Finally pasting requires selecting 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 meth-

ods to accomplish goals; similar methods are easy to see; unusually short or long methods jump out (as annotated) and can spur design ideas. In addition, the annotations indicate that this analyst has observed that the VERIFY operator explicitly records points of feedback to the user.

2.2.3 *Comparison to the KLM.* A major difference between the KLM and the CMN-GOMS models is that CMN-GOMS is in program form; 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. Subgoal invocation and method selection are predicted by the model given the task situation, and these need not be dictated by the analyst as they must for the KLM. Another major difference is that the goal hierarchy is explicit in CMN-GOMS, while it was implicit in the KLM.

Comparing Figure 3 with Figure 2 shows the relationship between CMN-GOMS and the KLM. For instance, there is a one-to-one mapping between the physical operators in the CMN-GOMS model and the **K**s and **P**s in the KLM. The CMN-GOMS model has other operators at this level: VERIFY-LOCATION and VERIFY-HIGHLIGHT, which are not overt physical actions. The KLM has no explicit goals or choices between goals, whereas the CMN-GOMS model represents these explicitly. Roughly, the VERIFY operators, subgoal invocations, and selection rules of the CMN-GOMS model are represented as the **M** operators in the KLM. That is, such operators appear in the CMN-GOMS model in groups that roughly correspond to the placement of **M**s 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.

Given the task specified by the manuscript in Figure 1, this model would predict the trace of operators shown with the estimates of operator times in the far-right column. The estimates for the physical operators are identical to the ones in the KLM. The VERIFY-HIGHLIGHT and VERIFY-POSITION operators are assigned 1.35 seconds, the same value as the KLM's **M** operator because this is Card et al.'s best estimate of mental time in the absence of other information.[5] Thus, the CMN-GOMS model produces the same estimate for task completion as the KLM.

Notice that the CMN-GOMS technique assigns time only to operators, not to any "overhead" required to manipulate the goal hierarchy. In their results, CMN found that time predictions were as good with the assumption that only operators contributed time to the task as they were when goal manipulation also contributed time. However, they suggested that at more detailed levels of analysis such cognitive activity might become more

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

important. Also notice that where the KLM puts **M**s 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, and only appeals to more detailed cognitive architectures can explain the distinction. Pragmatically, however, this difference is irrelevant in most design situations. We will discuss the issue of mental time again after presenting all the GOMS techniques.

2.3 Natural GOMS Language (NGOMSL)

NGOMSL is a structured natural-language notation for representing GOMS models and a procedure for constructing them [Kieras 1988; 1996]. An NGOMSL model is in program form and provides predictions of operator sequence, execution time, and time to learn the methods. An analyst constructs an NGOMSL model by performing a top-down, breadth-first expansion of the user's top-level goals into methods, until the methods contain only primitive operators, typically keystroke-level operators. Like CMN-GOMS, NGOMSL models explicitly represent the goal structure, and so they can represent high-level goals such as collaboratively writing a research paper.

2.3.1 *Architectural Basis and Constraints.* The NGOMSL technique refines the basic GOMS concept by representing methods in terms of a cognitive architecture called *cognitive complexity theory* (CCT) [Bovair et al. 1990; Kieras and Polson 1985]. CCT assumes a simple serial-stage architecture in which working memory triggers production rules that apply at a fixed rate. These rules alter the contents of working memory or execute primitive external operators such as making a keystroke. GOMS methods are represented by sets of production rules in a prescribed format. Learning procedural knowledge consists of learning the individual production rules. Learning transfers from a different task if the rules had already been learned (see also Anderson [1993]). CCT has been shown to provide good predictions of both execution time, learning time, and transfer of procedure learning [Bovair et al. 1988; 1990; Kieras and Bovair 1986].

NGOMSL originated from attempts to define a higher-level notation to represent the content of a CCT model [Bennett et al. 1987; Butler et al. 1989]. It is a structured natural-language notation in which methods are represented in program form as a list of steps which contain operators, both external keystroke-level operators and *internal* operators that represent operations of the CCT architectural mechanisms, such as adding and removing working memory information or setting up subgoals. The relationship between the NGOMSL notation and the CCT architecture is direct: there is essentially a one-to-one relationship between statements in the NGOMSL language and the production rules for a GOMS model written in the CCT format. Therefore, the CCT prediction results can be used by NGOMSL models to estimate not only execution time like KLM and CMN-GOMS, but also the time to learn the procedures.

Although an NGOMSL analysis can provide useful descriptions of a task at many levels of analysis [Karat and Bennett 1991], quantitative predictions of learning and execution times are meaningful only if the methods use operators that the user is assumed to already know and that have known properties. CCT and NGOMSL models have been empirically validated at the keystroke-level of analysis (operators like DETERMINE-POSITION and CLICK-MOUSE-BUTTON); thus models at that level can produce reliable quantitative estimates. In principle, other levels could be researched and empirically validated, but this has not yet been done.

Because NGOMSL models specify methods in program form, they can characterize the procedural complexity of tasks, both in terms of how much must be learned and how much has to be executed. However, the underlying simple serial-stage architecture of CCT limits NGOMSL to hierarchical and sequential methods. Thus, there is no provision for representing methods whose steps could be executed in any order or which could be interrupted and resumed. Also, there is no direct way to represent how perceptual, cognitive, and motor processing might overlap. For example, there is no provision for representing a user doing perceptual processing on an icon while simultaneously homing the hand to the mouse and doing a retrieval from long-term memory. To some extent it is possible to approximate overlapping operations by setting certain operator times to zero (as has been done in Figure 4; see Gong [1993]). Direct representation of processing overlap requires a different underlying cognitive architecture; such an approach is represented by the CPM-GOMS technique, to be discussed next.

2.3.2 *Example NGOMSL Model.* In the text-editing example, Figure 4 shows the NGOMSL methods involved in moving text. Notice that more methods are represented here than are executed in the example task instance.

Quantitatively, NGOMSL provides learning time as well as execution time predictions, discussed in detail below. Qualitatively, NGOMSL provides all that KLM and CMN-GOMS provide, and more. For example, NGOMSL makes the similarity between methods explicit, i.e., all menu commands use a submethod for issuing a command. Like CMN-GOMS, VERIFY operators draw the analyst's attention to feedback. In addition, NGOMSL models explicitly represent working memory and long-term memory usage, allowing the analyst to assess the demands of the design on those cognitive resources. In this example, working memory need only store the command name and the menu name, a reasonable amount of information. This model assumes that users will have learned which commands are in which menus; if they have not they will either systematically search through all the menus or guess. Because these assumptions are explicit, they can be questioned and considered in design.

