

Predictive Engineering Models Based on the EPIC Architecture for a Multimodal High-Performance Human-Computer Interaction Task

DAVID E. KIERAS, SCOTT D. WOOD, and DAVID E. MEYER
University of Michigan

Engineering models of human performance permit some aspects of usability of interface designs to be predicted from an analysis of the task, and thus they can replace to some extent expensive user-testing data. We successfully predicted human performance in telephone operator tasks with engineering models constructed in the EPIC (**E**xecutive **P**rocess-**I**nteractive **C**ontrol) architecture for human information processing, which is especially suited for modeling multimodal, complex tasks, and has demonstrated success in other task domains. Several models were constructed on an *a priori* basis to represent different hypotheses about how operators coordinate their activities to produce rapid task performance. The models predicted the total task time with useful accuracy and clarified some important properties of the task. The best model was based directly on the GOMS analysis of the task and made simple assumptions about the operator's task strategy, suggesting that EPIC models are a feasible approach to predicting performance in multimodal high-performance tasks.

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

General Terms: Human Factors

Additional Key Words and Phrases: Cognitive models, usability engineering

1. INTRODUCTION

Engineering models for human performance permit some aspects of user interface designs to be evaluated analytically for usability, without consuming resources for empirical user testing, by making usability predic-

This work was supported by the Office of Naval Research Cognitive Sciences Program under grant N00014-92-J-1173 and by NYNEX Science and Technology, Inc., who provided the data videotapes and support for the digitization and transcription of the protocols.
Authors' addresses: D. E. Kieras and S. D. Wood, Artificial Intelligence Laboratory, Electrical Engineering and Computer Science Department, University of Michigan, 1101 Beal Avenue, Ann Arbor, MI 48109-2110; email: kieras@eecs.umich.edu; D. E. Meyer, Department of Psychology, University of Michigan, 525 East University, Ann Arbor, MI 48109-1109.
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.
© 1997 ACM 1073-0516/97/0900–0230 $03.50

tions based on an analysis of the user's task in conjunction with principles and parameters of human performance [Card et al. 1983; John and Kieras 1996a]. This article, an expansion of Kieras et al. [1995], reports results on a new class of engineering models for a multimodal high-performance HCI task, namely the telephone operator tasks studied by Gray et al. [1993] in *Project Ernestine*. By "high performance" we mean that the task is time-stressed; the total execution time must be minimized, and the user of the workstation (the telephone operator) is well-practiced. By "multimodal" we mean that the task engages multiple perceptual-motor modalities: both visual and auditory perception and both vocal and manual motor systems. Such tasks are scientifically interesting because the multiple modalities involve the overall human cognitive and performance system, and because they are *active system tasks* [John and Kieras 1996a; 1996b] in that the user must respond to events produced by the external environment, unlike *passive system* text editing, which is basically paced by the user. As pointed out by John and Kieras [1996a; 1996b], engineering models for active system tasks are currently underdeveloped. Finally, predicting performance in such tasks can be economically important; a detailed information-processing analysis of telephone operator tasks, the Gray et al. [1993] CPM-GOMS models, were of considerable economic value in this domain where a second's reduction in average task completion time represents significant financial savings.

1.1 Background on CPM-GOMS

Since the CPM-GOMS methodology and its most noteworthy application in Project Ernestine [Gray et al. 1993] is the precursor to the present work, some background is important to make the contribution of the present work clear (see also John and Kieras [1996a; 1996b] for a general discussion of CPM-GOMS and other GOMS methodologies). CPM-GOMS is based on the Model Human Processor (MHP) [Card et al. 1983], which is a proposal for how human information processing is performed by a set of perceptual and motor processors surrounding a cognitive processor; these processors operate in parallel with each other. During performance of a task, the human engages in perceptual, cognitive, and motor activities; but since these activities can overlap each other in time, the total time to execute the task is often less than the total of the times for the individual activities. Predicting the time required to execute the task thus requires determining which individual perceptual, cognitive, and motor activities are overlapped.

In the CPM-GOMS methodology, the analyst constructs a schedule chart (PERT chart) to represent the temporal dependencies between the various sequential and parallel activities. Once this network of activities is constructed, the predicted execution time between the very first and the very last activity is the total of the times on the *critical path* through the network, which is the longest duration pathway along the dependencies between the task start and completion. The critical path can then be examined to find out which specific activities determine the time required to complete the task.

However, the practical problem with the CPM-GOMS methodology is that constructing the schedule charts required to analyze an interface design is quite labor-intensive. The analysis is performed on a set of specific benchmark task scenarios, or benchmark *task instances*. For each task instance and interface design, the interface analyst must choose the particular hypothetical pattern of perceptual, cognitive, and motor activities and construct the schedule chart that shows which MHP processors are active in what order and which processor actions depend on which other actions. Of course, the analyst may be able to reuse large portions of the schedule charts; for example, alternative designs or tasks that involve only small variations can be represented just by rearranging portions of the schedule charts (as in the Project Ernestine models). But due to the work involved, the CPM-GOMS method is recommended for predicting execution time only when there is a small number of benchmark tasks to be analyzed (see John and Kieras [1996a]).

1.2 Generative Models of Procedures Using the EPIC Architecture

This article presents a new family of engineering models that are more powerful and easier to apply than CPM-GOMS analysis. These models are based on the EPIC (**E**xecutive **P**rocess-**I**nteractive **C**ontrol) human information-processing architecture developed by Kieras and Meyer [Kieras and Meyer 1997; Meyer and Kieras 1997a; 1997b], and the earlier so-called Cognitive Complexity Theory (CCT) production-system analysis of human-computer interaction [Bovair et al. 1990; Kieras and Polson 1985]. EPIC is similar in spirit to the Model Human Processor (MHP) [Card et al. 1983], but EPIC incorporates many recent theoretical and empirical results about human performance, in the form of a computer simulation modeling software framework. Using EPIC, a *generative*[1] model can be constructed that represents the general procedures required to perform a complex multimodal task as a set of production rules. When the model is supplied with the external stimuli corresponding to a specific task instance, it will then execute the procedures in whatever specific way the task instance requires, thus simulating a human performing the task and *generating* the predicted actions and their time course. Such a model is also typically *reactive* (see John and Kieras [1996a; 1996b], in that the procedural knowledge in an EPIC model not only generates actions depending on the specific task situation, but also reacts in simulated real time to events initiated by the task environment.

The primary goal in the development of EPIC has been to account for human multiple-task performance in situations such as aircraft cockpit tasks. In these situations, the human has to perform two or more highly reactive tasks simultaneously, meeting constraints such as enforcing the relative priority of the tasks and performing at maximum speed consistent

[1]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.

with accuracy. Multiple perceptual and motor modalities are usually involved. Despite the practical importance of such tasks, the empirical and theoretical understanding of them has been quite limited. Nevertheless, EPIC models have been successful at accounting for performance in laboratory versions of multiple-task situations with unprecedented accuracy [Kieras and Meyer 1997; Meyer and Kieras 1997a; 1997b].

1.3 Can EPIC Models Predict Performance?

The work reported in this article shows further that the EPIC framework can go beyond providing a scientific account of laboratory tasks to providing engineering-style predictions of performance in a real-world task domain. The telephone operator task was chosen for this study, although it is a single-task situation, because (1) it involves multiple modalities, (2) the design goal is performance speed, which EPIC currently characterizes well, (3) the previous Project Ernestine work showed that the task domain is tractable, and (4) the original data from Project Ernestine, consisting of videotape recordings of actual operator performance, were available.

If a generative model based on EPIC can be applied to predicting execution time in a high-performance task, it should be considerably more efficient than the CPM-GOMS approach. Preliminary work with an EPIC model of the telephone operator tasks was encouraging, showing fairly good accuracy in predicting task and event times for a very small set of task instances. However, this preliminary model was constructed in a "scientific" mode, in which the model was developed iteratively to provide a good fit to a single protocol, and was then validated against two other protocols. But for an engineering model to be most useful, it should be accurate in an a priori mode, requiring little or no "tuning" based on empirical task observation. For a priori models to succeed, they must be constructed under constraints, here termed *modeling policies*, that govern how the capabilities of the architecture should be applied in representing a specific class of tasks.

Thus the work reported here investigated the extent to which accurate predictions could be made with predictive EPIC models that are based on a priori task analysis and principles of construction. The following topics are treated in the remaining sections of this article: the EPIC architecture, the telephone operator task modeled in this work, general issues of modeling and the concept of modeling policies, a set of a priori models constructed under the modeling policies, and finally a comparison of the EPIC model predictions to a newly analyzed set of performance protocols for the telephone operator task.

2. THE EPIC ARCHITECTURE

Figure 1 shows the overall structure of processors and memories in the EPIC architecture. Although at this level EPIC bears a superficial resemblance to earlier frameworks for human information processing, EPIC incorporates a new synthesis of theoretical concepts and empirical results,

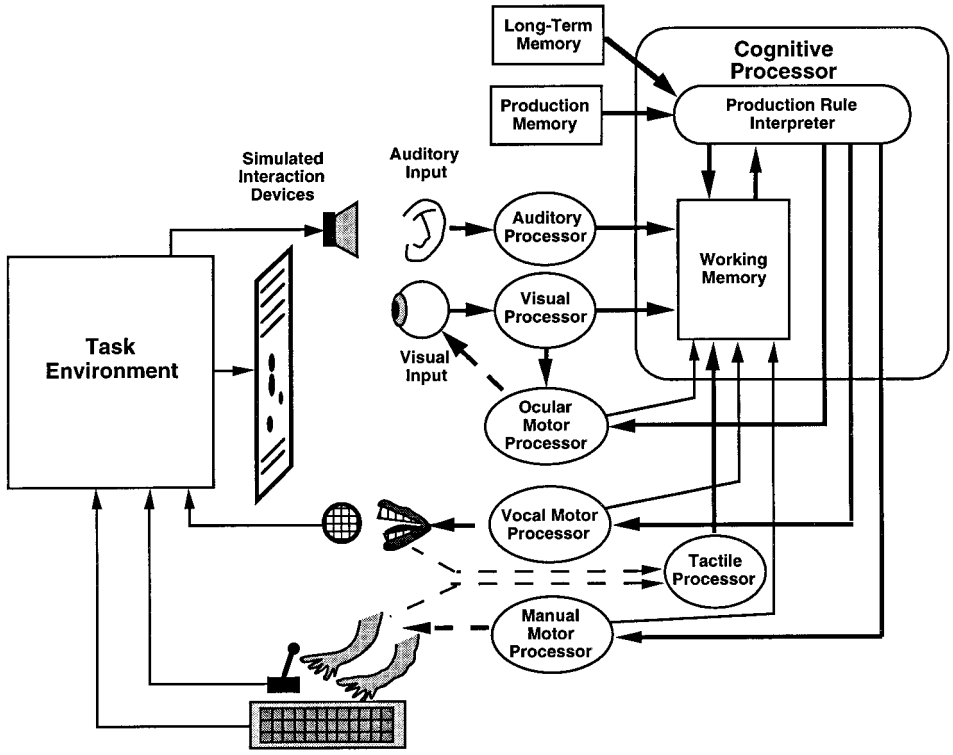


Fig. 1. Overall structure of the EPIC architecture simulation system. Task performance is simulated by having the EPIC model for a simulated human (on the right) interact with a simulated task environment (on the left) via a simulated interface between sensory and motor organs and interaction devices. The EPIC architecture is shown with information flow paths as solid lines and mechanical control or connections as dotted lines. The processors run independently and in parallel with both each other and the task environment module.

and so is more comprehensive, more formalized, and more detailed than proposals such as MHP, HOS, SAINT, and so forth (see McMillan et al. [1989]). It is important to note that EPIC was used "as is" for the modeling work reported here; the details and parameters of the architecture had been developed in other task domains and modeling projects.

EPIC was designed to explicitly couple basic information processing and perceptual-motor mechanisms like those in the MHP with a cognitive analysis of procedural skill, namely that represented by production system models such as CCT [Bovair et al. 1990], ACT-R [Anderson 1993], and SOAR [Laird et al. 1986]. Thus, EPIC has a production rule cognitive processor surrounded by perceptual-motor peripherals; applying EPIC to a task situation requires specifying *both* the production rule programming for the cognitive processor and the relevant perceptual and motor-processing parameters. EPIC computational task models are generative in that the production rules supply general procedural knowledge of the task, and thus when EPIC interacts with a simulated task environment, the EPIC model generates the specific sequence of serial and parallel human actions

required to perform the specific tasks. Thus, the task analysis reflected in the model is general to a class of tasks, rather than reflecting specific task scenarios.

The software for constructing EPIC models includes not only the modules for simulating a human, but also facilities for simulating the interaction of the human with an external system such as a computer. Figure 1 shows at the left a simulated task environment and on the right a simulated human as described by the EPIC architecture, with objects such as screen items and keys making up the physical interface between them. The *task environment module* assigns physical locations to the interface objects and generates visual events and sounds that the computer or other entities in the environment produce in response to the simulated human's behavior. Having a separate environment simulation module greatly simplifies the programming of a complete simulation and helps enforce the generality of the procedural knowledge represented in the EPIC model. That is, the task environment module is driven by a task instance description that consists only of the sequence and timing of events external to the human user, and the simulated user must deal with whatever happens in the simulated task environment.

