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

David E. Kieras [1], *Scott D. Wood*,
Artificial Intelligence Laboratory
Electrical Engineering & Computer Science Department
and
David E. Meyer
Department of Psychology

University of Michigan

ABSTRACT

Engineering models of human performance permit some aspects of usability of interface designs to be predicted from an analysis of the task, and thus 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.

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 predictions based on an analysis of the user's task in conjunction with principles and parameters of human performance (Card, Moran, & Newell, 1983; John & Kieras, 1994). This paper, an expansion of Kieras, Wood, and Meyer (1995), reports results on a new class of engineering models for a multi-modal high-performance HCI task, namely the telephone operator tasks studied by Gray, John, and Atwood (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 also because they are *active system* tasks (John & Kieras, 1994) 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 (1994), engineering models for active system tasks are currently under-developed. Finally, predicting performance in such tasks can be economically important; a detailed information-processing

[1] Address for correspondence: David Kieras, Artificial Intelligence Laboratory, Electrical Engineering & Computer Science Department, University of Michigan, 1101 Beal Avenue, Ann Arbor, Michigan 48109-2110, kieras@eecs.umich.edu

analysis of telephone operator tasks, the Gray, John, and Atwood (1993) CPM-GOMS models, were of considerable economic value in this domain where a second's reduction in average task completion time represents considerable financial savings.

Background on CPM-GOMS

Since the CPM-GOMS methodology and its most noteworthy application in Project Ernestine (Gray, et al.) is the precursor to the present work, some background is important to make the contribution of the present work clear (see also John & Kieras, 1994 for a general discussion of GOMS methodologies). CPM-GOMS is based on the Model Human Processor (MHP) (Card, Moran, & Newell, 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 benchmark task scenarios, or 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 & Kieras, 1994).

Generative Models of Interface Procedures

This paper 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 (Meyer & Kieras, in press, Kieras, Wood, & Meyer 1995; Kieras & Meyer, 1995), and the earlier so-called Cognitive Complexity Theory (CCT) production-system analysis of human-computer interaction (Bovair, Kieras, & Polson, 1990; Kieras & Polson, 1985). EPIC is similar in spirit to the Model Human Processor (MHP) (Card, Moran, & Newell, 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*[2] 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 & Kieras, 1994), 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

[2] 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.

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 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 (Meyer & Kieras, in press-a, b; Kieras & Meyer, 1995).

The work reported in this paper 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 recordings of actual operator performance, was 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 (Wood, Kieras, & Meyer, 1994) 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.

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 sections of the paper describe the EPIC architecture in more detail than previously available, describe how the predictive models were constructed for the telephone operator task, and finally compare the EPIC model predictions to a newly analyzed set of performance protocols for this task.

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, and so is more comprehensive, more formalized, and more detailed than proposals such as MHP, HOS, SAINT, and so forth (see McMillan, Beevis, Salas, Strub, Sutton, & Van Breda, 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, Kieras, & Polson, 1990), ACT-R (Anderson, 1993), and SOAR (Laird, Rosenbloom, & Newell, 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 also 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