Learning Time Predictions. NGOMSL models have been shown to be good predictors of time to learn how to use a system, keeping in mind that what is predicted is the *pure learning time* for the *procedural knowledge*

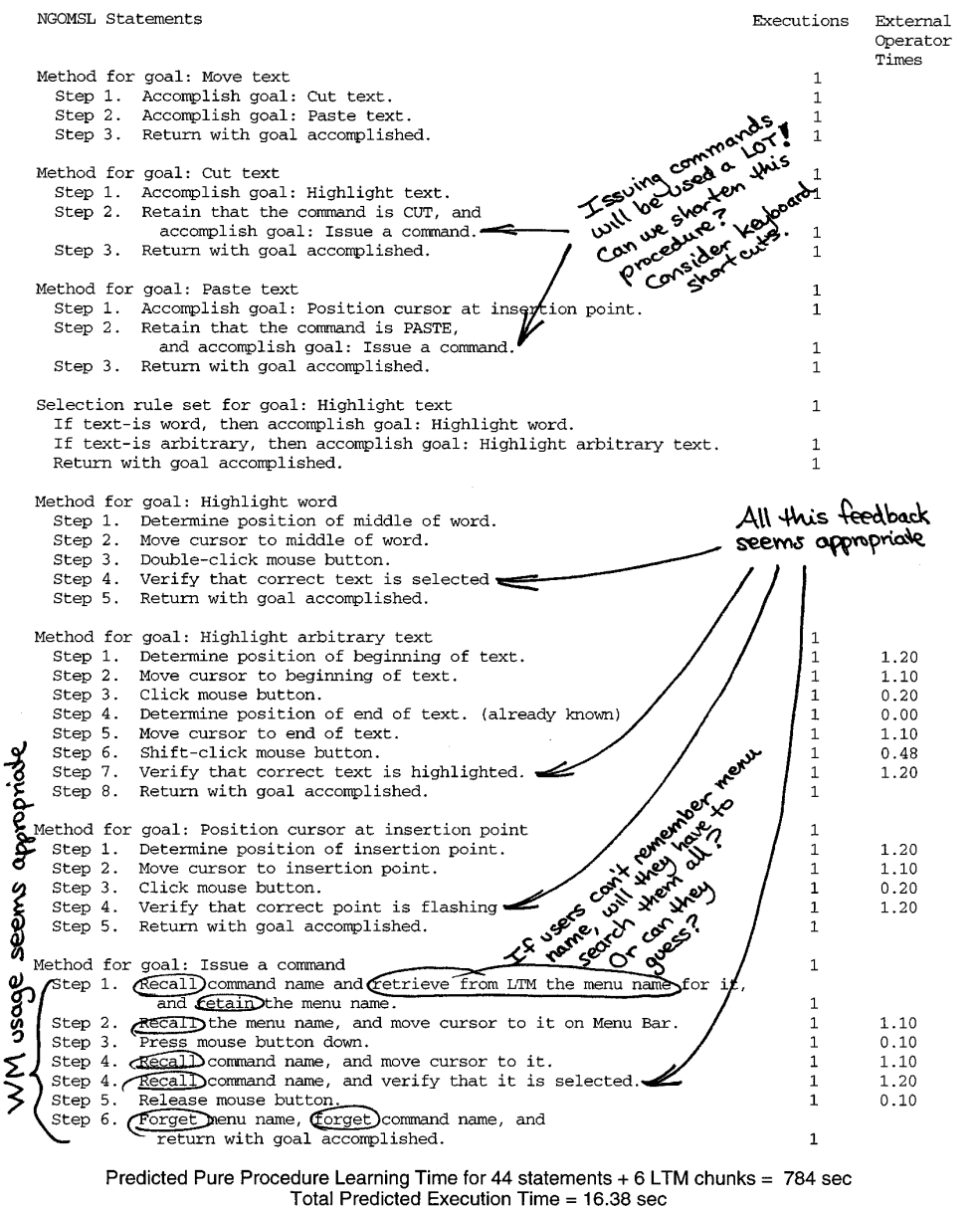

Fig. 4. An example of NGOMSL methods for moving text, showing a generic command-issuing method that uses items in long-term memory to associate menu names to the contained commands. Adapted from Kieras [1996].

represented in the methods. Note that, as mentioned above, the user is assumed to already know how to execute the operators; the GOMS methods do not represent the knowledge involved in executing the operators themselves, but only represent the knowledge of which operators to apply and in what order to accomplish the goal. Innovative interface technology often

results in new operators; moving the cursor with a mouse was a new operator for many users in the early 1980s, and selecting objects with an eye movement tracker or manipulating 3D objects and flying about in virtual space with data-glove gestures will be new operators as these technologies move into the workplace. Clearly, the time to learn how to execute new operators is a critical aspect of the value of new interface devices, but a GOMS model that *assumes* such operators cannot predict their learning time. The time to learn new operators themselves would have to be measured or simply not included in the analysis.

The total elapsed time to learn to use a system depends not only on how much procedural knowledge must be learned but on how much time it takes to complete the training curriculum itself. That is, most learning of computer use takes place in the context of the new user performing tasks of some sort, and this performance would take a certain amount of time even if the user were fully trained. Thus the total learning time consists of the time to execute the training tasks plus the extra time required to learn how to perform the tasks (the *pure* learning time). As Gong [1993] showed, training-task execution times can be estimated from a GOMS model of the training tasks.

The key empirical result is that the procedure learning time is approximately linear with the number of NGOMSL statements that must be learned. Thus, the pure learning time for the methods themselves can be estimated just by counting the statements and multiplying by an empirically determined coefficient. Transfer of training effects can be calculated by deducting the number of NGOMSL statements in methods that are identical, or highly similar, to ones already known to the learner (see Kieras [1988; 1996] and Bovair et al. [1988; 1990]). This characterization of interface consistency in terms of the quantitative transferability of procedural knowledge is perhaps the most significant contribution of the CCT research and the NGOMSL technique. An important limitation of this result is that the accuracy of absolute predictions of learning time will depend on whether the analyst has followed the same "style" in writing the methods as was used to obtain the empirical coefficient. This uncertainty can be dealt with by performing relative comparisons using models written in a consistent style. Further work is needed to describe and document a style for analysts to follow that will yield consistently accurate absolute predictions of learning time.

An additional component of the pure learning time is the time required to memorize chunks of declarative information required by the methods, such as the menu names under which commands are found. Such items are assumed to be stored in long-term memory (LTM), and while not strictly part of the GOMS methods, they are required to be in LTM for the methods to execute correctly.

Including this component in learning time estimates is a way to represent the learning load imposed by menu or command terms, and the heuristics suggested in CMN can be applied to estimate the time to memorize

these items based on the number of chunks. However, heuristics for counting chunks are not very well defined at this time (see Gong [1993]).