With regard to the EPIC architecture itself as shown in Figure 1, there is a conventional flow of information from sense organs, through perceptual processors, to a cognitive processor (consisting of a production rule interpreter and a working memory), and finally to motor processors that control effector organs. As do CPM-GOMS models, EPIC goes beyond the MHP by specifying (1) separate perceptual processors with distinct processing time characteristics for each sensory modality and (2) separate motor processors for vocal, manual, and oculomotor (eye) movements. There are feedback pathways from the motor processors, as well as tactile feedback from the effectors, which are important in coordinating multiple tasks. The declarative/procedural knowledge distinction of the "ACT-class" cognitive architectures (e.g., Anderson [1976]) is represented in the form of separate permanent memories for production rules and declarative information. Working memory (WM) contains all of the temporary information needed for and manipulated by the production rules, including control information such as task goals and sequencing indices, and conventional working memory items, such as representations of sensory inputs. Each of these processors will be described in more detail below. In what follows, parameter values described as *standard* are current estimates that are assumed to be constant for the architecture, while those described as *typical* are free to vary depending on properties of the task situation.

2.1 Perceptual Processors

The perceptual processors require roughly the same amount of processing time as assumed in the MHP, but have many differences. The perceptual processors in EPIC are simple "pipelines," in that an input produces outputs at a certain later time, with no "moving window" time integration

effect as assumed by the MHP. A single stimulus input to a perceptual processor can produce multiple outputs to be deposited in WM at different times. The tactile perceptual processor handles movement feedback from effector organs; this feedback can be important in coordinating multiple tasks [Meyer and Kieras 1997a; 1997b], but is not used in the models presented in this article.

2.1.1 *Visual Processor.* EPIC's model of the eye includes a retina that determines what kind of sensory information is available about visual objects in the environment based on the distance (in visual angle) on the retina between the object and center of the fovea. EPIC's current highly simplified model of the retina contains three zones: the fovea (typical radius: 1°), the parafovea (typical radius: 10°), and the periphery (typical radius: 60°). Depending on the exact physical situation, certain information, such as the contents of character strings, is typically available only in the fovea, whereas cruder information, such as whether an area of the screen is filled with characters, is available in the parafovea. Only severely limited information is available in peripheral vision, such as the location of objects, and whether an object has just appeared. The visual perceptual processor maintains a representation of which objects are visible and their properties in the visual working memory of the cognitive processor. Visual working memory is "slaved" to the visual situation; it is kept up-to-date as objects appear, disappear, or change color, and so forth, or as eye movements or object movements change what visual properties are available from the retina. In response to visual events, the visual processor can produce multiple outputs with different timings. When an object appears, the first output is a representation that a perceptual event has been detected (standard delay: 50 msec.), followed later by a representation of sensory properties (e.g., shape, standard delay: 100 msec.), and still later by the results of pattern recognition, which might be task-specific (e.g., a particular shape represents a left-pointing arrow, typical delay: 250 msec.).

2.1.2 *Auditory Processor.* The auditory perceptual processor accepts (1) auditory input and (2) outputs to working-memory representations of auditory events and sequences of auditory events (e.g., speech) that disappear after a time. For example, a short tone signal first produces an item corresponding to the onset of the tone (standard delay: 50 msec.), then at a later time, an item corresponding to a discriminated frequency of the tone (typical delay: 250 msec.), then an offset item (standard delay: 50 msec.). After some time, all the items will disappear from auditory working memory. For simplicity at this time, rather than a graded decay or probabilistic loss function, items simply disappear after a fixed time (typical delay: 4 sec.).

Speech input is represented as items for single words in auditory working memory. The auditory perceptual processor requires a certain time to recognize input words (typical delay: 150 msec. after the acoustic data are present) and produces representations of them in auditory working memory. These items then disappear after a time, the same as other auditory

input. To represent the sequential order of the speech input, the items contain arbitrary symbolic tags for the previous and the next item that link the items in sequence. Thus, a speech input word carries a certain next-tag value, and the next word in the sequence is the item that contains the same tag value as its previous tag. Using these tags, a set of production rules can step through the auditory working-memory items for a series of spoken words, processing them one at a time. For example, the models described in this article process a spoken telephone billing number by retrieving the recognized code for each digit in the tag-chained sequence and using it to specify a key press action.

2.2 Cognitive Processor

2.2.1 *Production Rules and Cycle Time.* The cognitive processor is programmed in terms of production rules, and so an EPIC model for a task must include a set of production rules that specify what actions in what situations must be performed to do the task. Example production rules for the models described in this article will be presented below. EPIC uses the Parsimonious Production System (PPS) interpreter, which is especially suited to task modeling work, as in the CCT models [Bovair et al. 1990]. PPS rules have the format (⟨rule-name⟩ IF ⟨condition⟩ THEN ⟨actions⟩); the rule condition can test only the contents of the production system working memory. The rule actions can add or remove items from the working memory or can send a command to a motor processor. Examples will be given later in this article.

The cognitive processor operates cyclically; at the beginning of each cycle, the contents of working memory are updated with the output from perceptual processors and the previous cycle's modifications; at the end of each cycle, commands are sent to the motor processors. The duration of a cycle is a standard 50 msec., but EPIC can run in a mode in which the cycle duration is stochastic, with a standard mean value of 50 msec. and all other time parameters scaled to this stochastic value. Unlike many other production system architectures, on each cognitive processor cycle, PPS will fire all rules whose conditions match and will execute all of their actions. Thus EPIC models can include true parallel cognitive processing; the EPIC cognitive processor is not constrained to be doing only one thing at a time. Rather, multiple processing threads can be represented simply as sets of rules that happen to run simultaneously.

The multiprocessing ability of the cognitive processor, together with the parallel operation of all the perceptual-motor processors, means that EPIC models for multiple task performance do not conform to the traditional assumption of limited central-processing capacity. Rather, EPIC emphasizes the role of executive process strategies in coordinating perceptual-motor peripherals and other limited structural resources in order to perform multiple tasks. As shown by the detailed quantitative modeling of a variety of multiple-task data reported in Meyer and Kieras [1997a; 1997b] and Kieras and Meyer [1997], EPIC and its approach to modeling multiple-task perfor-

mance has excellent empirical support. However, for the single-task domain in this article, only limited use is made of this parallel-processing capability.

2.2.2 *Working Memory.* The production system working memory is in effect partitioned into several working memories.[2] Visual, auditory, and tactile working memory contain the current information produced by the corresponding perceptual processors. The timing and duration characteristics of these forms of working memory were described above. Motor working memory contains information about the current state of the motor processors, such as whether a hand movement is in progress. This information is updated on every cycle.

Two other forms of working memory deserve special note; these are *amodal* in that they contain information not directly derived from sensory or motor mechanisms. One amodal working memory is the *control store*, which contains items that represent the current goals and the current steps within the procedures for accomplishing the goals, as in the CCT models. An important feature of PPS is that this control information is simply another type of working memory item, and so it can be manipulated by rule actions; this feature is critical for modeling multiple-task performance, in that production rules for an executive process can control subprocesses simply by manipulating the control store (see Meyer and Kieras [1997] for more).

The second amodal working memory, simply termed WM at this time, can be used to store miscellaneous task information, like the working memory NOTEs in the CCT models. At this time EPIC does not include assumptions about the decay, capacity, and representational properties of this general working memory. Clearly, conventional working-memory capacity is limited, but the research strategy in developing EPIC has been to see what constraints on the nature of WM are required to model task performance in detail, rather than following the customary strategy in cognitive modeling of assuming these constraints in advance. Such capacity and loss assumptions for these memory systems do not seem to be required to account for performance in tasks modeled in EPIC thus far; rather, other limitations determined by the perceptual and motor systems appear to dominate performance. These substantial but underappreciated limitations would have been obscured by gratuitous assumptions about central capacity or working-memory limitations (see Meyer and Kieras [1997a; 1997b] for more discussion). For similar reasons, at this time EPIC assumes that information is not lost from the control store, and there is no limit on the capacity of the control store.

[2]EPIC's working-memory structure is not "hard-wired" into PPS. PPS actually has only a single "working memory" which could more clearly be termed the "database" for the production rules. PPS can be used as a multiple-memory system simply by following the convention that the first term in database items indicates the "type" of memory item, as in the examples that follow.

2.3 Motor Processors

The EPIC motor processors are much more elaborate than those in the MHP, producing a variety of simulated movements of different effector organs and taking varying amounts of time to do so. As shown in Figure 1, there are separate processors for the hands, eyes, and vocal organs, and all can be in operation simultaneously. The cognitive processor sends a command to a motor processor that consists of a symbolic name for the type of desired movement and any relevant parameters, and the motor processor then produces a simulated movement with the proper time characteristics. The different processors have similar structures, but different timing properties and capabilities, based on the current human performance literature in motor control (see Rosenbaum [1991]). The manual motor processor has many movement forms, or *styles*, and the two hands are bottlenecked through a single manual processor, and thus normally can be operated either one at a time or synchronized with each other. The oculomotor processor can generate eye movements either upon cognitive command or in response to certain visual events. The vocal motor processor produces a sequence of simulated speech sounds given a symbol for the desired utterance.

2.3.1 *Movement Preparation and Execution.* The different motor processors represent movements and movement generation in the same basic way. Current research on movement control [Rosenbaum 1991] suggests that movements are specified in terms of movement *features* and that the time to produce a movement depends on its feature structure as well as its mechanical properties.

The overall time to complete a movement can be divided into a *preparation phase* and an *execution phase*. The preparation phase begins when the motor processor receives the command from the cognitive processor. The motor processor recodes the name of the commanded movement into a set of movement features, whose values depend on the style and characteristics of the movement, and then generates the features, taking a standard 50 msec. for each one. The time to generate the features depends on how many features can be reused from the previous movements (repeated movements can be initiated sooner) and how many features have been generated in advance. Once the features are prepared, the execution phase begins with an additional delay of a standard 50 msec. to initiate the movement, followed by the actual physical movement. The time to execute the movement depends on its mechanical properties, both in terms of which effector organ is involved (e.g., the eye versus the hand) and the type of movement to be made (e.g., a single finger flexion to press a button under the finger versus a pointing motion with a mouse).

The movement features remain in the motor processor's memory, so that future movements that share the same features can be performed more rapidly. However, there are limits on whether features can be reused; for example, if a new movement is different in style from the previous movement, all of the features must be generated anew. Also, if the task

permits the movement to be anticipated, the cognitive processor can command the motor processor to prepare the movement in advance by generating all of the required features and saving them in motor memory. Then when it is time to make the movement, only the initiation time is required to commence the mechanical execution of the movement.

Finally, a motor processor can prepare the features for only one movement at a time and will reject any commands received during the preparation phase, but the preparation for a new movement can be done in parallel with the physical execution of a previously commanded movement. Once prepared, the movement features are saved in motor memory until the previous execution is complete, and the new movement is then initiated. The cognitive processor production rules can take advantage of this capability by commanding a motor processor with a new movement as soon as it is ready to begin preparing the features for the new movement. The result can be a series of very rapid movements whose total time is little more than the sum of their initiation and mechanical execution times.

2.3.2 *Manual Motor Processor.* EPIC's manual motor processor represents several movement styles, including punching individual keys or buttons already known to be below the finger, pecking keys that may require some horizontal motion, posing the entire hand at a specified location, pointing at an object, and plying a control (e.g., a joystick) to position a cursor onto an object. Each style of movement has a particular feature structure and an execution time function that specifies how long the mechanical movement takes to actuate the device in the task environment.

An example movement style that has particular interest for this article is the *peck* movement style, which is used to strike a key using a single finger that moves about from one key to another (as in "hunt-and-peck" typing). The cognitive processor commands the manual motor processor to perform a peck movement with, for example, the right index finger to a specified object in the physical environment (the key). This movement style involves five features: the peck style, the hand, the finger, the direction of the motion, and the extent of the motion, which is the distance between the current location of the designated finger and the location of the target object. If a previous movement was also a peck movement with the same hand and finger, only the direction and extent might have to be generated. If the movement is also similar in direction and extent to the previous movement, then all of the features could be reused; none would have to be generated. Once the features are generated, the movement is initiated. The time required to physically execute the movement to the target is given by Welford's form of Fitts' Law (see Card et al. [1983, Ch. 2]), with a standard minimum execution time of 100 msec., reflecting that for small movements to large targets there is a physiologically plausible lower bound for the actual duration of a muscular movement. After the simulated finger hits the key, it is left in the location above the key to await the next movement.

2.3.3 *Vocal Motor Processor.* EPIC's vocal motor processor is not elaborated very much at this time; it is based on the minimal facilities needed to model certain dual-task situations (see Meyer and Kieras [1997a; 1997b]). A more complete version of the vocal motor processor would be able to produce extended utterances of variable content, taking into account that the sequential nature of speech means that movements could be prepared on-the-fly during the ongoing speech. However, the simple current version sufficed for the present work because in the telephone operator tasks considered in this article, the operator produced only three possible utterances (i.e., "*New England Telephone*. *May I help you*?," "*New England Public Telephone*. *May I help you*?," and "*Thank you*."), and these are heavily practiced and routine. The current version of EPIC assumes that such utterances can be designated with a single symbol, and it requires only the preparation of two features before execution begins. The actual production of the sound is assumed to be delayed by about 100 msec. after initiation and continues for a time estimated from the data. Further development of the vocal motor processor is planned in the future.