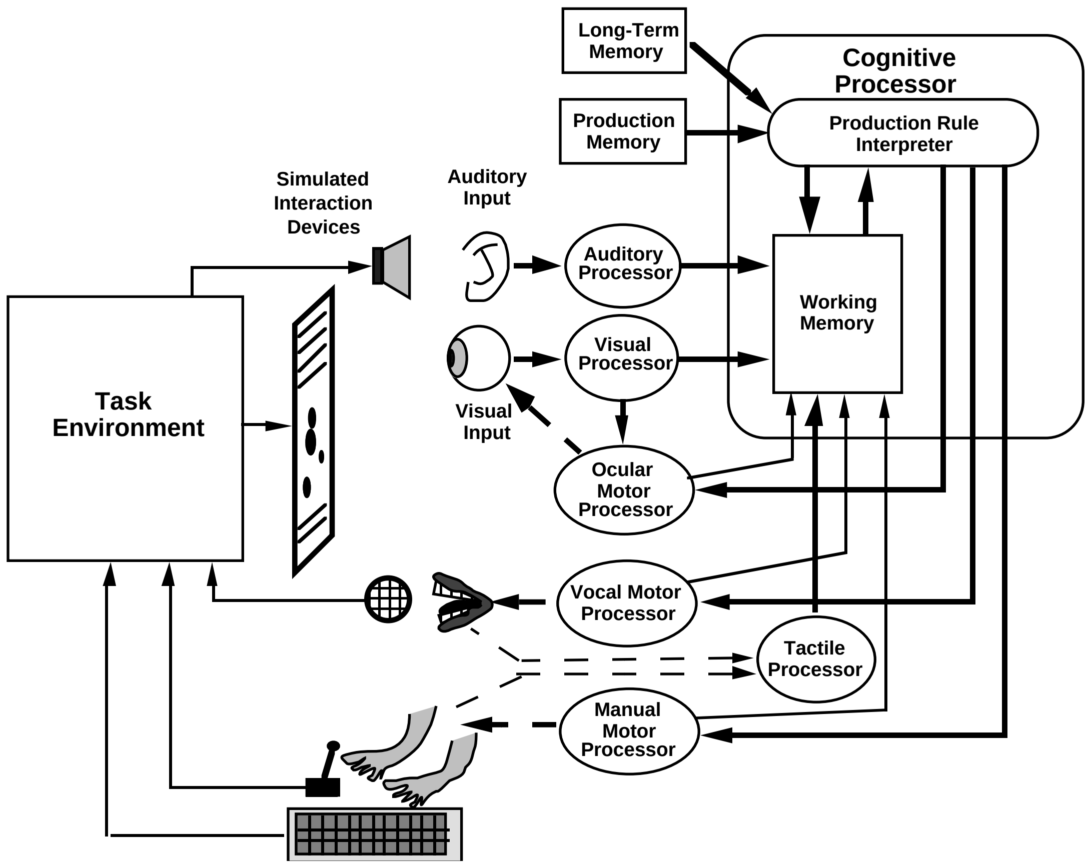

Figure 1. Overall structure of the EPIC architecture simulation system. Task performance is simulated by having the EPIC model for a simulated human (on the right) interact with a simulated task environment (on the left) via a simulated interface between sensory and motor organs and interaction devices. 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.

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 separate perceptual processors with distinct processing time characteristics for each sensory modality, and 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 also conventional working memory items, such as representations of sensory inputs. Each of these processors will be described in more detail below.

Perceptual Processors

The perceptual processors require roughly the same amount of processing time as assumed in the MHP, but have many differences. A single stimulus input to a perceptual processor can produce multiple outputs to be deposited in WM at different times. The perceptual processors in EPIC are simple "pipelines," in that an input produces an output at a certain later time, with no "moving window" time-integration effect as assumed by the MHP. The tactile perceptual processor handles movement feedback from effector organs; this feedback can be important in coordinating multiple tasks (Meyer & Kieras, in press-a, b), but is not used in the models presented in this paper.

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 (radius: 1°), the parafovea (radius: 10°), and the periphery (radius: 60°). Certain information, such as the contents of character strings, is 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 ms), followed later by a representation of sensory properties (e.g., shape, standard delay: 100 ms), 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 ms).

Auditory processor. The auditory perceptual processor accepts auditory input, and 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 ms), then at a later time, an item corresponding to a discriminated frequency of the tone (typical delay: 250 ms), then an offset item (standard delay: 50 ms). 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 s).

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 ms after the acoustic data is 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 paper 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.

Cognitive Processor

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 paper 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, Kieras, & Polson, 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 send a command to a motor processor. Examples will be given later in this paper.