The validity and utility of the learning time predictions depend on the general requirements of the learning situation. Clearly, if the learner is engaged in problem solving, or in an unstructured learning situation, the time required for learning is more variable and ill defined than if the learner is trained in a tightly controlled situation. The original work by Kieras, Polson, and Bovair used a mastery-learning situation, in which the users were explicitly trained on the methods and were required to repeatedly execute each procedure fully and exactly before going to the next [Bovair et al. 1990; Kieras and Bovair 1986; Polson 1988]. The CCT predictions were extremely accurate in this sort of learning situation. Gong [1993] used a more realistic learning situation in which users were given a demonstration and explanation and then had to perform a series of training tasks at their own pace, without detailed feedback or correction. The NGOMSL method length, transfer measures, and the number of memory chunks were excellent predictors of this more realistic training time, although the prediction coefficients were different than those in Kieras [1988]. Finally, even in learning situations that are naturalistically unstructured, at least the ordinal predictions of learning time should hold true, as suggested by results such as in Ziegler et al. [1986]. It seems reasonable that regardless of the learning situation, systems whose methods are longer and more complex will require more time to learn, because more procedural knowledge has to be acquired, either by explicit study or inferential problem solving. But clearly more work on the nature of relatively unstructured learning situations is required.

The above discussion of estimating learning time can be summarized as follows, using the values determined by Gong [1993]:

$$\begin{aligned} \textit{Total Procedure Learning Time} = {} & \textit{Pure Procedure Learning Time} \\ & + \textit{Training Procedure Execution Time} \end{aligned}$$

$$\begin{aligned} \textit{Pure Procedure Learning Time} = {} & \textit{NGOMSL Method Learning Time} \\ & + \textit{LTM Item Learning Time} \end{aligned}$$

$$\begin{aligned} \textit{NGOMSL Method Learning Time} = {} & 17\ \textit{seconds} \\ & \times\ \textit{No. of NGOMSL Statements to be Learned} \end{aligned}$$

$$\begin{aligned} \textit{LTM Item Learning Time} = {} & 6\ \textit{seconds} \\ & \times\ \textit{Number of LTM Chunks to be Learned} \end{aligned}$$

These formulas give a pure procedure learning time estimate for the whole set of methods shown in Figure 4 of 784 seconds, in a "typical" learning situation, assuming no prior knowledge of any methods and assuming that

learning the proper command words for the two menu terms will require learning three chunks each.

Execution Time Predictions. Like the other GOMS models, execution time predictions are based on the sequence of operators executed while performing the benchmark tasks. A trace of the example NGOMSL model performing the text-moving example is summarized in Figure 4. The trace includes the same sequence of physical operators as the KLM and CMN-GOMS models in Figures 2 and 3. The predicted execution time is obtained by counting 0.1 seconds for each NGOMSL statement executed and adding the total external operator time, using values based on the KLM recommended in Kieras [1996]. This gives a predicted execution time of 16.38 seconds, which is comparable to the predictions of the other two models (14.38 seconds for both the KLM and CMN-GOMS models).

2.3.3 *Comparison with KLM and CMN-GOMS.* The primary difference between execution time predictions for NGOMSL, KLM, and CMN-GOMS is how time is assigned to cognitive and perceptual operators. There are some stylistic differences in how many large mental operators are assumed; for example, the NGOMSL example follows the recommendations [Kieras 1988; 1996] for the number and placement of DETERMINE-POSITION and VERIFY operators and so has more **M**-like operators than the CMN-GOMS and KLM models. These stylistic differences could be resolved with further research.

A more important difference is in the nature of the unobservable operators. The KLM has a single crude **M** operator that precedes each cognitive unit of action. NGOMSL, because it is based on CCT, uniformly requires some cognitive execution time for every step, manipulating goals and working memory, and for entering and leaving methods. In contrast, CMN-GOMS assigns no time to such cognitive overhead. But all three models include **M**-like operators for substantial time-consuming mental actions such as locating information on the screen and verifying entries. Thus, these methods assign roughly the same time to unobservable perceptual and cognitive activities, but do so at different places in the trace.

2.4 Cognitive-Perceptual-Motor GOMS (CPM-GOMS)

CPM-GOMS, like the other GOMS models, predicts execution time based on an analysis of component activities. However, CPM-GOMS requires a specific level of analysis where the primitive operators are simple perceptual, cognitive, and motor acts. Unlike the other extant GOMS techniques, CPM-GOMS does not make the assumption that operators are performed serially; rather, perceptual, cognitive, and motor operators can be performed in parallel as the task demands. CPM-GOMS uses as *schedule chart* (or *PERT chart*, familiar to project managers; see Stires and Murphy [1962]) to represent the operators and dependencies between operators. The acronym *CPM* stands for both the **C**ognitive-**P**erceptual-**M**otor level of

analysis and the **C**ritical-**P**ath **M**ethod, since the *critical path* in a schedule chart provides the prediction of total task time.

2.4.1 *Architectural Basis and Constraints.* CPM-GOMS is based directly on the Model Human Processor (MHP) (see Card et al. [1983, Ch. 2]), which is a basic human information-processing architecture similar to those appearing in the human cognitive and performance literature for the last few decades. The human is modeled by a set of processors and storage systems in which sensory information is first acquired, recognized, and deposited in working memory by perceptual processors, and then a cognitive processor acts upon the information and commands motor processors to make physical actions. Each processor operates serially internally, with a characteristic cycle time, but processors run in parallel with each other. The unique contribution of CMN was to present this standard picture of human information processing in the form of an engineering model, which by careful simplifications and approximations is able to quantitatively account for many basic phenomena relevant to human-computer interaction (see Card et al. [1983, Ch. 2]). The CPM-GOMS technique directly applies the MHP to a task analysis by identifying the operators that must be performed by each processor and the sequential dependencies between them.

The MHP architecture allows parallelism between CPM-GOMS operators, which is necessary for analyzing some tasks, but it also forces the primitive operators to be at the level of the cycle-times of the MHP's processors. Thus, CPM-GOMS models are much more detailed than previous GOMS variants. As the following example will make clear, CPM-GOMS models are too detailed for tasks that can be usefully approximated by serial operators.

CPM-GOMS models also make an assumption of extreme expertise in the user. That is, they typically model performance that has been optimized to proceed as fast as the MHP and information-flow dependencies will allow. We will discuss the implications of this assumption in the context of the text-editing example, below.

2.4.2 *Example CPM-GOMS Model.* To build a CPM-GOMS model, the analyst begins with a CMN-GOMS model of a task (thereby inheriting all the qualitative information obtained from doing a CMN-GOMS model). The CMN-GOMS model can start at any level but must stop with operators at the *activity level*, primarily high-level perceptual (READ-SCREEN) or motor (ENTER-COMMAND) actions. The analyst continues by dropping to a lower level where these operators are then expressed as goals, which is accomplished by methods containing MHP-level operators. John and Gray [1995] have provided *templates* (assemblies of cognitive, perceptual, and motor operators and their dependencies) for different activities under different task conditions. For instance, Figure 5 contains the template for the READ-SCREEN goal when an eye movement is required, and Figure 6 contains the template for when the user is already looking at the right spot on the display. Each operator in the templates has a duration estimate, or a