2.3.4 *Oculomotor Processor.* EPIC's eye movements are produced in two modes: a voluntary and an involuntary (reflexive) mode. The cognitive processor commands voluntary eye movements, which are saccades to a designated object. A saccade requires generation of up to two features: a direction and extent of the movement from the current eye position to the target object. Execution of the saccade currently is estimated to require a standard 4 msec./degree of visual angle which includes a residual component duration for settling on the movement endpoint. Involuntary eye movements were not involved in this work, but are either saccades or small smooth adjustments made autonomously by the oculomotor processor in response to changes in the visual situation (hence the arrow between the visual perceptual processor and the oculomotor processor in Figure 1). Visual changes that can trigger involuntary eye movements are a sudden onset (appearance) of an object or the slow movement of a fixated object (cf. Hallett [1986]). In the tasks reported in Kieras and Meyer [1997], EPIC can follow moving objects with a mixture of voluntary and involuntary eye movements.

2.3.5 *Comparison with the MHP Motor Processor.* While much more complicated than the MHP, EPIC's motor processors can be reconciled with MHP's by considering that the 70-msec. cycle time proposed for the MHP motor processor is comparable to the total time required by EPIC's manual motor processor in a simple reaction time task. In such a task, there is only a single response to be made to a stimulus, meaning that the style, hand, and finger used for the response is fixed, and so the feature programming for the response movement can be done in advance. When it is then time to make the movement, only EPIC's initiation time of 50 msec. would be required to start the movement. In such a task, the finger is normally positioned on top of a sensitive key, so only a finger twitch is required, a movement style whose mechanical movement time is very small, only on

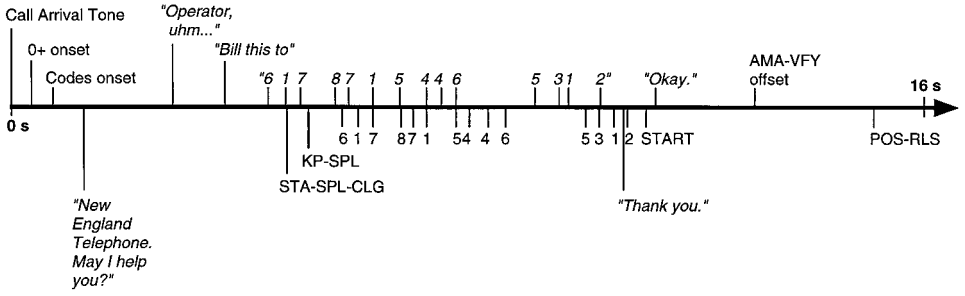


Fig. 2. Timeline of a typical task instance from the data. The horizontal time scale covers the range from zero to 16 seconds. Events above the line are customer's speech (italics) and events displayed by the computer workstation. Events below the line are the operator's speech (italics) and keystrokes. Note how the keystrokes overlap with the customer's speech and lag it by a fairly constant amount.

the order of 20 msec. EPIC would thus produce the same motor processor time (50-msec. initiation + 20-msec. execution) as assumed in the MHP; however, EPIC's motor processors produce a wide variety of movement latencies and execution times, and a much richer set of movements, than originally proposed in the MHP.

3. THE TELEPHONE OPERATOR TASK

3.1 Task Description

Briefly, the task analyzed in this work is a subset of a task domain in which a human *operator* sits at a computer-based workstation and assists a *customer* to complete telephone calls. The general work domain is called *Call Completion Services* (CCS), and the many possible CCS situations are termed *call types*. The workstation consists of a conventional "dumb terminal" display and a standard keyboard with additional key labels for special functions and with a numeric keypad at one end. In the type of task analyzed in this article, the customer dials "0" followed by the destination telephone number, and then supplies orally a billing number to the operator which needs to be entered into the computer and verified before the call can go through. This call type is abbreviated as *Bill-to* calls in this article. The timeline in Figure 2 illustrates a typical instance of this call type, showing the actual events and their timing from the data. The total time to complete the task is directly relevant to the cost of providing operator assistance, and so minimizing total task time is a key goal of the workstation design.

As shown in Figure 2, the task begins when the workstation beeps (the Call Arrival Tone) and then displays several alphanumeric codes on the screen about the call characteristics, the first of which is the code 0+, and another is a code that if present means that the customer is at a pay phone. The operator must greet the customer with one of two standard greetings, depending on whether the customer is calling from a private phone (as in

Figure 2) or a pay phone. The customer supplies the billing information for the call by saying something like "Operator, bill this to 6-1-7 . . . ." At some point after getting the billing information from the customer, the operator says "Thank you." From the screen information and the customer speech, the operator determines which keys to press to specify the billing class of the call. In this class of task, the operator first strikes the Station Special Calling key on the main keyboard (hereafter abbreviated STA-SPL-CLG) and then the Keypad Special key on the main keyboard (KP-SPL), followed by the billing number digits on the numeric keypad, and finally presses the START key on the main keyboard. The computer system then checks the number for validity. If the number is valid, the computer flashes certain codes on the screen (only one of which, AMA-VFY, is shown in Figure 2, to be discussed later). The operator then presses the Position Release (POS-RLS) key on the main keyboard to allow the call to proceed and then signal readiness to handle the next call. As is obvious from Figure 2, the task structure permits (but does not require) some of the activities to go on in parallel, overlapped in time. For example, the operator may press keys while the customer is still speaking and can overlap his or her own speech with keystrokes.

3.2 Rationale for Using the Bill-to Call Type

The reasons for selecting the Bill-to call type for this work needs some explanation in the context of the earlier Project Ernestine [Gray et al. 1993]. Project Ernestine addressed practical questions of how a new workstation compared with the current workstation, and so it considered a large variety of possible CCS call types. All of these call types involve interacting via speech with the customer, but many types also involved speech interaction with more than one customer, as in collect calls. While the practical goals of Project Ernestine required analyzing the broad spectrum of CCS call types, the work reported here had a much narrower focus of verifying whether EPIC models could predict the time required for the operator-computer interaction, and so only call types especially suited to that goal were analyzed.

3.2.1 *Internally and Externally Determined Events.* When testing models against empirical task data, it is critical to distinguish between task events whose content and timing are *internally determined* by the perceptual, cognitive, and motor mechanisms of the human, versus other task events whose content and timing are *externally determined* by the behavior of the task environment, which includes the computer system and the customer. Like other models for human performance, EPIC attempts to represent the processes underlying the internally determined events, not those externally specified by the task environment. The other human (i.e. the customer) who is part of the task environment is not interacting directly with the computer system, but rather plays a role similar to the computer system by acting as a source of operator inputs and a target of operator actions. However, since predicting the content and timing of the

customers' speech is not currently practical, the only reasonable way to include such speech interactions is to represent them as externally determined events.

3.2.2 *Strong Tests of Models Require Internally Determined Events.* The problem with including externally determined event times in the to-be-predicted total task time is that they will "automatically" help account for the empirical total task time; in an extreme case where the operator's task consisted solely of listening to the customer speaking, the externally determined events would perfectly account for the total task time. Thus, including the timing of externally determined events in the total predicted task time tends to inflate the goodness of fit of a model, resulting in an artificially good showing. So if testing a model is the primary goal, it is important to minimize the influence of externally determined events in the to-be-predicted times.

In addition, note that in the full CCS domain, the dominant activity in terms of time will be the operator interacting with the other people, rather than the operator interacting with the computer system, meaning that this externally determined human-human time by far is the strongest determinant of the time to complete the task, swamping the operator-computer contribution. Because the time required for the human-human activity is relatively unrelated to the effects of the user interface design on the time required to interact with the computer, including the human-human activity contributes little to testing a model of how the interface design affects the operator.

Thus the full Project Ernestine CCS call types would not be suitable for testing EPIC because in many of them the externally determined events dominate the situation and dilute the test of the models' ability to predict the internally determined events. Rather, to provide a more rigorous test of the EPIC architecture and its ability to predict operator-computer interaction, we chose as a task domain a subset of call types in which there was a minimum of interaction between the operator and a single customer and a maximum of interaction between the operator and the computer system. In this subset of call types, the Bill-to type described above, the interaction between the operator and customer is limited almost completely to the customer supplying data to be entered into the computer system: the operator greets the customer; the customer says the billing information; and the operator thanks the customer. The interaction between the operator and the computer is relatively elaborate: the operator examines the screen to determine the form of the greeting, keys in the type of billing, keys in the billing number, waits for the system to verify the billing, and then hits a final key to complete the call. Thus, the ability of EPIC models to predict the details of human performance can be given a strong test.

4. MODELING THE TELEPHONE OPERATOR TASK

To simulate the operator's performance of the selected Bill-to type of telephone operator tasks, the task environment module of EPIC was

programmed to generate simulated displays and customer input for this class of calls, and EPIC's cognitive processor was "programmed" with production rules capable of performing all possible instances of the task. Under direction of the cognitive processor, the perceptual and motor processors move the eyes around, perceive stimuli on the operator's workstation screen, and reach for and strike keys. Before presenting the specific models for the task, it is important to present some general issues in predictive modeling and how these are implemented with EPIC.

4.1 General Modeling Properties of EPIC

4.1.1 *Fixed and Free Parameters.* The presentation of any modeling approach should document what aspects or parameters of the modeling framework are fixed and are thus supposed to generalize across applications, and what parameters have to be estimated from data specific to the situation being modeled. In EPIC, the fixed aspects and parameters are: (1) the connections and mechanisms of the EPIC processors; (2) most time parameters in the processors (the ones described as *standard* above); and (3) the feature structure of the motor processors. Thus adopting the EPIC framework entails committing to all these details of the modeling framework. The model properties and parameters that are then free to vary from model to model or task to task are: (1) the task-specific production rule programming, which is constrained to some extent because it must be written to execute the task correctly and reasonably efficiently, and as described below, can be based on a simple GOMS model for the task; (2) the task-specific perceptual encoding types and times (i.e. the *typical* times above) involved in the task, which must be defined in terms of the production rules, and are constrained to be similar and constant over similar perceptual events; (3) the style of movements used to control the device (e.g., touch-typing versus visually-guided pecking), if it is not constrained by the task.

4.1.2 *What Goes In and What Comes Out?* Similarly, any modeling approach should document what information the model builder has to supply in order to construct the model and what information the constructed model will then produce in return for the supplied information. To construct an EPIC model, the model builder has to supply the information corresponding to the free parameters described above, namely (1) a production rule representation of the task procedures, (2) task-specific perceptual processor encodings and timings, and (3) any movement styles not determined by the task requirements. In addition, the model builder must supply (4) the simulated task environment, which includes the physical locations and characteristics of relevant objects external to the human, and (5) a set of task instances whose execution time is of interest; these instances must specify only environmental events and their timing and are used to control only the environment module of the simulation.

In return for these inputs, an EPIC model will generate the predicted sequences of simulated human actions required by each task instance, and

the predicted time of occurrence of each action. If the production rules were written to describe general procedural knowledge of how to perform the task, these predictions can be generated for any task instance subsumed by these general procedures.

4.2 Choosing Task Strategies for Models: The Need for Modeling Policies

A key insight from the modeling work done thus far with EPIC and other cognitive architectures is that the architecture itself does not necessarily determine the strategy that should be used to represent the task. There are many alternative strategies for accomplishing a task that differ in how the human abilities and limitations represented in the architecture will be manifested in performance. For example, in modeling multiple-task situations using EPIC (see Meyer and Kieras [1997a; 1997b]), there are multiple possibilities for how the activities in different tasks that involve different stimulus and response modalities can be overlapped, both with each other and with cognitive activities.

One way to identify the specific strategy governing overlapping in a task is to propose a strategy, generate predicted performance under that strategy, compare the predicted performance to empirical data, and repeat until the predicted data matches the empirical results. In typical scientific cognitive modeling work devoted to verifying a cognitive architecture and understanding how a task might be done, it is acceptable to arrive at task strategies in this post hoc model-fitting mode.

However, using EPIC, or any other cognitive architecture, for predictive engineering purposes requires the ability to develop reasonably accurate task strategies in an a priori mode. That is, the whole rationale for engineering models is to predict performance independently of empirical observation of the task in question. Indeed, once scientific work on cognitive architectures has progressed beyond simple demonstrations of feasibility, success at a priori prediction success is also required to fully establish a body of theory on scientific grounds. Predicting performance on an a priori basis requires not only a usefully accurate cognitive architecture, but also a set of *modeling policies* for how to choose and represent task strategies that are usefully accurate on an a priori basis. In particular, the modeling policies specify which of the possible approaches permitted in EPIC should be used to deal with each specific modeling issue for how the strategy utilizes the EPIC architecture. The ideal modeling policy would be directly related to the results of an a priori task analysis, easy to represent as an EPIC model, and empirically accurate.

4.3 Some Possible Policies for Overlapping Task Activities

Table I lists the modeling issues that arise from applying EPIC to tasks such as the telephone operator task, along with some possible policies for each issue that are relevant to the models in this article. The next section presents a series of models that represent interesting combinations of these policies.

Table I. Modeling Issues and Alternative Modeling Policies Used in Constructing the A Priori Models

Modeling Issues	Alternative Modeling Policies
Procedure Representation	Production rules in GOMS format
Method Structure	Hierarchical methods Single flat method
Coordination within a Motor Modality	Complete all movement phases before executing the next method step Prepare the next movement while current movement is executing
Coordination between Motor Modalities	Move eye to target before keystrokes Do not overlap movements in different modalities Overlap eye, hand, and vocal movements
Movement Anticipation	Prepare features for next movement as far in advance as possible Preposition the eyes and hands to next locations as far in advance as possible

4.3.1 *Procedure Representation.* The first modeling issue in Table I concerns procedure representation, the basic approach by which the procedures involved in the task will be represented in terms of production rules. The policy used throughout this article is that the production rules follow a GOMS model format (see Card et al. [1983] and John and Kieras [1996b]). A GOMS model consists of a set of methods for accomplishing task goals. A method is a series of steps containing elementary operators that when executed will accomplish the goal. Selection rules specify which method to apply, depending on the specific situation. Earlier work in the Cognitive Complexity Theory approach (CCT) by Kieras and Polson [1985] and Bovair et al. [1990] developed a format for representing GOMS methods with production rules in a simplified cognitive architecture. The present work is an extension and refinement of that approach.