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 ms, but EPIC can run in a mode in which the cycle duration is stochastic, with a standard mean value of 50 ms and all other time parameters scaled to this stochastic value. Unlike the MHP and 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 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 automatically incorporate a gratuitous assumption of limited central processing capacity. Rather, EPIC emphasizes the role of executive process strategies in coordinating perceptual-motor peripherals in order to perform multiple tasks. As shown by the detailed quantitative modeling of a variety of multiple-task data reported in Meyer & Kieras (in press-a, b) and Kieras & Meyer (1995), EPIC and its approach to modeling multiple-task performance has excellent empirical support. However, for the single-task domain in this paper, only limited use is made of this parallel-processing capability.

Working memory. The production system working memory is in effect partitioned into several working memories.[3] 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 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 & Kieras, in press, for more discussion).

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. EPIC's WM may in fact be what is usually termed *verbal working memory*, which is strongly modal, being based on auditory-vocal coding. 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 under-appreciated limitations would have been obscured by gratuitous assumptions about central capacity or working memory limitations (see Meyer & Kieras, in press-a, b, 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.

[3] 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 below.

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 so 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.

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 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 ms 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 ms to initiate the movement, followed by the actual physical movement. The time to physically 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.

An example of motor processor operation. To strike a key using a one-finger *peck* movement style (like that used in "hunt-and-peck" typing), the cognitive processor commands the manual motor processor to perform a peck movement with (e.g.) 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 a form of Fitts' law (see Card, Moran, Newell, 1983, Ch. 2). After the simulated finger hits the key, it is left in the location above the key to await the next movement.

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 ms 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 ms would be required to start the movement. In such a task, the finger is normally positioned on top of a sensitive key, so the mechanical movement to produce the response event is very small, taking only on the order of 20 ms. EPIC would thus produce the same motor processor time (50 ms initiation + 20 ms 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.

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, pattering two-fingers movements one after the other, 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. Of particular interest for this paper is the peck style, whose execution time is given by Welford's form of Fitts' Law (see Card, Moran, Newell, 1983, Ch. 2), with a standard minimum execution time of 100 ms, reflecting that for small movements to large targets, there is a physiologically plausible lower bound for the actual duration of a muscular movement.

Vocal motor processor. EPIC's vocal motor processor is not very elaborated at this time; it is based on the minimal facilities needed to model certain dual-task situations (see Meyer & Kieras, in press). 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 paper, 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 requires only the preparation of two features before execution begins. The actual production of the sound is assumed to be delayed by about 100 ms after initiation, and continues for a time estimated from the data. Further development of the vocal motor processor is planned in the future.

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 ms/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 (1995), EPIC can follow moving objects with a mixture of voluntary and involuntary eye movements.

Modeling Issues

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; 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 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.

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.

Lessons from EPIC Models of Multiple-Task Performance

Multiple possible strategies for parallelism. An immediate insight from the application of the EPIC architecture to multiple-task domains is that there are multiple possibilities for performing task activities in parallel. That is, in the dual-task models using EPIC, the role of the cognitive strategies in coordinating activity between the two tasks is critical to accounting for the observed effects, and in many situations, these strategies are surprisingly subtle and efficient. In dual-task experiments, the person is supposed to complete each of two tasks as rapidly as possible, but the higher-priority task must be completed before the lower-priority task, regardless of the relative speed of the perceptual or motor processing involved in the two tasks. If two tasks require the same motor processor, both perceptual and cognitive processing on the lower priority task can go on while the higher-priority task is allowed to control the motor processor. Once the motor processor has commenced execution of a higher-priority response, the lower-priority task can be given control of the motor processor, thereby honoring the task coordination requirements while maximizing speed. If the two tasks involve different motor modalities, portions of the lower-priority response can be prepared far in advance, so that it can be made more quickly when its turn comes. If the two tasks compete for the use of both the eyes and the hands, the executive rules can dynamically switch control of the two processors between the two tasks so that their processing is interleaved, with little wasted time. Thus, in a dual-task situation, the cognitive processor strategies are responsible for allocating the eyes and the motor processors to the two tasks as needed to maximize overall performance. But in a single multimodal task with a requirement for speed, the task strategy is responsible for ensuring that the individual processors do their work at the right time so as to minimize the total time required for the task.