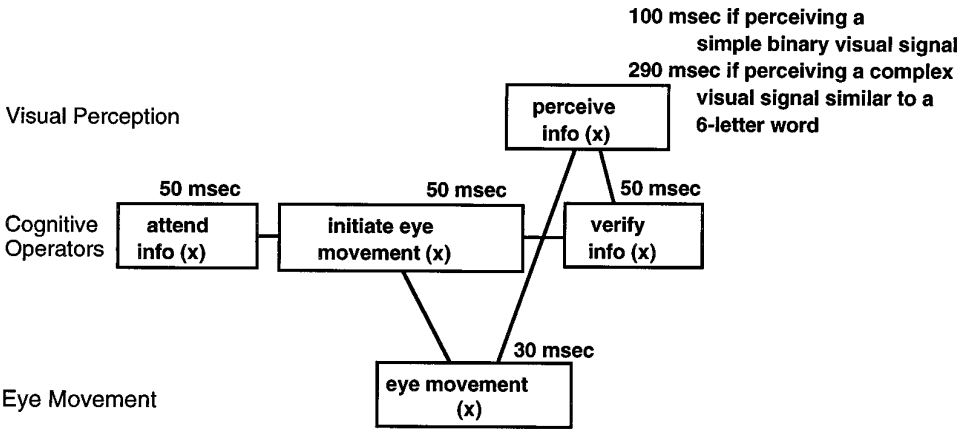

Fig. 5. Example of a template for building CPM-GOMS models adapted from John and Gray [1995]. This template accomplished the goal READ-SCREEN, when an eye movement is required in the task.

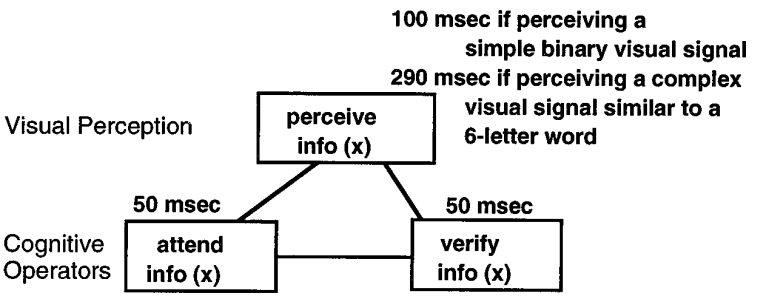

Fig. 6. Example of a template for building CPM-GOMS models adapted from John and Gray [1995]. This template accomplished the goal READ-SCREEN, when an eye movement is *not* required in the task.

set of estimates, that depends on task conditions. For instance, visually perceiving and comprehending a six-character word takes 290ms, whereas visually perceiving and comprehending that a symbol is merely present or absent (e.g., the presence of highlighting) takes 100ms (Figures 5 and 6).

These templates are first joined together serially and then interleaved to take advantage of the parallelism of the underlying cognitive architecture. The operators, their estimates of duration, and the pattern of dependencies between them combine to produce a detailed model of which actions will occur in the performance of the task and when they will happen. The sequence of operators which produces the longest path through this chart is called the *critical path*; the sum of the durations of operators on the critical path estimates the total duration of the task. If empirical data about actual performance of observable motor operators are available from a current system that is similar to the system being designed, it is desirable to verify the model against these data. Then the verified models are modified to

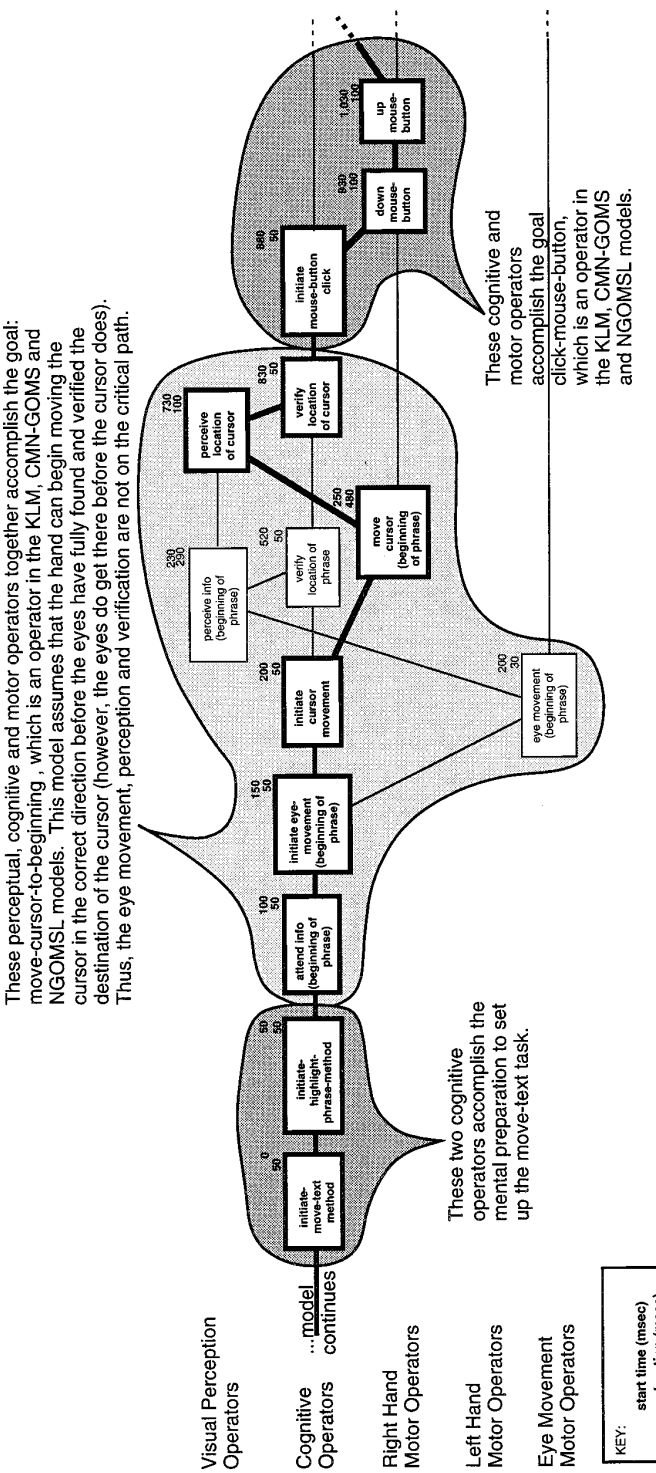

Fig. 7. CPM-GOMS model of a MOVE-TEXT method for the text-editing task in Figure 1.