4.3.2 *Method Structure.* Given that the GOMS procedural knowledge is represented in production rules, exactly how should the methods be represented? The first policy listed in Table I is that the goal and method structure is hierarchical with multiple subgoals and subprocedures, reflecting the hierarchical task decomposition typically used in a GOMS analysis. The second policy is that there is a single flat, or nonhierarchical, method. Note that just because the analytic task decomposition is hierarchical does not mean that the internal representation of the human procedural knowledge is also hierarchical; for example, extreme practice might well produce a more efficient flattened method representation. The EPIC work on multiple-task performance assumes partially hierarchical methods, but

typical experiments do not involve the extreme amounts of practice that telephone operators have.

4.3.3 *Coordination within a Motor Modality.* This modeling issue concerns how a single motor processor is used by the cognitive processor. A simple policy is to wait for a motor processor to complete both preparation and execution phases for a movement before proceeding to the next step, producing very deliberate step-by-step activity. The second policy is to prepare the next movement while the current one is executing, enabling the next movement to start execution sooner.

4.3.4 *Coordination between Motor Modalities.* This issue concerns how the cognitive processor coordinates the different motor systems. There are two different subissues and corresponding policies. The first involves eye-hand coordination. Based on research on human aimed movements (e.g., Abrams et al. [1990]), the proposed policy is that when making keystrokes, the eye must be moved in advance to the vicinity of the target keys, and that when the visual system has acquired them (e.g., the shape of the keys is available), the location of the target key is then known; the manual motor processor can then be instructed to make the keystroke movement. The second subissue concerns whether the task strategy makes use of the ability of the motor processors to operate overlapped, in parallel. While overlapping can result in much faster task execution, as shown in the multiple-task modeling (see Meyer and Kieras [1997a; 1997b], such strategies to coordinate the use of the different motor processors can be quite complex. Two possible policies are the simple strategy of not using any motor parallelism, while the other policy is the opposite, using the parallel capability of the motor processors to fully overlap movements.

4.3.5 *Movement Anticipation.* Often a task permits movements to be anticipated, whereupon they can be made sooner than otherwise; the issue is whether the task strategy takes advantage of this possibility. Three possible policies are shown. The first is to make no use of any sort of movement anticipation. Second, the cognitive processor could instruct the motor processor to prepare the features for a later movement in advance, thereby saving time when the movement is to be actually made. Third, the eyes or the hands could be actually *premoved* into position before the movement would ordinarily be made, thereby again saving time by reducing the physical distance that must be traveled when it is time to actually make the movement. The premovement would then be followed by an advance preparation of the anticipated movement. Note that advance movement and preparation might not be of value; the extent of time savings depends on subtle details of how long prepositioning movements take and whether the overhead of managing advance preparation exceeds the resulting motor time savings.

5. A SET OF A PRIORI MODELS

A variety of policies for representing tasks are meaningful and reasonable. Our approach to understanding which policies would be most useful for tasks such as the telephone operator task was to propose several models that implemented different policies and then determine which could account for performance. It is important to keep in mind that the purpose of this work is to attempt to predict performance on an a priori basis, so the approach was to generate the alternative models based on the task analysis, the EPIC architecture, and the modeling policies; none of the models were subsequently altered to provide a better fit to the data.

Choice of Models. We constructed a series of models, listed in Table II, to represent points on a policy continuum starting with a nonoptimized purely hierarchical and sequential description of task performance, through models that took advantage of the parallel-processing possibilities of the cognitive architecture, to models that represented highly optimized utilizations of the architecture. Thus, the sequence of models represents sets of policies that describe a hypothetical increase in processing efficiency and sophistication, which presumably would be related to the degree of practice in the task.

As shown in Table II, only a small subset of the possible combinations of policies were developed in the set of a priori models; the number of possible models is quite large, and so only a subset is feasible to develop and test. The subset chosen was based on which combinations of features seemed most likely to occur and which models would be most useful to test. For example, the Hierarchical Fully-Sequential model is closely related to the earlier Bovair et al. [1990] CCT production rule models, and so it provides a conceptual baseline for the more complex models. The other Hierarchical models differ in one policy each to help understand the effects of each policy. The Flattened methods models should reflect the effects of extreme practice on the task procedures; accordingly, only two of them were included: the Flattened Motor-Parallel model provides again a conceptual baseline in that it is the simplest model with Flattened methods; in contrast, the Flattened Premove Prepared Motor-Parallel model represents the likely additional effects of extreme practice. That is, if operators were so practiced as to have developed flattened methods, then it seems likely that they would also be fully anticipating movements with both premovements and preparation. Since the operators in this task domain are highly experienced, we expected that one of the more optimized models would provide the best account of their performance, but as it happens, one of the simpler models and policies appears instead to provide the best fit.

Each of the models and how they implement the corresponding policies will be described with example production rules in the following sections.

Simplifications Based on the Operator's Expertise. In this task domain, and in the data for this work, the operators are very well practiced, having years of experience in the task. Such experience would determine certain

Table II. A Priori Models Developed for the Telephone Operator Task and the Policies Represented in Them

A Priori Models	Policies
Hierarchical Fully-Sequential Model	*Production rules in GOMS format* *Hierarchical methods* *Complete all movement phases before executing the next step* *Do not overlap movements in different modalities*
Hierarchical Motor-Parallel Model	Production rules in GOMS format Hierarchical methods *Prepare next movement while current movement is executing* *Overlap eye, hand, and vocal movements*
Hierarchical Prepared Motor-Parallel Model	Production rules in GOMS format Hierarchical methods Prepare next movement while current movement is executing Overlap eye, hand, and vocal movements *Prepare features for next movement as far in advance as possible*
Hierarchical Premove Prepared Motor-Parallel Model	Production rules in GOMS format Hierarchical methods Prepare next movement while current movement is executing Overlap eye, hand, and vocal movements Prepare feature for next movement as far in advance as possible *Preposition the eyes and hands to next location as far in advance as possible*
Flattened Motor-Parallel Model	Production rules in GOMS format *Single flat method* Prepare next movement while current movement is executing Overlap eye, hand, and vocal movements
Flattened Premove Prepared Motor-Parallel Model	Production rules in GOMS format *Single flat method* Prepare next movement while current movement is executing Overlap eye, hand, and vocal movements Prepare features for next movement as far in advance as possible Preposition the eyes and hands to next locations as far in advance as possible

Policies that distinguish a model from those previously listed are shown in italics.

features of the task strategy followed by operators. Thus, the models made certain general strategy assumptions, some of which are similar to the extreme expertise assumption underlying the CPM-GOMS modeling approach (see Gray et al. [1993] and John and Kieras [1996b]).

Eye movements can be made directly to the fields on the computer display; visual search for the task-relevant information is not required. Note that because of the fairly large distance observed on the videotape between the operators' eyes and the display, certain fields on the display can be seen in parafoveal vision well enough for the task. So although separate eye movements are made to these fields, the information may be available in visual working memory sooner.

As a simplifying assumption, the words of the customer request consist of either utterances that can be ignored, or an utterance that means "bill this call to the following number." Once this bill-to utterance is heard, the model proceeds to strike the STA-SPL-CLG key (Station Special Calling, meaning "bill to another number") and the KP-SPL key (Keypad Special, meaning a keypad entry follows). If digits are heard before such an utterance, it is assumed that the same call type is intended.

In accordance with the policy on eye-hand coordination, before the STA-SPL-CLG and the KP-SPL keys can be pressed, the eye must be moved to them, and their shape must be available in visual working memory. Likewise, before the digit keys can be pressed, the eye must be moved to the center key of the keypad (the FIVE key), and its shape must be available in visual working memory; however, eye movements to individual digit keys are not required. But, because of the high frequency of the striking the POS-RLS key (Position Release, meaning that the call processing is complete), no eye movement to it or visual acquisition of it is required.

The operator says "thank you" after the customer has finished speaking, regardless of whether other actions are still under way.

The operator knows that the call has been verified when a certain designated screen event occurs.

5.1 Hierarchical Method Models

5.1.1 *The Hierarchical Fully-Sequential Model.* This first model and its production rule representation will be described in some detail, because it represents the simplest combinations of modeling policies; all of the other models were more optimized versions of this one, and they can be explained very briefly after this detailed introduction. In accordance with the policy on representing the task procedures in terms of a GOMS model, the model was developed by starting with a GOMS model for the task which was then translated into production rules. As shown in Table II, this model assumed hierarchical methods, but was strictly sequential in that the policies followed for both within-modality and between-modality motor coordination required all movement phases to be complete before the next movement was commanded, with either the same or a different motor processor. This

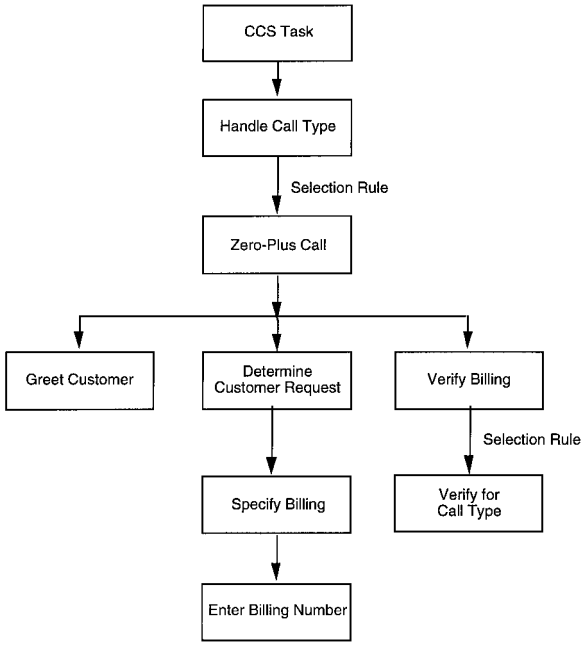


Fig. 3. The hierarchy of goals and methods in the GOMS model for the telephone operator task. Connections labeled as selection rules indicate possible additional subgoals.

represents a simple "baseline" model that takes advantage only of the ability of the perceptual processors to operate in parallel with the rest of the system.

We started with a GOMS model of the NGOMSL type (see John and Kieras [1996a; 1996b]). This type of GOMS model describes the task procedures as a hierarchical set of methods consisting of sequential executed actions, both perceptual and motor. Figure 3 shows the hierarchy of Goals and Methods, and Figure 4 contains an excerpt of the GOMS methods as they were expressed in the natural-language-like NGOMSL notation (see Kieras [1988; 1998]). Constructing this GOMS model was a routine activity needing no special explanation here (see John and Kieras [1996a; 1996b] for discussion) and was performed as part of the preliminary modeling work described by Wood, Kieras, and Meyer.[3] Then a set of production rules were written to implement this GOMS model in a style similar to the CCT templates described in Bovair et al. [1990].

Figure 5 shows example production rules for some of the NGOMSL methods shown in Figure 4. The production rules in EPIC conform to the Parsimonious Production System format described in Bovair et al. [1990]; each is in the format (⟨name⟩ IF ⟨condition⟩ THEN ⟨actions⟩). The condition

[3]S. Wood, D. Kieras, and D. Meyer, "An EPIC Model for a High-Performance Task." A poster presented at the CHI '94 ACM Conference on Human Factors in Computing (Boston, Mass., April 25–28).

```
Method for goal: Do CCS Task
    Step 1. Wait for call arrival tone
    Step 2. Look at the CLG-Type field
    Step 3. Retain CLG-Type and Accomplish goal: Handle CLG Type
    Step 4. Press POS-RLS
    Step 5. Return with goal accomplished
Selection rule set for goal: Handle CLG Type
    If CLG-Type is 0+ then Accomplish goal: Handle zero-plus-call
    If CLG-Type is 0 then Accomplish goal: Handle zero-call
    If CLG-Type is 1+ then Accomplish goal: Handle one-plus-call
    If CLG-Type is OVT then Accomplish goal: Handle overtime-call
    Return with goal accomplished
Method for goal: Handle zero-plus-call
    Step 1. Retain payment is paid and call is station-to-station
    Step 2. Accomplish goal: Greet customer
    Step 2. Accomplish goal: Get customer request
    Step 5. Accomplish goal: Do call type verification
    Step 6. Return with goal accomplished
Method for goal: Greet customer
    Step 1. Look at coin-pre field
    Step 2. Decide: If "COIN-PRE" is present then say public greeting
            else say private greeting
    Step 3. Return with goal accomplished
Method for goal: Get customer request
    Step 1. Wait for customer speech to start
    Step 2. Get next phrase
    Step 3. Decide: If no next phrase then Return with goal accomplished
    Step 4. Decide: If phrase is "bill to"
            then Accomplish goal: Get billing number.
    Step 5. Goto step 2.
Method for goal: Get billing number
    Step 1. Retain billing is special
    Step 2. Press KP-SPL
    Step 3. Accomplish goal: Enter billing number
    Step 4. Say "Thank you"
    Step 5. Return with goal accomplished
Method for goal: Enter billing number
    Step 1. Get the next recognized digit
    Step 2. Decide: if no more digits then Goto step 5
    Step 3. Press key for digit
    Step 4. Goto step 1
    Step 5. Press START
    Step 6. Return with goal accomplished
```

Fig. 4. Excerpt of NGOMSL methods for a subset of telephone operator tasks.

is a list of *clauses* (e.g., (GOAL DO CCS TASK)), all of which must match items in working memory for the rule to fire. Each item in working memory consists simply of a list of symbols. A clause in a rule condition may contain a wildcard term (three consecutive question marks) or a variable (a term denoted by a symbol with a question mark as a prefix) which can be bound to the corresponding symbol in a working-memory item. By convention, each rule has control store conditions of a GOAL clause and a STEP clause that determine the flow of control, typically followed by additional clauses that test the contents of other partitions of working memory.

```
(*MFG-Do-CCS
    IF
    ((GOAL DO CCS TASK) (NOT (WM Executing CCS Task))
     (MOTOR OCULAR MODALITY FREE))
    THEN
    ((SEND-TO-MOTOR OCULAR FIXATE FIXATION-POINT)
     (ADDDB (STEP Waiting for call)) (ADDDB (WM Executing CCS Task))))
(*Do-CCS*Wait-for-CAT
    IF
    ((GOAL DO CCS TASK)(STEP Waiting for call)
     (AUDITORY DETECTION EVENT ONSET))
    THEN
    ((DELDB (STEP Waiting for call)) (ADDDB (STEP Look at CLG type))))
(*Do-CCS*Look-at-CLG-Type
    IF
    ((GOAL DO CCS TASK) (STEP Look at CLG type)
     (MOTOR OCULAR MODALITY FREE))
    THEN
    ((SEND-TO-MOTOR OCULAR FIXATE CLG-TYPE)
     (DELDB (STEP Look at CLG type)) (ADDDB (STEP Handle Call))))
(*Do-CCS*Handle-Call
    IF
    ((GOAL DO CCS TASK) (STEP Handle Call)
     (VISUAL ?OBJECT SHAPE CLG-TYPE) (VISUAL ?OBJECT LABEL ?TEXT)
     (MOTOR OCULAR MODALITY FREE))
    THEN
    ((ADDDB (WM CLG type is ?TEXT))
     (ADDDB (GOAL Handle CLG type))
     (DELDB (STEP Handle Call)) (ADDDB (STEP Release Position))))
(*Do-CCS*Release-Pos
    IF
    ((GOAL DO CCS TASK) (STEP Release Position)
     (WM CLG-type Done)
     (MOTOR MANUAL MODALITY FREE))
    THEN
    ((SEND-TO-MOTOR MANUAL PERFORM Peck POS-RLS-KEY)
     (DELDB (WM CLG-type Done))
     (DELDB (STEP Release Position)) (ADDDB (STEP Do-CCS Finish))))
(*Do-CCS*RGA
    IF
    ((GOAL DO CCS TASK) (STEP Do-CCS Finish)
     (MOTOR MANUAL MODALITY FREE))
    THEN
    ((DELDB (WM Executing CCS Task))
     (ADDDB (WM CCS Task Done))
     (DELDB (GOAL DO CCS TASK)) (DELDB (STEP Do-CCS Finish))))
```

Fig. 5. Example production rules from the Hierarchical Fully-Sequential model that implement the top-level method for the telephone operator task.

The actions in the rules usually include removing the current step clause from the working-memory database (with DELDB) and adding the step clause corresponding to the next step in the procedure (with ADDDB). This *chain* structure results in the rules firing one at a time in sequence as in the CCT models, implementing the step-by-step execution of NGOMSL

methods. Note that EPIC and PPS can fire multiple rules on each cycle, but the Hierarchical Fully-Sequential model makes no use of this capability.

The example in Figure 5 shows the production rules for the top-level method in Figure 4, *Method for goal: Do CCS Task*. The first rule, named *MFG-Do-CCS, is an initial housekeeping rule that also initializes the eye location. The rule named *Do-CCS*Wait-for-CAT implements Step 1 of the method by waiting for the onset event of the Call Arrival Tone, (AUDITORY DETECTION EVENT ONSET), to arrive in auditory working memory. The next rule, *Do-CCS*Look-at-CLG-Type, does Step 2 by instructing the ocular motor processor to move the eye to the location where the type of the call will appear on the screen. The clause (MOTOR OCULAR MODALITY FREE) in the rule means that the rule will not fire unless the motor partition of working memory contains the information that the ocular motor system (modality) is idle; an eye movement is neither being executed nor being prepared. The rule *Do-CCS*Handle-Call performs Step 3 by waiting until the visual perceptual processor has deposited the contents of the calling-type field in working memory as the value of the variable ?TEXT, and then it sets up a subgoal to be accomplished by adding the item (GOAL Handle CLG type) to working memory. The appearance of this item will trigger the startup rule for the corresponding submethod.

The next rule, *Do-CCS*Release-Pos, waits for two things: the previously called submethod to signal completion by depositing the item (WM CLG-type Done) in working memory and the appearance of (MOTOR MANUAL MODALITY FREE) in working memory, which means that the manual modality has finished any keystrokes that were under way in the submethod. When these conditions are met, the rule fires, instructing the manual motor processor to "peck" the POS-RLS key. The final rule, *Do-CCS*RGA, corresponds to the final Step 5 in the method; when the manual modality becomes idle upon completion of the POS-RLS keystroke from the previous rule, control returns to the calling method.

The main features of this example can be summarized: (1) each GOMS method entails a pair of "housekeeping" productions corresponding to the NGOMSL method statement (name prefixed by convention with MFG) and the return statement (postfixed with RGA); (2) there is a separate production rule for each basic perceptual or motor operator step in the method; (3) the production rule for a step always waits for any motor action to be completed before it fires to instruct the next motor action, or to invoke a submethod; (4) likewise, if an action such as an eye movement was made to acquire perceptual information, the rule for the next step always waits until the perceptual information becomes available; (5) when a submethod is invoked, execution of the next step waits for a signal that the submethod is completed.

This set of features implements the policy for representing the GOMS methods in terms of EPIC production rules that results in a model that corresponds closely to the original CCT models for text editing. In particular, the elapsed time between each production rule firing contains not just the cognitive processor cycle time, but all perceptual time associated with

```
(*Enter-number*Get-next-digit
    IF
    ((GOAL Enter number) (STEP Get next digit)
     (WM Next speech is ?prev)
     (AUDITORY SPEECH PREVIOUS ?prev NEXT ?next TYPE DIGIT CONTENT ?digit)
     (VISUAL ??? SHAPE FIVE-KEY)
     (MOTOR OCULAR MODALITY FREE)
     (MOTOR MANUAL MODALITY FREE))
    THEN
    ((SEND-TO-MOTOR MANUAL PERFORM Peck ?digit)
     (DELDB (WM Next speech is ?prev)) (ADDDB (WM Next speech is ?next))))
```

Fig. 6. Production rule that enters each digit in the Hierarchical Fully-Sequential model.

the rule conditions and all times resulting from motor processor activities initiated by the previous step. Thus, each production rule is constrained to act as a single step in the NGOMSL analysis; each step in each method is fully executed before the next step is executed; and the execution of a submethod suspends execution of the calling method (see John and Kieras [1996b] for a related discussion). This model had a total of 50 production rules; one rule for each step in each method plus the additional "housekeeping" rules for each method.

Although it had strictly sequential methods, the Hierarchical Fully-Sequential model overlaps some aspects of the task; for example, typing the billing number can begin while the customer is still speaking digits. Figure 6 shows the production rule *Enter-number*Get-next-digit that does this work. Each recognized spoken digit is represented as a sequentially tagged item in auditory working memory, of the form (AUDITORY SPEECH PREVIOUS ?prev NEXT ?next TYPE DIGIT CONTENT ?digit), where the variable ?digit represents a recoding supplied by the auditory perceptual processor that designates the physical target of the corresponding key. The rule *Enter-number*Get-next-digit uses a "pipeline" approach similar in spirit to John's [1996] model of transcription typing: as each digit arrives in working memory, the cognitive processor waits until the manual motor processor has finished the previous keystroke and then sends the keystroke command corresponding to the digit to the manual motor processor and updates a "pointer" in WM to the next speech item in auditory working memory to be processed. In conformance with the eye-hand motor coordination policy, the rule requires that before the digit can be typed, the shape of the center key on the keypad, the FIVE key, must be in visual working memory to ensure that the target key is in view.

5.1.2 *The Hierarchical Motor-Parallel Model.* This model took advantage of the parallel operation capabilities of the motor processors and removed the heavy sequential constraints of the first model. As shown in Table II, the Motor-Parallel model is like the Fully-Sequential model except that it does not wait for all motor activity to be completed before starting a step, and it takes advantage of the motor processor's ability to prepare the next movement while a prior movement is currently under way.

The policy represented by the Hierarchical Motor-Parallel model was implemented by starting with the production rules from the Hierarchical Fully-Sequential model, and then each condition clause of the form (MOTOR ⟨type⟩ MODALITY FREE) was either deleted or changed to the form (MOTOR ⟨type⟩ PROCESSOR FREE), which indicates that the motor processor is free to accept instructions for a new movement preparation. The clause was deleted if the rule did not instruct the corresponding motor processor; the clause was changed to the processor-free form if the rule instructed that motor processor. The two panels of Figure 7 illustrate the modification. In the Fully-Sequential model, each rule did not fire until *all* previously initiated motor activity was fully completed, as in rule *Get-Billing-Number*Press-STA-SPL-CLG. The effect of the change to the Motor-Parallel form is that each rule waits only the minimum time for motor activity to finish. Each rule that instructs a motor processor merely waits for that same motor processor to be ready to prepare a new movement, even if a movement is still being executed.

Thus, in the Motor-Parallel model, activities involving different processors are performed in parallel; preparations for the next movement are made in parallel with the execution of a movement; and rules wait on motor processors only the minimum required by the architecture. As a result, many purely cognitive activities, such as the rules performing method housekeeping, execute while perceptual-motor actions are taking place. The Hierarchical Motor-Parallel model still follows strict hierarchical GOMS methods, but takes full advantage of the ability of the perceptual and motor mechanisms to run in parallel both internally and with other processors, including the cognitive processor. The time occupied by each step in the methods is now a complex function that depends on the exact timing relationships between perceptual and motor processes.

5.1.3 *The Hierarchical Prepared Motor-Parallel Model.* This next model took further advantage of the motor processors. As shown in Table II, this model assumes that the operator would anticipate eye or hand movements by instructing the motor processors to prepare the movements in advance, as soon as it was ready to accept movement preparation instructions and as early as logically possible. This advance preparation results in substantial time savings (typically 100–250 msec.) when the movement is actually made. Note that EPIC's motor processors do not impose a time penalty for a movement preparation that is subsequently not used or is overwritten by a different movement instruction. Thus, it is possible to speed up performance if the likely next keystroke can be predicted; there is no slowing down of movement preparation if the prediction is incorrect.

This model was constructed by adding additional production rules to the Motor-Parallel model to send the preparation instructions to the motor processors at the right time. Rather than inserting the preparation rules into the existing methods, the implementation of the policy took advantage of PPS's parallel rule-firing ability by simply attaching these rules to the

Fully-Sequential Model

```
(*Get-Billing-Number*Look-at-STA-SPL-KEY
    IF
    ((GOAL Get billing number) (STEP Look at STA-SPL-CLG)
     (MOTOR OCULAR MODALITY FREE))
    THEN
    ((SEND-TO-MOTOR OCULAR FIXATE STA-SPL-CLG-KEY)
     (DELDB (STEP Look at STA-SPL-CLG)) (ADDDB (STEP Press STA-SPL-CLG key))))
(*Get-Billing-Number*Press-STA-SPL-CLG
    IF
    ((GOAL Get billing number) (STEP Press STA-SPL-CLG key)
     (VISUAL ??? SHAPE STA-SPL-CLG-KEY)
     (MOTOR OCULAR MODALITY FREE) (MOTOR MANUAL MODALITY FREE))
    THEN
    ((SEND-TO-MOTOR MANUAL PERFORM Peck STA-SPL-CLG-KEY)
     (DELDB (STEP Press STA-SPL-CLG key)) (ADDDB (STEP Press KP-SPL key))))
(*Get-Billing-Number*Press-KP-SPL-key
    IF
    ((GOAL Get billing number) (STEP Press KP-SPL key)
     (MOTOR MANUAL MODALITY FREE))
    THEN
    ((SEND-TO-MOTOR MANUAL PERFORM Peck KP-SPL-KEY)
     (DELDB (STEP Press KP-SPL key)) (ADDDB (STEP Get number entered))))
```

Motor-Parallel Model

```
(*Get-Billing-Number*Look-at-STA-SPL-KEY
    IF
    ((GOAL Get billing number) (STEP Look at STA-SPL-CLG)
     (MOTOR OCULAR PROCESSOR FREE))
    THEN
    ((SEND-TO-MOTOR OCULAR FIXATE STA-SPL-CLG-KEY)
     (DELDB (STEP Look at STA-SPL-CLG)) (ADDDB (STEP Press STA-SPL-CLG key))))
(*Get-Billing-Number*Press-STA-SPL-CLG
    IF
    ((GOAL Get billing number) (STEP Press STA-SPL-CLG key)
     (VISUAL ??? SHAPE STA-SPL-CLG-KEY)
     (MOTOR MANUAL PROCESSOR FREE))
    THEN
    ((SEND-TO-MOTOR MANUAL PERFORM Peck STA-SPL-CLG-KEY)
     (DELDB (STEP Press STA-SPL-CLG key)) (ADDDB (STEP Press KP-SPL key))))
(*Get-Billing-Number*Press-KP-SPL-key
    IF
    ((GOAL Get billing number) (STEP Press KP-SPL key)
     (MOTOR MANUAL PROCESSOR FREE))
    THEN
    ((SEND-TO-MOTOR MANUAL PERFORM Peck KP-SPL-KEY)
     (DELDB (STEP Press KP-SPL key)) (ADDDB (STEP Get number entered))))
```