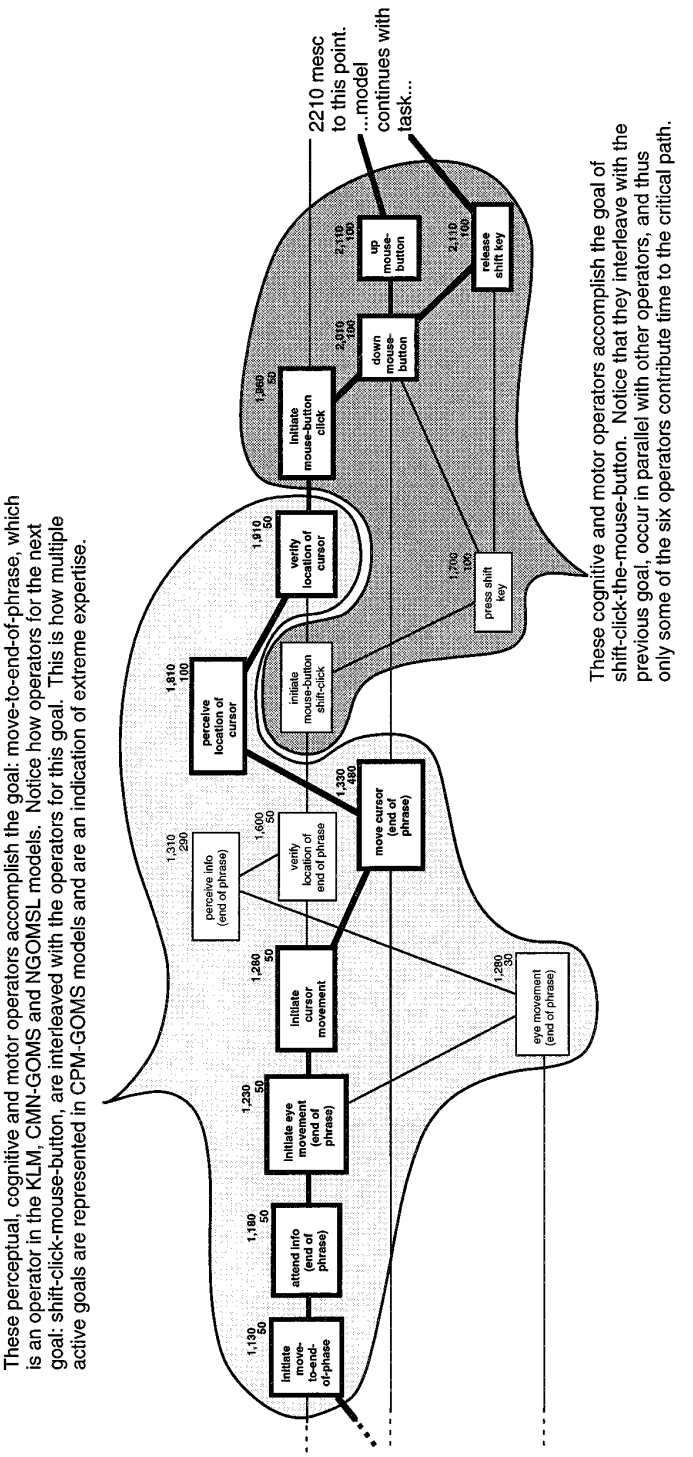

Fig. 7. *Continued*

represent the proposed design, and quantitative predictions of performance time can be determined from the critical path of the CPM-GOMS model. Qualitative analyses of what aspects of a design lead to changes in 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 [Chuah et al. 1994; Gray et al. 1993].

In the example of the MOVE-TEXT goal, Figure 7 shows a CPM-GOMS model of this task. For brevity, the model covers only the portion of the procedure involved with highlighting the text to be moved. Each box in the chart represents an operator, and each horizontal line of boxes represents the sequence of operators executed by a perceptual, cognitive, or motor processor. The lines connecting the boxes indicate sequential dependencies between the operators, and the highlighted lines correspond to the critical path.

Before discussing this example model in detail, it is important to note that text editing is *not* a good application of the CPM-GOMS technique, and we present it here only to compare it to the other GOMS techniques. Text editing is *usefully approximated* by serial processes, which is why the KLM, CMN-GOMS, and NGOMSL have been so successful at predicting performance on text editors. CPM-GOMS is overly detailed for such primarily serial tasks and 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 in Gray et al. [1993] and transcription typing [John 1996; John and Newell 1989].

Execution Time Predictions. In Figure 7, the times for the operators shown on the boxes in the schedule chart are based on the durations estimated by John and Gray [1995]. The highlighted lines and boxes comprise the critical path. Reading the total duration on the final item of the critical path gives a total execution time through this subsequence of the task equal to 2.21 seconds.

The ability of CPM-GOMS to represent parallel processing is illustrated in the set of operators that accomplish the *MOVE-TO-BEGINNING-OF-PHRASE* goal. These operators are not performed strictly serially; that is, the eye movement and perception of information occur in parallel with the cursor being moved to the new location. The information-flow dependency lines between the operators ensure that the eyes must get there first, before the new position of the cursor can be verified to be at the right location; however, the movement of the mouse takes longer than the eye movement and perception, so it defines the critical path.

Multiple active goals can be represented in CPM-GOMS models and are illustrated in Figure 7 in the sets of operators that accomplish the MOVE-TO-END-OF-PHRASE goal and the *SHIFT-CLICK-MOUSE-BUTTON* goal. Because the shift key is hit with the left hand (in this model of a right-handed person), and the mouse is moved with the right hand, pressing the shift-key can occur while the mouse is still being moved to the end of the phrase. Thus, the operators that accomplish the *SHIFT-CLICK-*

MOUSE-BUTTON goal are interleaved with the operators that accomplish the *MOVE-TO-END-OF-PHRASE* goal. This interleaving represents a very high level of skill on the part of the user.

2.4.3 *Comparison with KLM, CMN-GOMS, and NGOMSL.* 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 7 compared to the other example models. 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 because the particular model in Figure 7 was built with reference to the one in Figure 3. As with the KLM, selection rules are not explicitly represented because CPM-GOMS models are in sequence form, and the analyst chooses a particular method for each task instance. For example, in Figure 7, the selection between *HIGHLIGHT-ARBITRARY-PHRASE* and *HIGHLIGHT-WORD* that is explicitly represented in CMN-GOMS and NGOMSL is only implicit in the analyst's choice of the method for this particular model.

Although the qualitative process represented in this CPM-GOMS model is reasonable, its quantitative prediction is much shorter than the estimates from the other models. The CPM-GOMS model predicts the total execution time to be 2.21 seconds; totaling the execution time over the same steps in the other models gives 4.23 seconds for both the KLM and CMN-GOMS and 6.18 seconds for the NGOMSL model. The primary source of the discrepancy between the GOMS variants is the basic assumption in the CPM-GOMS technique that the user is extremely experienced and executes the task as rapidly as the MHP architecture permits.

One aspect of the extreme-expertise assumption is that the CPM-GOMS 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 target phrase and that the mouse movements to these points can be initiated prior to the completion of the eye movements. In some real-world situations, such as telephone operators handling calls [Gray et al. 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, so visual search would be required to locate the target phrase. CPM-GOMS has been used to model visual search processes [Chuah et al. 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 7, the method selection is implicit in a single cognitive operator (INITIATE-MOVE-TEXT-METHOD) which is the minimum cognitive activity required by the MHP to recognize a situation and note it in working memory. 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 required in CMN-GOMS and NGOMSL. Thus, Figure 7 represents a minimum of cognitive activity, which is an unreasonable assumption for a normal text-editing task. However, in an experiment by 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, but after 1100 trials on the *exact same task instance*, 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. 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 commands [John and Newell 1987]. However, the technique for modeling complex decisions in CPM-GOMS is still a research issue, and so it 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 specifically for the particular target size and distance in this situation, yielding much shorter times than CMN's 1.10 second estimate of average pointing time used in the other models (further discussion appears in the next section).