Fig. 7. Example modifications made to transform the Hierarchical Fully-Sequential rules (top panel) into the Hierarchical Motor-Parallel form. The only change is in the condition clauses that test the motor states. The Fully-Sequential model tests for all active modalities to be free (idle) before performing the actions for the next step; the Motor-Parallel model tests only that the relevant motor processor is free to accept a command for a new movement.

```
(*Handle-Zero-Plus-Call*PrepareSTA-SPL-CLG
    IF
    ((Goal Do Zero-Plus Task)
     (STEP Greet customer)
     (NOT (STEP PREPARE STA-SPL-CLG-KEY))
     (NOT (WM PREPARED STA-SPL-CLG-KEY)))
    THEN
    ((ADDDB (STEP PREPARE STA-SPL-CLG-KEY))))
(*Prepare*STA-SPL-CLG
    IF
    ((STEP PREPARE STA-SPL-CLG-KEY)
     (MOTOR MANUAL PROCESSOR FREE))
    THEN
    ((SEND-TO-MOTOR MANUAL PREPARE Peck STA-SPL-CLG-KEY)
     (ADDDB (WM PREPARED STA-SPL-CLG-KEY))
     (DELDB (STEP PREPARE STA-SPL-CLG-KEY))))
```

Fig. 8. Example rules for preparing a movement in the Hierarchical Prepared Motor-Parallel model.

existing methods as separate execution threads. Figure 8 shows an example of rules for preparing a movement in advance. During the step of greeting the customer in one of the submethods, the rule *Handle-Zero-Plus-Call*PrepareSTA-SPL-CLG fires, enabling rule *Prepare*STA-SPL-CLG which waits until the manual motor processor is free to prepare the STA-SPL-CLG keystroke. Thus once the method has advanced to the customer-greeting step, the preparation rules fire independently of what the other rules are doing. Making these rules be potentially executable in parallel with the preexisting task rules requires no modification of the preexisting rules. This policy thus uses the parallel capabilities of the cognitive processor as well as those of the motor processors.

Such preparation was possible only for movements that could be assumed to be constant at that point in the task; for example, typing a digit of the billing number could not be prepared in advance, since the billing number would always vary from one task instance to the next. In contrast, pressing the billing category key could be prepared far in advance, given that the task structure makes it reasonable to assume that this key is probably the next one to be hit. Thus, the policy was to identify the earliest possible preparation point for each predictable keystroke and attach the preparation rules to the corresponding procedure step.

5.1.4 *The Hierarchical Premove Prepared Motor-Parallel Model.* Table II shows that this model went even further in the direction of anticipating movements by incorporating premovements of the eyes or hands. For example, certain keystrokes could be anticipated by moving the hand to the location of the key in advance (a *pose* style movement, comparable to a *home* movement in CPM-GOMS) and then preparing the actual keystroke movement. Thus, both the physical movement and the motor preparation were done as much in advance as possible, further speeding task execution. The implementation for this policy was like that of the Prepared model; an example is shown in Figure 9. During the greet-customer step, the hand is

```
(*Handle-Zero-Plus-Call*SetupSTA-SPL-CLG
    IF
    ((Goal Do Zero-Plus Task) (STEP Greet customer)
     (NOT (WM SETUP STA-SPL-CLG-KEY IN PROGRESS))
     (NOT (WM PREPARED STA-SPL-CLG-KEY)))
    THEN
    ((ADDDB (WM SETUP STA-SPL-CLG-KEY IN PROGRESS))
     (ADDDB (STEP PREMOVE STA-SPL-CLG-KEY))))
(*Premove*STA-SPL-CLG
    IF
    ((STEP PREMOVE STA-SPL-CLG-KEY)
     (MOTOR MANUAL PROCESSOR FREE))
    THEN
    ((SEND-TO-MOTOR MANUAL PERFORM POSE STA-SPL-CLG-KEY)
     (ADDDB (STEP PREPARE STA-SPL-CLG-KEY))
     (DELDB (STEP PREMOVE STA-SPL-CLG-KEY))))
(*Prepare*STA-SPL-CLG
    IF
    ((STEP PREPARE STA-SPL-CLG-KEY)
     (MOTOR MANUAL PROCESSOR FREE))
    THEN
    ((SEND-TO-MOTOR MANUAL PREPARE PECK STA-SPL-CLG-KEY)
     (ADDDB (WM PREPARED STA-SPL-CLG-KEY))
     (DELDB (WM SETUP STA-SPL-CLG-KEY IN PROGRESS))
     (DELDB (STEP PREPARE STA-SPL-CLG-KEY))))
```

Fig. 9. Example rules for making a prepositioning movement to a key followed by advance preparation for a subsequent keystroke in the Hierarchical Premove Prepared Motor-Parallel model.

posed at the key and then prepared to peck it. Thus, this policy attempts to optimize execution speed as much as possible within the confines of the hierarchical method structure.

5.2 Flattened-Method Models

The original Hierarchical Motor-Parallel model was then modified in a different direction, *flattening* the methods, along the lines suggested by widely accepted principles of learning of cognitive skill, such as those proposed in learning theories such as ACT [Anderson 1976; 1987] and SOAR [Laird et al. 1986]. Extreme practice of a skill should cause the method housekeeping and other such rules to be replaced by a more efficient set of rules that effectively turn "subroutine" methods into "in-line" methods. For example, a rule that invoked the submethod for entering a billing number would be replaced by a rule that simply performed the first substantive step for entering the billing number and then chained to the next step. The resulting rule set could be represented as a tree, in which each class and subclass of the task would be performed by a sequence of rule firings along a single linear path through the tree, and each rule performs some substantive task action or decision, with no housekeeping rules. However, as in the Hierarchical Motor-Parallel models, the perceptual-motor activities can overlap substantially. The flattened-method models are perhaps closest to the CPM-GOMS models for the telephone

operator tasks [Gray et al. 1993], in that the methods consist simply of sequences of operators, with no hierarchical submethod structure and consequently no cognitive execution overhead (see John and Kieras [1996b] for more discussion of this distinction). As mentioned above, only two models of this type were developed here.

5.2.2 *The Flattened Motor-Parallel Model.* The rule set for this model was constructed under a policy of modifying the Hierarchical Motor-Parallel model to eliminate the housekeeping rules by concatenating the substantive steps of the separate methods, with selection rules being replaced by simple conditional tests on each branch. Thus, the hierarchical method structure was flattened into a single method with branches. This model provides a conceptual baseline for assessing the effect of method flattening in that, as Table II shows, the flattened methods are the only modeling policy different from the Hierarchical Motor-Parallel model.

Figure 10 shows a portion of the single resulting method which can be compared to the rules for the top-level method in the original hierarchical model (Figure 5). In the original model, the second rule is chained to a rule that invoked a submethod via a selection rule for the Bill-to call type. However, here there is a single flat method, in which the second rule, *Do-CCS*0+Look-at-Coin-Pre, fires only if the call type is 0+ (Bill-to) and then chains directly to the rule for the next step of looking at the coin prefix field, which then chains directly to one of two rules that instructs the vocal motor processor with the appropriate greeting. Thus, all of the intermediate layers of method overhead have been removed; the task is executed with a minimum number of rules firing; and all of these rules test perceptual input or instruct motor processors.

5.2.3 *The Premove Prepared Flattened Motor-Parallel Model.* Finally, as Table II shows, this model incorporated the same advance movement and preparation as the Premove Prepared Hierarchical Motor-Parallel model. The movement anticipation followed the same approach shown in Figure 9 of attaching independent sets of rules to the appropriate steps in the main method. As discussed above, it seems that if operators had practiced the procedures to the point of fully flattening the methods, they would also have fully anticipated the movements with both premovements and advance feature preparation; hence there was no model using only one of these anticipation policies. Because the minimum number of activities are on the critical path, this model produces the fastest execution times.

6. COMPARISON OF THE MODELS TO EMPIRICAL DATA

6.1 Observed and Predicted Times

The basic question concerns how well the a priori constructed models predict actual total task performance time. While both the model and the data contain keystroke-by-keystroke times, practical engineering models will be generally more concerned with completion times for whole tasks,

```
(*MFG-Do-CCS
    IF
    ((GOAL DO CCS TASK)
     (NOT (WM PREPARED FOR CCS TASK))
     (NOT (WM Executing CCS Task))
     (MOTOR OCULAR PROCESSOR FREE))
    THEN
    ((SEND-TO-MOTOR OCULAR FIXATE FIXATION-POINT)
     (ADDDB (STEP Waiting for call))
     (ADDDB (WM Executing CCS Task))))
(*Do-CCS*Wait-for-CAT
    IF
    ((GOAL DO CCS TASK) (STEP Waiting for call)
     (AUDITORY DETECTION EVENT ONSET)
     (MOTOR OCULAR PROCESSOR FREE))
    THEN
    ((SEND-TO-MOTOR OCULAR FIXATE CLG-TYPE)
     (DELDB (STEP Waiting for call)) (ADDDB (STEP Look at coin-pre))))
(*Do-CCS*0+Look-at-Coin-Pre
    IF
    ((GOAL DO CCS TASK)
     (STEP Look at coin-pre)
     (VISUAL ?OBJECT SHAPE CLG-TYPE)
     (VISUAL ?OBJECT LABEL 0+)
     (MOTOR OCULAR PROCESSOR FREE))
    THEN
    ((SEND-TO-MOTOR OCULAR FIXATE COIN-PRE-FIELD)
     (ADDDB (WM CLG type is 0+))
     (ADDDB (WM Call is for station))
     (ADDDB (WM Payment is paid))
     (DELDB (STEP Look at coin-pre))
     (ADDDB (STEP Decide coin-pre))))
(*Do-CCS*0+Decide-greeting-pub
    IF
    ((GOAL DO CCS TASK)
     (STEP Decide coin-pre)
     (VISUAL ??? SHAPE Coin-Pre-FIELD)
     (VISUAL ?OBJECT APPEARANCE VISIBLE)
     (MOTOR VOCAL PROCESSOR FREE))
    THEN
    ((SEND-TO-MOTOR VOCAL SAY WORD-GREETING-PUBLIC)
     (DELDB (STEP Decide coin-pre))
     (ADDDB (STEP Wait for customer request))
     (ADDDB (WM Coin Pre is Coin-Pre))))
```

Fig. 10. Example production rules from the Flattened Motor-Parallel model; compare these with Figure 5 containing the top-level method for the Hierarchical Motor-Sequential model.

and so this presentation is limited to total task time prediction. Some discussion of individual keystroke-level events appears in Meyer and Kieras [1997b] and Kieras and Meyer [1997].

We selected Bill-to call type task instances from the videotaped task performances of experienced operators collected, but not analyzed, during the Gray et al. [1993] Project Ernestine. In the selected task instances, the operators followed the same basic procedure covered by the models and

made no substantial overt errors, and the customers provided the relevant task information smoothly, without discussion or interaction with the operator. Unfortunately, despite the many hours of task performance available in the recordings, the number of task instances meeting these requirements was severely limited, resulting in a set of four task instances for each of two operators. While a larger sample size would be obviously desirable, this data set was also remarkable in that it contained detailed recordings for users who were extremely highly practiced at a real-world task. Such data is not easily obtained, so the decision was made to model this small sample of realistic data; much larger data sets from laboratory tasks have already been modeled extensively using EPIC (see Meyer and Kieras [1997a; 1997b] and Kieras and Meyer [1997]).

6.1.1 *Details of the Videotaped Data.* The Project Ernestine videotapes contain staged calls to actual operators who used actual operator equipment, but who were aware that they were in an experiment. The staged calls followed a script that was prepared not with detailed model-fitting or traditional experimental design in mind, but rather to collect a sample of behavior that was intended to represent actual customer behavior and telephone operator activity. Thus, compared to a standard laboratory study, the task instances were selected in a way that could be called "realistically haphazard," rather than fully balanced and replicated. In many cases the customer behavior contains pauses, corrections, or suboptimal behaviors, such as forcing the operator to prompt the customer for the required information. Together with the lack of replications of call types, as well as errors or hesitations on the part of the operator, the result is that often there is no "clean" version of a particular call type in which the normative procedure is followed by both customer and operator.

The eight different telephone operators each handled from as few as 9 to as many as 57 staged calls, with about 20 calls being typical, for a total of 187 recorded task instances. A total of 56 task instances were of the Bill-to type used in this modeling. Of these, many involved errors, hesitations, or additional verbal interactions, leaving 25 that were deemed "clean" task instances. Of these, 9 involved an alternative task procedure,[4] and 5 were from a single operator who used a touch-typing style on the keypad, as opposed to the pecking style used by the other operators. Setting these cases aside left 11 candidate task instances, which were almost all contributed by only 2 operators. The final selection was to use the same number of cases from these 2 operators to produce the most stable data set available

[4]The workstation will accept the STA-SPL-CLG "bill to another number" key *after* the billing number is entered, as well as before the KP-SPL key that starts the billing number entry. Apparently, neither the operator training nor the workstation documentation dictate a single procedure. Note that such procedural indeterminacy may affect which actions are on the critical path for completing the task and thus determine the overall task completion time. Therefore, training can be as important as workstation design for such tasks. For simplicity, this work used only the procedure described earlier, which seems to be the trained procedure and appears most often in these data.

for the model testing. The result was 4 task instances from each of these two operators.

The video and audio recordings of the selected task instances were digitized at full frame rate, and the times of the roughly 50 individual events (display changes, words of speech, and keystrokes) in each task instance were determined to the nearest video frame (1/30 sec., 33 msec.). The externally determined events (e.g., response time of the workstation, content and timing of each word of the customer's speech measured from the operator's greeting) were used to program the environment simulation module. The internally determined observed events (i.e., the operator's speech and keystrokes) were used to test the accuracy of the models.

6.1.2 *Details of the Simulation Process.* Each of the eight task instances was simulated with the EPIC models by programming the environment simulation module with the externally determined events and then running the EPIC system with the production rules for each model. The EPIC models generated the sequence of observed operator actions and the predicted time at which each action occurred.

For each task instance, a script for the task environment module was prepared. It contained the information from the videotaped task instance that represented the behavior of the workstation and customer and included measured values for the operator's speech duration. Specifically, the environment module was programmed with the following items for each task instance:

(1) The time delay measured from the call arrival tone for the appearance of each item initially displayed on the screen, e.g., COIN-PRE.
(2) The duration of the operator's vocal greeting and "Thank you" speech, since these are not currently predicted by EPIC's vocal processor.
(3) The time delay of each phrase or digit spoken by the customer, measured from the end of the operator's greeting, and a definition of the recognized content (e.g., "uh operator bill this to" is recognized as the symbol BILL-TO; "five" is recognized as FIVE key).
(4) The time delay for the appearance of each of the final items displayed on the screen, e.g., AMA-VFY, measured from the START keystroke.

Note that the environment module programming does not include when the vocal greeting starts or when the operator makes keystrokes; these events are under the control of the simulated operator represented by the EPIC model.

The predictions for the models were obtained by running each model on each task instance. Note that running the simulation is the easiest way to generate the predicted total task times because the exact timing of the events in a specific task instance can depend in subtle ways on exactly what happens to be on the critical path, which depends in turn on the exact timing of the preceding input and output events.

For each simulation of a task instance, the task environment module starts the simulated task by producing the call arrival tone and putting the

initial items on the screen. When the simulated operator says the appropriate greeting, the task environment module then produces each unit of the customer's speech at the observed delays from the greeting. In response, the simulated operator presses the appropriate keys based on the content of the customer's speech. For example, in response to "bill this to five, seven . . ." the simulated operator hits the keys STA-SPL-CLG, KP-SPL, 5, 7, . . . START, with from 4 to 14 digits being involved. Then the task environment module puts up the final screen items with the observed delays after the START key is hit. The simulated operator then responds to these items with the final keystrokes to complete the task.

During the simulation runs, all EPIC parameters were kept fixed at the values mentioned above in the architecture description. These values were the fixed architectural parameter values set during the development of EPIC and modeling other tasks (given above as the *standard* and *typical* values) and a single task-specific value that was determined during the preliminary model-fitting work in this task domain,[5] namely the time to recode a spoken digit to the name of a key (also given above as a *typical* value). The execution time predictions produced by the different models differed only as a result of how the production rules controlled the EPIC architecture; parameters such as perceptual encoding times were not changed from one model or task instance to the next. For each policy model, all the sample task instances were performed by the same set of production rules; the rules were not altered to fit specific task instances.

6.1.3 *A Baseline Model.* To provide a basis for judging the relative contribution of the EPIC models, the total task execution time was predicted for each task instance using the Keystroke-Level Model (KLM) which usually produces usefully accurate results in ordinary computer interface applications [Card et al. 1983; John and Kieras 1996a; 1996b]. The KLM-predicted task execution time was simply the total of the observed relevant workstation response times, the customer and operator speaking times, and the total time for keystrokes (280 msec. each) and homing operators for movements of the right hand to and from the keypad (400 msec. each).

7. RESULTS

7.1 Prediction of Total Task Execution Times

7.1.1 *Definition of Total Task Execution Time.* The telephone company defines the total task execution time as the time during which the operator and the workstation are occupied by a call, which is the duration between the initial call arrival signal tone and the last keystroke, the POS-RLS (Position Release) key that indicates that the call processing is completed.

[5]S. Wood, D. Kieras, and D. Meyer, "An EPIC Model for a High-Performance Task." A poster presented at the CHI '94 ACM Conference on Human Factors in Computing (Boston, Mass., April 25–28).

However, this definition of total task time is a poor choice for testing the models. First, as mentioned both by our informants and Gray et al. [1993, p. 264], the POS-RLS keystroke is not constrained very much by the task structure; the key can be struck at a variety of valid times, no one of which seems to be tightly specified in the training materials or common practice in the task domain. Second, the total time terminated by POS-RLS includes a large externally determined time during which the computer searches the billing number database. Accordingly, we used a more stable and more internally determined definition of the total task execution time: the time between the call arrival tone and the time to press the penultimate key, the START key, which is struck immediately after the last digit of the billing number is entered and which initiates the search through the database to verify the billing number. Some implications of the reaction time for the POS-RLS key will be discussed below.

7.1.2 *Goodness-of-Fit Measures.* A popular way to assess the goodness-of-fit of model predictions to data is to calculate regression equations and coefficients. However, such a calculation is not very informative in this case for two reasons. First, all of the models, even the Keystroke-Level Model, accounted for 83% or more of the variance (r^2) in the task execution times, which was statistically significant ($p < 0.05$) in all cases. This result is not as surprising as it might seem. For this call type, the major determinant of the task execution time is the length of the billing number supplied by the customer, and all the models (along with common sense) predict that the execution time will be longer as the length of the billing number increases.

Second, the goal of engineering models is to supply predicted values of usability metrics that are not merely correlated with the empirically measured values, but are actually similar in numerical value. If the predictions are close to the actual empirical values, the regression equation will have an intercept close to zero and a slope close to one, but this is not reflected by the correlation coefficient, making the traditional r^2 metric a poor choice for a figure of merit.

Thus, a measure of the actual difference between the predicted values and the observed values is more informative than correlation measures. The mean error is a poor choice because the goal of engineering models is to obtain accurate predictions for each task instance considered; underpredictions of some situations must not be allowed to compensate for overpredictions in others. Thus, the best choice for a simple summary statistic for the accuracy of the models is the *average absolute error of prediction*; the difference between predicted and observed task times was calculated for each task instance, and then the absolute values of these differences were averaged and expressed as a percentage of the average observed value. Engineers often use a rule of thumb which says that predictions accurate within 10–20% are useful for design purposes.

Finally, another issue of goodness of fit concerns whether the model generally over- or underpredicts the data. These models, like most other engineering models, are idealizations. They apply a task strategy consis-

Table III. Quality of Fit between Models and Observed Total Task Execution Times

Model Type	Total Task Time (msec.)	Error
Observed	9292	
Hierarchical Methods		
Fully-Sequential	9887	8.8%
Motor-Parallel	8672	6.6%
Prepared Motor-Parallel	8447	9.5%
Premove/Prepared Motor-Parallel	8290	11.7%
Flattened Methods		
Motor-Parallel	8415	9.6%
Premove/Prepared Motor-Parallel	8065	14.3%
Keystroke-Level Model	12094	28.4%

tently and correctly and do not suffer from lapses of attention, slips of the finger, and so forth. Thus, such models are generally more efficient than a human who is attempting to follow the same strategy, so the model should underpredict, i.e., predict execution times that are less than the observed values (cf. Gray et al. [1993]). Therefore, if a model overpredicts by predicting times longer than the observed values, then either the parameters in the model are incorrect, or the model can be rejected as following an incorrect strategy. Since most of the parameters in the EPIC architecture are fixed, a model that seriously overpredicts the data can be ruled out as misrepresenting how the task is done.

7.1.3 *Model Accuracy.* Table III lists the average total task execution time for the observed data and each model, along with the average absolute error of prediction for each model. The average absolute error of prediction ranges from 6.6% for the Hierarchical Motor-Parallel model to 14.3% for the worst-fitting EPIC model to 28.4% for the Keystroke-Level Model. The question is which models predicted better than others.

Using the engineer's 10–20% rule of thumb, all of the EPIC models appear to be usefully accurate in predicting total task execution time, while the Keystroke-Level model appears to be inadequate. The EPIC models are all reasonably accurate because they all represent to some extent how the task activities can be overlapped with each other (e.g., the billing number can be keyed in while the customer is still speaking the digits). This is the critical determinant of execution time in this task. In contrast, the Keystroke-Level model does not overlap any activities; so it not only produced the largest average absolute error, but it overpredicted every one of the task instances, with an error over 30% in five cases. In fact, the Keystroke-Level model was within 10% only on two rather brief task instances that involved entering only four digits, which of course present far fewer opportunities for overlapping task activities. Clearly the Keystroke-Level model is quite inappropriate for predicting execution times in the tasks studied here.

Comparing the EPIC models to each other is technically difficult due to the small sample size and apparently nonnormal distributions of the

prediction errors; conventional significance tests cannot be trusted to be accurate under these conditions. Instead, the Wilcoxon Signed-Ranks test was used to calculate the statistical significance for two-tailed planned comparisons of the absolute error of prediction on each task for the different models. This procedure tests the null hypothesis that the differences between paired interval-scale values are symmetric about zero and then produces exact probabilities for the test statistic under this hypothesis. Rejecting this null hypothesis means that one of the models produces errors that tend to be less than the other. Unless stated otherwise, all claimed differences are significant at the 0.05 level or better.

The Hierarchical Motor-Parallel model (6.6% error) produced prediction errors that were significantly less than all of the other EPIC models except for the Hierarchical Fully-Sequential model (8.8%). Even though the Hierarchical Fully-Sequential model was fairly accurate, it also suffers from a problem in overprediction like the Keystroke-Level model; it overpredicts the task instances in all but one case. With regard to other models, the least accurate EPIC model, the Flattened Premove Prepared model (14.3% error), is significantly less accurate than the Hierarchical Prepared Motor-Parallel model (9.5%) and the Flattened Motor-Parallel model (9.6%). This leaves the Hierarchical Motor-Parallel model as the best-fitting model, being both simple and the most accurate of the models that underpredict the data.

The major surprise in the results is that the Hierarchical Motor-Parallel model, which is only moderately efficient, is more accurate than the more optimized models, even though the human operators are very highly practiced in the task. Apparently, operators take advantage of the parallel preparation and execution capabilities of their motor processors to speed up their performance, but they make little use of prepositioning the eyes and hands in advance, at least in the task instances studied. This conclusion is consistent with informal observation from the videotapes that operators did not seem to engage in prepositioning of the hands very often; eye and head positions generally could not be observed in the videotapes. Performance slower than architecturally possible could stem from the fact that very fast performance is either not needed or is not sustainable in the task environment; this will be discussed more below.

The Flattened method models represent an even more extreme effect of practice, but they are also less accurate than the simple Hierarchical Motor-Parallel model, and in fact the most "expert" model, the Flattened Premove Prepared model, was the least accurate. Based on these results, one would be tempted to conclude that there is no evidence that "flattening" of methods takes place as a result of extreme practice. However, there are two caveats about this conclusion. First, although consistent, the difference between the Hierarchical Motor-Parallel model and the Flattened Motor-Parallel model on these task instances is not large, being only about an average of 200 msec. An examination of how the models execute the task instances shows that in the Hierarchical Motor-Parallel model, the rules for submethod "calls" and "returns" tend to be overlapped with perceptual

motor processing or external events, so the cognitive overhead does not contribute much to the total task execution time. A different type of task in which the overhead of submethods was proportionately larger would provide a stronger test of this important hypothesis about the effects of practice. Second, again as will be discussed more below, this task domain may not in fact require or encourage extremely optimized performance, despite the amount of practice.

7.1.4 *Comparison of EPIC Models to CPM-GOMS.* A natural question is how the EPIC models compare to the Gray et al. [1993] CPM-GOMS models. A fundamental difference is that the EPIC models are based on an a priori task analysis and modeling policies that specified a single task strategy for each model that was then used to generate predictions for each of the 8 task instances. Thus, the EPIC models attempt to account for all of the task instances with a single task strategy, and they do so with an average absolute error of 7–14%. In contrast, the CPM-GOMS models in Gray et al. were individually constructed to fit each of a set of selected benchmark task instances, and they predicted these task times quite well, with an average absolute error of only 3%. However, these benchmark models did not predict a set of field trial data very well, producing an average absolute error of about 25%, because the benchmarks were not accurately representative of the field data tasks. The accuracy of EPIC models in the same situation would likewise depend on whether they were supplied with representative task instances. However, the generative property of EPIC models would make it a simple matter to obtain predictions for a large number of different task instances chosen to produce a representative cross section of the actual task. In contrast, performing CPM-GOMS analysis is not practical for a large number of benchmarks. In summary, the EPIC models succeeded at generating usefully accurate predictions for the selected task instances and could be readily applied to a larger number of task instances, but we do not yet know the predictive accuracy of CPM-GOMS for a comparable situation. See the Appendix for a more detailed discussion.

7.2 Implications of Individual Keystroke Time Predictions

7.2.1 *Speech Recognition Delay and Input Rate.* The main thrust of this work was to use EPIC models to predict total task times, but the models also predicted the timing of individual keystrokes. Most of the keystrokes involved typing the digits of the billing numbers in response to the customer's speaking them. These keystroke reaction times were predicted very poorly by some of the models and only moderately well by the best-fitting Hierarchical Motor-Parallel model, so a detailed presentation of the predicted and observed keystroke reaction times is not justified in this article. However, some important implications can be summarized.