3. SUMMARY AND COMPARISON OF THE GOMS TECHNIQUES

We have modeled the same goal, *MOVE-TEXT*, with four different GOMS task analysis techniques. For purposes of comparison, we included a CPM-GOMS model for the same text-editing task, although the technique is not recommended for modeling such sequential tasks, and for brevity, it was shown only for the text-highlighting submethod.

3.1 Summary Comparison of Predictions

The KLM, CMN-GOMS, and NGOMSL models all produce the same sequence of observable operators, as does the CPM-GOMS model (although at a more detailed level). Table I summarizes the quantitative predictions from the above presentation, both for the overall example task and the subtask consisting just of highlighting the to-be-moved text.

NGOMSL is the only one of the four techniques that makes learning time predictions, and these are limited to the effects of the amount of procedural knowledge and related LTM information to be learned and to learning situations for which the coefficients have been empirically determined. KLM, CMN-GOMS, and NGOMSL produce execution time predictions that are roughly the same for both the overall task and the subtask, although they make different assumptions about unobservable cognitive and perceptual operators and so distribute the time in different ways (see below). An important difference is that the NGOMSL technique currently entails more

Table I. Predicted Time Measures (in Seconds) for Each Technique for the MOVE-TEXT Example

	KLM	CMN-GOMS	NGOMSL	CPM-GOMS
Overall Measures				
Procedure Learning (both highlighting methods)	—	—	784.00	—
Total Example Task Execution Time	14.38	14.38	16.38	not shown in this example
Text Highlighting Submethod				
Highlighting Submethod Execution Time	4.23	4.23	6.18	2.21
Total Cognitive Overhead	—	—	0.90	1.10

M-like operators than the other techniques, as well as some cognitive overhead due to method step execution. Thus, NGOMSL will typically predict execution times that are longer than KLM or CMN-GOMS predictions.

As shown in the execution time predictions for the text-highlighting submethod, the CPM-GOMS model predicts a substantially shorter execution time than the other models. As discussed above, this is due to the assumption of extreme expertise, which produces maximum operator overlapping, finer-grain time estimates for the individual operators, and the minimum of cognitive activity allowed by the MHP. An interesting similarity between NGOMSL and CPM-GOMS is the roughly similar cognitive overhead time in the example submethod; in NGOMSL this value is the statement execution time at 0.1 seconds/statement; in CPM-GOMS it is the total time for which the cognitive processor is on the critical path in Figure 7.

3.2 Summary Comparison of Operator Times

Table II lists the operator times assumed in the different techniques and used in the MOVE-TEXT example. There are basically two types of operators: those that are directly observable when looking at human performance (the motor operators) and those that are not or usually not observable (perceptual and cognitive operators, eye movements). [6] The values for directly observable operators are quite similar across the GOMS techniques, while the assumptions about unobservable operators vary more widely.

Mouse Button Operations. The times for mouse button operators and using the shift-key in KLM, CMN-GOMS, and NGOMSL are based on

[6]Although eye movements are observable with an eye tracker, eye-tracking research in HCI is sparse, and we will treat them as unobservable in this task.

Table II. Operator Times (in Seconds) Used in Each Technique for the MOVE-TEXT Example

	KLM	CMN-GOMS	NGOMSL	CPM-GOMS *critical path*
Directly Observable Motor Operators:				
Click-mouse-button	0.20	0.20	0.20	0.250
Shift-click-mouse-button	0.48	0.48	0.48	0.250
Cursor movement	1.10	1.10	1.10 or Fitts' Law	0.680 by Fitts' Law
Unobservable Perceptual or Cognitive Operators				
Mental Preparation	1.35	not used	not used	0.100
Determine Position	not used	not used	1.20	0.100
Edit Verification	not used	1.35	1.20	not used

values from Card et al. [1983]. The slightly different value for CLICK-MOUSE-BUTTON in the CPM-GOMS technique can be read from the example in Figure 7. That is, clicking the mouse button requires a 50ms cognitive operator and two motor operators at 100ms each.

The SHIFT-CLICK operation is assumed to be the sequence of hitting the shift-key (280msec from CMN) and then the mouse button (200msec) in the first three techniques. However, in CPM-GOMS, the shift-key operator can overlap with earlier processing in the MOVE-TEXT task, so that it is not on the critical path. Thus, the entire SHIFT-CLICK operation adds only 250msec to the critical path (the same as CLICK-MOUSE-BUTTON).

Cursor Movement. The 1.10 seconds used in KLM, CMN-GOMS, and NGOMSL is the average value suggested by CMN 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 much faster than 1.10 seconds, and that Fitts' Law estimates (see Card et al. [1983, p. 55] were more accurate. Thus, Fitts' Law values based on the actual or typical locations and sizes 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 7), and only some of them occur on the critical path in any particular task situation. The duration of the mouse movement motor operator itself was calculated using Fitts' Law (480ms for both movements). In this example, moving to the beginning of the phrase put 680ms on the critical path (2 cognitive, 1 motor, and 1 perceptual in Figure 7) and, coincidentally, moving to the end of the phrase also put 680ms on the critical path (also 2 cognitive, 1 motor, and 1 perceptual).

Unobservable Operations. All of the GOMS variants make assumptions about unobservable operations. The KLM makes the simplest assumption, putting all such operations (perceiving information, eye movements, comparisons, decisions, mental calculations, etc.) into one operator, **M**, 1.35

seconds in length. This operator is always put at the beginning of a cognitive unit. CMN-GOMS and NGOMSL break this catch-all **M** into more specific unobservable operators. CMN-GOMS uses unobservable operators to verify the editing actions (VERIFY-HIGHLIGHT and VERIFY-POSITION), also assigned the estimate of 1.35 seconds; NGOMSL uses DETERMINE-POSITION and VERIFY, both 1.20 seconds. For the KLM, CMN-GOMS, and NGOMSL models, the estimates for the unobservable operators shown are those currently recommended for each technique as average values to be used in the absence of more specific measurements. They are all roughly the same at about a second duration, but are slightly different because they were determined empirically with different data sets at different historical points in the development of GOMS techniques. None of these techniques have a theoretical commitment to any particular value. Any available empirically determined values for the operators involved in a particular analysis should be used instead of these average estimates.

There are also differences in the distribution of mental time. The KLM tends to place mental time in the preparation for action, while CMN-GOMS mental time tends to come at the end of actions in VERIFY operators, and NGOMSL has mental time in both places. These stylistic differences could probably be resolved with further research.