Detailed examination shows, in the observed task instances, that the customer speaks the digits at a rate typically slower than the model can make the corresponding keystrokes, often pausing at apparent "chunk"

boundaries in the billing number. However, exactly how the model responds to these pauses and the speaking rate depends on the exact relationship of the various processing delays involved. A pause in the input may or may not produce a pause in the keystroking output, depending on whether the model has been keeping up with the input or lagging behind. In turn, whether the model keeps up with the input depends heavily on an estimated perceptual processing parameter, the time required to recode a spoken digit or phrase into the identity of the corresponding key. The value estimated from the preliminary model work appears to be too inaccurate to predict closely these complex timing relationships, but because the perceptual recoding process is often not on the critical path, the total task time can still be predicted reasonably accurately. Kieras and Meyer [1997] report some results in which individual keystroke times can be accurately modeled in these data. This better-fitting model is not included here, because it was constructed *post hoc* to fit the data, while the goal of the present work is a priori prediction. Future predictive modeling in the telephone operator domain should of course be informed by such additional results.

In the meantime, these observations suggest that the major bottleneck in the task execution time is the rate at which the customer speaks the digits, not the rate at which the operator can type. A related conclusion appears in Gray et al. [1993], in that the speech interaction occupies most of the critical path through the task. A second implication is that perhaps the reason why operators appear to be following a task strategy that is only moderately efficient is that the task is so limited by the customer's speaking rate that there is little need for the greater efficiency of the more highly optimized strategies.

7.2.2 *What Controls the POS-RLS Keystroke?* As discussed above, POS-RLS (Position Release), the last keystroke in the task, is problematic. Gray et al. [1993, p. 264] and our informants point out that the POS-RLS keystroke can be made at a variety of times, even in advance of the system's billing number verification. That is, no harm is done if the operator strikes the key before the system has completed verification: the system will halt and wait for the operator to resolve the situation if the billing number is invalid or will simply allow the call to go through immediately if the number is valid. Once POS-RLS is struck, the workstation will accept the next arriving call. Thus striking the key early would result on average in the call being completed more quickly and the next call being processed sooner. However, according to the workstation training materials, a certain screen event indicating a successful billing verification is the proper signal for hitting POS-RLS, and this was assumed in the GOMS analysis underlying the models (cf. Gray et al. [1993]).

Given that the system verification process is relatively long, the operator should be idle at the time of the relevant screen event, so the latency of the POS-RLS keystroke should depend on only a simple reaction to the screen event. However, the timing of this keystroke was quite unstable in the

observed data and did not appear to be systematically related to any of the events on the computer screen. In fact, in one case, the operator was not even looking at the workstation while making the keystroke! Furthermore, the observed reaction time for the keystroke was quite large: the POS-RLS keystroke is made an average of 1581 msec. after the very last screen event that could reasonably serve as a stimulus, the AMA-VFY event shown in Figure 2. In contrast, the otherwise well-fitting Hierarchical Motor-Parallel model, which performs no advance prepositioning or preparation, predicts a reaction time to this same event averaging only about 727 msec. The more optimized models predict even shorter times. A longer reaction time could be produced by adding an eye movement and a perceptual delay to locate the key before launching the keystroke, but this artificial move would still leave a large amount of the time discrepancy unaccounted for.

Thus this keystroke takes much longer than it should according to the EPIC architecture. A possible explanation is that the keystroke is not being made with maximum speed in response to a screen event, but rather it is under the control of some other aspect of the task situation. Since POS-RLS indicates that the operator is ready to accept another call, perhaps the operator is delaying this keystroke in order to insert a bit of "breathing room" before the next call arrives. Given that this task is normally done all day by the operator, it is not surprising that the task strategy might involve such workload regulating techniques. EPIC's a priori predictions help reveal the presence of such task factors by providing a "best-case" execution time based on the human cognition and performance architecture. Thus, discrepancies between predicted and observed results signal the possible presence of factors not represented in the model strategy or the architecture. In this way, EPIC models can help identify task properties that are not a matter of human information-processing characteristics.

8. CONCLUSIONS

Some EPIC models for a high-performance task were constructed using a priori task analysis, modeling policies, and parameter values, and these models were able to predict total task execution times with an accuracy high enough to be useful as engineering models for interface design. These results show the potential for EPIC to provide a framework for engineering models in complex, high-performance domains in which the operator's performance time depends on the overlapped activity of separate processing capabilities.

Of particular interest was the result that the most accurate model was rather easy to construct because it followed very simple modeling policies corresponding to fairly unoptimized performance. The accuracy of this relatively unoptimized model contrasts sharply with the observations in Meyer and Kieras [1997a; 1997b] and Kieras and Meyer [1997] that in many laboratory tasks people follow highly optimized strategies which often involve advance movement preparation like those in the more optimized models tested here. Apparently, the telephone operator task does not

involve such highly optimized task strategies, although the operators are very practiced, and execution speed is important. As discussed above, the task speed is limited in ways that may make highly optimized strategies of little value; overlapping helps the manual activity to keep up with the input, but there is little utility in spending energy over the course of a day on movement propositioning and preparation in this task. Taking into account the larger context of a task, such as whether it has to be done for hours at a stretch, is not the norm in cognitive modeling. But the fact that the corresponding strategies are distinguishable in the data argues that such considerations are critical for practical task modeling. Thus, a possible heuristic for a priori task modeling is to select a modeling policy that is no more optimum than the *overall* task situation makes energetically and economically worthwhile. An important goal for engineering model research will be to try to characterize this larger context of task performance.

The effort required to construct EPIC models seems to be considerably less than that for CPM-GOMS. In both approaches, the analyst must make many decisions about the details of task execution, such as when eye movements are necessary, but for EPIC models, these decisions are made only once for the general task procedures, rather than possibly multiple times in each specific benchmark task instance. Constructing the present models was relatively easy. The initial GOMS model was routine once the information on the actual task procedures became available, and building the production rule models was a matter of applying readily standardizable templates. Finally, the EPIC architecture itself was fixed and required no development for this analysis. In return for the rather modest construction effort, the resulting EPIC model can generate predicted execution times for all possible task instances within the scope of the GOMS model. Thus EPIC models would appear to be very efficient engineering models for multimodal high-performance tasks.

At this point, EPIC is definitely a research system, and certainly it is not ready for routine use by interface designers. However, in some situations, such as the Gray et al. Project Ernestine, the economics of the interface evaluation problem can make even a novel and demanding analysis approach a practical and useful solution. Following the precedent of the CCT and NGOMSL models [John and Kieras [1996a; 1996b], as the EPIC architecture stabilizes and as experience is gained in applying it to interface analysis problems, it should be possible to develop a simplified method of analysis that will enable designers to conveniently apply engineering models based on EPIC.

APPENDIX

The exact set of models, data, and evaluations of prediction accuracy appearing in the Gray et al. [1993] CPM-GOMS study is rather complicated and must be summarized here. Their overall execution time analyses used both unweighted predictions and weighted predictions that take the frequency of different call types into account, but for present purposes, only

the unweighted values of prediction error are relevant. For each of 15 selected call types, they first devised a "benchmark" scenario intended to represent that call type and collected videotaped task instances from several operators performing each benchmark task. Then they selected 15 individual performances, one of each type, and constructed a specific CPM-GOMS model for each one of these benchmark task instances as performed on a current workstation, giving 15 models, one for each of the selected task instances. As in the present EPIC models, the times for externally determined events and the operator's speech duration were estimated from the data and included in the model. They reported an average unweighted error of 3% when these models were used to predict the total task time for the same task instances used to construct them. Since the models all underpredicted the task times, this result is equivalent to an average absolute error of 3%.

Gray et al. then modified the models to produce predicted execution times for the same task instances as they would be performed on a new workstation, giving a second set of 15 CPM-GOMS models. Both the first and second set of models were compared to execution time data from a large-scale field trial that included both the current and the new workstations. Gray et al. report an average unweighted error of about 11% and 12% when the two sets of 15 models predicted the observed average execution times for the current and the new workstations, respectively.

However, there are two problems with this initially impressive result. First, using the average error allows overpredictions to compensate for underpredictions; the average *absolute* error can be calculated from the data reported in Gray et al., and this is substantially greater, about 26% and 25% error for the current and the new workstations, respectively. These substantially greater errors are due to the second problem, which Gray et al. term *benchmark selection error*: the benchmark tasks were supposed to be representative of the actual tasks in the field data, but in fact they turned out not to be very representative. This is shown by the fact that the *observed* times on the benchmark tasks predicted the observed field data times for the same call type and workstation only slightly better than the models; the average error was about 8%, and the average absolute error was almost 25%. Since the CPM-GOMS models were constructed to predict the benchmark task data, they could not predict the field data any better than the benchmark task data themselves. Hence, to a great extent, the accuracy of the CPM-GOMS models for predicting the field data was limited by the accuracy of the benchmark tasks used to represent the corresponding field data. Gray et al. point out that future applications of the CPM-GOMS methodology should take steps to ensure that the modeled benchmarks are accurately representative.

It is tempting to compare the EPIC model accuracy with the CPM-GOMS accuracy, but first note that any comparison must be done with the same measure of error, with the average absolute error being the most suitable. The two modeling efforts have two major differences. First, the data being predicted is rather different. Gray et al. used POS-RLS to mark the end of

the task execution; START was used in the present results. Furthermore, the Gray et al. data consisted of either single instances of 15 call types or averaged field trial data for the same 15 call types. The data predicted in the present work consisted of 8 instances of a single call type that subsumes several of the Gray et al. types (e.g., Gray et al. classify pay-phone Bill-to calls as a separate type from ordinary Bill-to phone calls). The Gray et al. data would seem to be more general and were certainly appropriate for the practical goals of their work, but the call type selected for the present work has a maximum of internally determined events and a minimum of externally determined ones. The more externally determined events there are in a task instance, the easier it will be to get a good fit to the total execution time, but to what extent this might have affected the Gray et al. results versus the present results would have to be determined.

Second and most important, the ground rules for prediction in the two projects were rather different. While both approaches used parameters estimated both inside and outside the task domain, the EPIC models were based on an a priori task analysis and modeling policies that specified a single task strategy for each model that was then used to predict each of the eight task instances. In contrast, the CPM-GOMS models in Gray et al. were individually constructed to fit each benchmark task instance. The CPM-GOMS models then predicted their construction benchmarks quite well, but due to the error in selecting benchmarks, did not do very well in predicting the field data. If the problem was indeed due to a poor selection of benchmarks, then the conclusion is that at least for this domain, we do not yet know how accurately CPM-GOMS models can predict times for task instances other than the benchmark instances used to construct the models.

REFERENCES

ABRAMS, R. A., MEYER, D. E., AND KORNBLUM, S. 1990. Eye-hand coordination: Oculomotor control in rapid aimed limb movements. *J. Exper. Psychol. Human Percept. Perf. 16*, 2, 248–267.

ANDERSON, J. R. 1976. *Language, Memory, and Thought*. Lawrence Erlbaum, Hillsdale, N.J.

ANDERSON, J. R. 1987. Skill acquisition: Compilation of weak-method problem solutions. *Psychol. Rev. 94*, 192–210.

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

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

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

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. *Human-Comput. Interact. 8*, 3, 209–237.

HALLETT, P. E. 1986. Eye movements. In *Handbook of Perception and Human Performance*, K. R. Boff, L. Kaufman, and J. P. Thomas, Eds. Vol. 1. Wiley, New York, 10-1–10-112.

JOHN, B. E. 1996. TYPIST: A theory of performance in skilled typing. *Human-Comput. Interact. 11*, 321–355.

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

JOHN, B. E. AND KIERAS, D. E. 1996b. The GOMS family of user interface analysis techniques: Comparison and contrast. *ACM Trans. Comput.-Human Interact. 3*, 4 (Dec.), 320–351.

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

KIERAS, D. E. 1998. A Guide to GOMS model usability evaluation using NGOMSL. In *Handbook of Human-Computer Interaction*, M. Helander, T. Landauer, and P. Prabhu, Eds. 2nd ed. North-Holland, Amsterdam. To be published.

KIERAS, D. E. AND MEYER, D. E. 1997. An overview of the EPIC architecture for cognition and performance with application to human-computer interaction. *Human-Comput. Interact*. To be published.

KIERAS, D. E. AND POLSON, P. G. 1985. An approach to the formal analysis of user complexity. *Int. J. Man-Mach. 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 *Proceedings of CHI* (Denver, Colo., May 7–11). ACM, New York.

LAIRD, J., ROSENBLOOM, P., AND NEWELL, A. 1986. *Universal Subgoaling and Chunking*. Kluwer Academic Publishers, Boston, Mass.

MEYER, D. E. AND KIERAS, D. E. 1997a. A computational theory of executive cognitive processes and multiple-task performance: Part 1. Basic mechanisms. *Psychol. Rev. 104*, 3–65.

MEYER, D. E. AND KIERAS, D. E. 1997b. A computational theory of executive cognitive processes and multiple-task performance: Part 2. Accounts of psychological refractory-period phenomena. *Psychol. Rev*. To be published.

MCMILLAN, G. R., BEEVIS, D., SALAS, E., STRUB, M. H., SUTTON, R., AND VAN BREDA, L. 1989. *Applications of Human Performance Models to System Design*. Plenum Press, New York.

ROSENBAUM, D. A. 1991. *Human Motor Control*. Academic Press, New York.

Received November 1995; revised July 1997; accepted July 1997