In addition to the **M**-like operators, NGOMSL also takes time for the unobservable activity associated with the production-rule cycling assumed in the underlying architecture and represented with the 0.1-second/statement "cognitive overhead." In considerably more detail, CPM-GOMS also represents the underlying unobserved operations in terms of the cycle times of the MHP processors, such as the cognitive cycle time (estimated at 70ms by CMN, but refined by subsequent work to be 50ms [John and Newell 1989; Nelson et al. 1994; Wiesmeyer 1992], perceptual cycle time (which depends on the complexity of the signal being perceived; see Figures 5 and 6), and eye movement time (estimated to be 30msec) [Card et al. 1983, p. 25]. Both the duration and dependencies of these unobservable operators are specified in the templates used to construct the model. However, the other operators needed to accomplish a task and their dependencies make every critical path different, and no one estimate of "mental time" is meaningful in CPM-GOMS. For example, in the MOVE-TEXT task in Figure 7, the entry in Table II for Mental Preparation is the sum of the durations of the two cognitive operators on the critical path that set up the MOVE-TEXT task and HIGHLIGHT-PHRASE subtask. The entry for Determine Position is the sum of the durations of those operators that locate the beginning of the phrase on the screen that occur on the critical (3 cognitive operators, 1 eye movement motor operator, and 1 perceptual operator). All of these operators depend on each other and have to occur in order; thus, if this were the only activity taking place in a task, they would all be on the critical path and take 420ms. However, since looking for the beginning of the phrase is just one part of the MOVE-TEXT task, other activities can occur in parallel (e.g., moving the mouse, discussed in the last section), and their operators are interleaved with these,

making the critical path more complicated, so that only the first two cognitive operators appear on the critical path for this task.

3.3 Summary Comparison of Architectural Assumptions

The assumed cognitive architectures range from the trivial, in the case of the KLM, to slightly more complicated for CMN-GOMS, to an elaborated sequential architecture with a working memory and specified procedure knowledge representation in NGOMSL, to a powerful but relatively unspecified multiple parallel processor architecture in CPM-GOMS. The strengths and weaknesses of the techniques correspond quite directly to these architectural differences. The KLM is easy to apply, but predicts only execution time, and only from analyst-supplied methods. At the other extreme, CPM-GOMS predicts execution time for subtle, overlapping patterns of activities, but also requires analyst-supplied methods. CMN-GOMS, once its program methods have been worked out, can predict execution time for all subsumed task instances, and NGOMSL, with the additional investment in its explicit representation of procedural knowledge, can then also predict some aspects of learning time. Thus, rather than being radically different, the GOMS techniques occupy various points in a space of possible techniques defined by different architectural assumptions and the form of the methods supplied by the analyst (see John and Kieras [1994] for more discussion). Some important possibilities for research lie in the gaps in this space; for example, the extant set of ready-to-use GOMS techniques lack a program form approach to analyzing overlapping cognitive, perceptual, and motor activities.

4. CONCLUSIONS

The four specific GOMS modeling techniques discussed here are all related to a general task-analysis approach. This general approach emphasizes the importance of the procedures for accomplishing goals that a user must learn and follow in order to perform well with the system. By using descriptions of user procedures, the techniques can provide quantitative predictions of procedure learning and execution time and qualitative insights into the implications of design features. While other aspects of system design are undoubtedly important, the ability of GOMS models to address this critical aspect makes them not only a key part of the scientific theory of human-computer interaction, but also useful tools for practical design [John and Kieras 1996].

The current GOMS models are quite effective because they capture procedural speed and complexity. But other aspects of human performance with an interface are not addressed by the simple cognitive architectures underlying the current GOMS variants. Current research in cognitive architectures assumes both more detail and more variety of human mechanisms, and so can potentially account for a wider and deeper range of design issues. Representative examples of such work use architectures that represent perceptual-cognitive-motor interactions [Kieras and Meyer 1996;

Kieras et al. 1995; Nelson et al. 1994], comprehension processes [Doane et al. 1992; Kitajima and Polson 1992], and problem-solving and learning mechanisms [Altmann et al. 1995; Anderson 1993; Bauer and John 1995; Howes 1994; Polson and Lewis, 1990; Rieman et al. 1994].[7] Because these research efforts are rigorous and make use of computational models, they should eventually lead to engineering-style design tools for additional aspects of interface design and usability. Thus while the current generation of GOMS models are ready for application, we can expect to see future models of human-computer interaction that are even more comprehensive, accurate, and useful.

ACKNOWLEDGMENTS

The authors contributed equally to this article; the order of their names reflects alphabetical order and not seniority of authorship. We thank Wayne Gray and Judy Olson for their comments on drafts of this article.

REFERENCES

ALTMANN, E. M., LARKIN, J. H., AND JOHN, B. E. 1994. Display navigation by an expert programmer: A preliminary model of memory. In *Human Factors in Computing Systems* (*CHI' 1995*). ACM, New York, 3–10.

ANDERSON, J. R. 1993. *Rules of the Mind.* Lawrence Erlbaum, Hillsdale, N.J.

BAUER, M. I. AND JOHN, B. E. 1994. Modeling time-constrained learning in a highly-interactive task. In *Human Factors in Computing Systems* (*CHI '1995*). ACM, New York, 19–26.

BEARD, D. V., SMITH, D. K., AND DENELSBECK, K. M. 1996. Quick and dirty GOMS: A case study of computed tomography interpretation. *Hum. Comput. Interact. 11.* To be published.

BENNETT, J. L., LORCH, D. J., KIERAS, D. E., AND POLSON, P. G. 1987. Developing a user interface technology for use in industry. In *Proceedings of the 2nd IFIP Conference on Human-Computer Interaction* (*INTERACT '87*), H. J. Bullinger and B. Shackel, Eds. Elsevier Science Publishers B. V., North-Holland, Amsterdam, 21–26.

BOVAIR, S., KIERAS, D. E., AND POLSON, P. G. 1988. The acquisition and performance of text-editing skill: A production-system analysis. Tech. Rep. No. 28. Technical Communication Program, Univ. of Michigan, Ann Arbor, Mich.

BOVAIR, S., KIERAS, D. E., AND POLSON, P. G. 1990. The acquisition and performance of text editing skill: A cognitive complexity analysis. *Hum. Comput. Interact. 5,* 1–48.

BUTLER, K. A., BENNETT, J., POLSON, P., AND KARAT, J. 1989. Report on the workshop on analytical models: Predicting the complexity of human-computer interaction. *SIGCHI Bull. 20,* 4, 63–79.

CARD, S. K., MORAN, T. P., AND NEWELL, A. 1980a. The keystroke-level model for user performance time with interactive systems. *Commun. ACM 23,* 7 (July), 396–410.

CARD, S. K., MORAN, T. P., AND NEWELL, A. 1980b. Computer text-editing: An information-processing analysis of a routine cognitive skill. *Cog. Psychol. 12,* 32–74.

CARD, S. K., MORAN, T. P., AND NEWELL, A. 1983. *The Psychology of Human-Computer Interaction.* Lawrence Erlbaum, Hillsdale, N.J.

CHUAH, M. C., JOHN, B. E., AND PANE, J. 1994. Analyzing graphic and textual layouts with GOMS: Results of a preliminary analysis. In *Human Factors in Computing Systems* (*CHI '1994*). ACM, New York, 323–324.

[7]A special issue on cognitive architectures for HCI is scheduled to appear in *Human-Computer Interaction*, 1997).

DOANE, S. M., MANNES, S. M., KINTSCH, W., AND POLSON, P. G. 1992. Modeling user action planning: A comprehension based approach. *User Model. User-Adapted Interact. 2,* 249–285.

GONG, R. 1993. Validating and refining the GOMS model methodology for software user interface design and evaluation. Ph.D. dissertation, Univ. of Michigan, Ann Arbor, Mich.

GRAY, W. D., JOHN, B. E., AND ATWOOD, M. E. 1993. Project Ernestine: A validation of GOMS for prediction and explanation of real-world task performance. *Hum. Comput. Interact. 8,* 3, 237–209.

HAUNOLD, P. AND KUHN, W. 1994. A keystroke level analysis of a graphics application: Manual map digitizing. In *Human Factors in Computing Systems* (*CHI '1994*). ACM, New York, 337–343.

HOWES, A. 1994. A model of the acquisition of menu knowledge by exploration. In *Human Factors in Computing Systems* (*CHI '1994*). ACM, New York, 445–451.

JOHN, B. E. 1996. TYPIST: A theory of performance in skilled typing. *Hum. Comput. Interact. 11.* To be published.

JOHN, B. E. AND GRAY, W. D. 1995. CPM-GOMS: An analysis method for tasks with parallel activities. In *Human Factors in Computing Systems* (*CHI '1995*). ACM, New York, 393–394.

JOHN, B. E. AND KIERAS, D. E. 1994. The GOMS family of analysis techniques: Tools for design and evaluation. Tech. Rep. CMU-CS-94-181, School of Computer Science, Carnegie-Mellon Univ., Pittsburgh, Pa.

JOHN, B. E. AND KIERAS, D. E. 1996. Using GOMS for user interface design and evaluation: Which technique? *ACM Trans. Comput. Hum. Interact. 3,* 4 (Dec.), 287–319. This issue.

JOHN, B. E. AND NEWELL, A. 1987. Predicting the time to recall computer command abbreviations. In *Human Factors in Computing Systems* (*CHI '1987*). ACM, New York, 33–40.

JOHN, B. E. AND NEWELL, A. 1989. Cumulating the science of HCI: From S-R compatibility to transcription typing. In *Human Factors in Computing Systems* (*CHI '1989*). ACM, New York, 109–114.

KARAT, J. AND BENNETT, J. 1991. Modeling the user interaction methods imposed by designs. In *Human Factors in Information Technology.* Vol. 2, *Mental Models and Human-Computer Interaction,* M. Tauber and D. Ackermann, Eds. North-Holland, Amsterdam.

KIERAS, D. E. 1988. Towards a practical GOMS model methodology for user interface design. In *The Handbook of Human-Computer Interaction.* North-Holland, Amsterdam, 135–158.

KIERAS, D. E. 1996. A Guide to GOMS model usability evaluation using NGOMSL. In *The Handbook of Human-Computer Interaction.* 2nd ed. North-Holland, Amsterdam. To be published.

KIERAS, D. E. AND BOVAIR, S. 1986. The acquisition of procedures from text: A production-system analysis of transfer of training. *J. Mem. Lang. 25,* 507–524.

KIERAS, D. E. AND MEYER, D. E. 1996. An overview of the EPIC architecture for cognition and performance with application to human-computer interaction. *Hum. Comput. Interact. 11.* To be published.

KIERAS, D. E. AND POLSON, P. G. 1985. An approach to the formal analysis of user complexity. *Int. J. Man-Machine Stud. 22,* 365–394.

KIERAS, D. E., WOOD, S. D., AND MEYER, D. E. 1995. Predictive engineering models using the EPIC architecture for a high-performance task. In *Human Factors in Computing Systems* (*CHI '1995*). ACM, New York, 11–18.

KITAJIMA, M. AND POLSON, P. G. 1992. A computational model of skilled use of a graphical user interface. In *Human Factors in Computing Systems* (*CHI '1992*). ACM, New York, 241–249.

LANE, D. M., NAPIER, H. A., BATSELL, R. R., AND NAMAN, J. L. 1993. Predicting the skilled use of hierarchical menus with the Keystroke-Level Model. *Hum. Comput. Interact. 8,* 2, 185–192.

LERCH, F. J., MANTEI, M. M., AND OLSON, J. R. 1989. Translating ideas into action: Cognitive analysis of errors in spreadsheet formulas. In *Human Factors in Computing Systems* (*CHI '1989*). ACM, New York, 121–126.

NELSON, G. H., LEHMAN, J. F., AND JOHN, B. E. 1994. Integrating cognitive capabilities in a real-time task. In *Proceedings of the 16th Annual Conference of the Cognitive Science Society.* Cognitive Science Society, Pittsburgh, Pa.

NEWELL, A. AND SIMON, H. A. 1972. *Human Problem Solving.* Prentice-Hall, Englewood Cliffs, N.J.

OLSON, J. R. AND OLSON, G. M. 1990. The growth of cognitive modeling in human-computer interaction since GOMS. *Hum. Comput. Interact. 5,* 221–265.

POLSON, P. G. 1988. Transfer and retention. In *Cognitive Science and Its Application for Human-Computer Interaction,* R. Guindon, Ed. Lawrence Erlbaum, Hillsdale, N.J., 59–162.

POLSON, P. AND LEWIS, C. 1990. Theory-based design for easily learned interfaces. *Hum. Comput. Interact. 5,* 191–220.

RIEMAN, J., LEWIS, C., YOUNG, R. M., AND POLSON, P. G. 1994. "Why is a Raven like a writing desk?" Lessons in interface consistency and analogical reasoning from two cognitive architectures. In *Human Factors in Computing Systems* (*CHI '1994*). ACM, New York, 438–444.

STIRES, D. M. AND MURPHY, M. M. 1962. *PERT* (*Program Evaluation and Review Technique*) CPM (*Critical Path Method*). Materials Management Inst., Boston, Mass.

WIESMEYER, M. D. 1992. An operator-based model of human covert visual attention. Ph.D. thesis, Univ. of Michigan, Ann Arbor, Mich.

VERA, A. H. AND ROSENBLATT, J. K. 1995. Developing user model-based intelligent agents. In *Proceedings of the 17th Annual Conference of the Cognitive Science Society,* J. D. Moore and J. F. Lehman, Eds. Lawrence Erlbaum, Hillsdale, N.J., 500–505.

ZIEGLER, J. E., HOPPE, H. U., AND FAHNRICH, K. P. 1986. Learning and transfer for text and graphics editing with a direct manipulation interface. In *Proceedings of CHI '1986.* ACM, New York.

Received September 1994; revised October 1995 and June 1996; accepted June 1996