The need for modeling policies. There are multiple possibilities for how activities in a multimodal task can be overlapped under the EPIC architecture. 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 *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, or aspect of 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.

Some Possible Policies for Overlapping Task Activities

Table 1 lists the modeling issues addressed by the policies in this paper, some possible resolutions of each issue, and the corresponding models presented in this paper. Following this brief summary of the policy issues, each of the models will be presented along with more detail about the policy issues, and how the resolutions of the issues were implemented in terms of the production rules in the models.

Method structure. The first policy issue in Table 1 concerns method structure, where by method is meant a series of procedural steps for accomplishing a goal, as in the GOMS model (Card, Moran, & Newell, 1983; Kieras 1988; John & Kieras, 1994), a standardized framework for describing procedural knowledge. A typical GOMS analysis breaks a task down into subgoals, and for each goal or subgoal, there is a method that when executed, will accomplish the goal. How should the methods be represented in terms of production rules? The first resolution listed in Table 1 is that the goal and method structure is hierarchical, reflecting the hierarchical task decomposition with multiple subgoals and subprocedures; the second resolution is that there is a single flat, or non-hierarchical method. Note that just because the task analysis 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 extreme amounts of practice. For present purposes, four models were constructed assuming hierarchical methods, and two assuming flattened methods.

Inter-motor coordination. The motor coordination policy issue concerns how the cognitive processor coordinates the different motor systems. There are two different subissues. The first involves eye-hand coordination. All the present models followed the approach that when making keystrokes, the eye must be moved in advance to the vicinity of the target keys, and 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, and 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. The multiple-task models may have such inter-motor parallel activity, and emphasize the role of the cognitive strategies in enforcing the coordination required by the task. Two resolutions of this subissue are represented; one model follows a simple strategy of not using the motor parallelism, while the other model implemented the opposite resolution, using the parallel capability of the motor processors.

Intra-motor coordination. These policy issues concern how a single motor processor is used by the cognitive processor. A simple resolution 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 resolution, used by most of the present models, is to prepare the next movement while the current one is executing, enabling the next movement to start execution much sooner.

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. Two subissues appear in

Table 1. Policy issues addressed in each model.

	Models					
	Hierarchical				Flattened	
Policy Issue	**Fully Sequential**	**Motor Parallel**	**Prepared Motor-Parallel**	**Premove/Prepared Motor-Parallel**	**Motor Parallel**	**Premove/Prepared Motor-Parallel**
Method Structure						
Hierarchical methods.	X	X	X	X		
Single flat method.					X	X
Inter-Motor Coordination						
Move eye to target before keystrokes.	X	X	X	X	X	X
Overlap eye, hand, and vocal movements.		X	X	X	X	X
Intra-Motor Coordination						
Complete all movement phases before executing the next method step.	X					
Prepare next movement while current movement is executing.		X	X	X	X	X
Movement Anticipation						
Prepare features for next movement as far in advance as possible.			X	X		X
Pre-position the eyes and hands to next locations as far in advance as possible.				X		X

how EPIC can implement movement anticipation: First, 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. Second, 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 present models represent three different resolutions to these two subissues: no movement anticipation, advance preparation only, or both premovement and advance preparation.

MODELING THE TELEPHONE OPERATOR TASK

Task Description

Briefly, the tasks analyzed in our work are 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 paper, 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 *0+* calls. 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 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.

Rationale for Using the 0+ Call Type

The reasons for selecting the 0+ call type for this work needs some explanation in the context of the earlier Gray, John, and Atwood (1993) Project Ernestine. Project Ernestine addressed practical questions of how a new workstation compared with the current workstation, and so considered of 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.

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 is coordinate with the computer system in 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. However, including externally-determined event

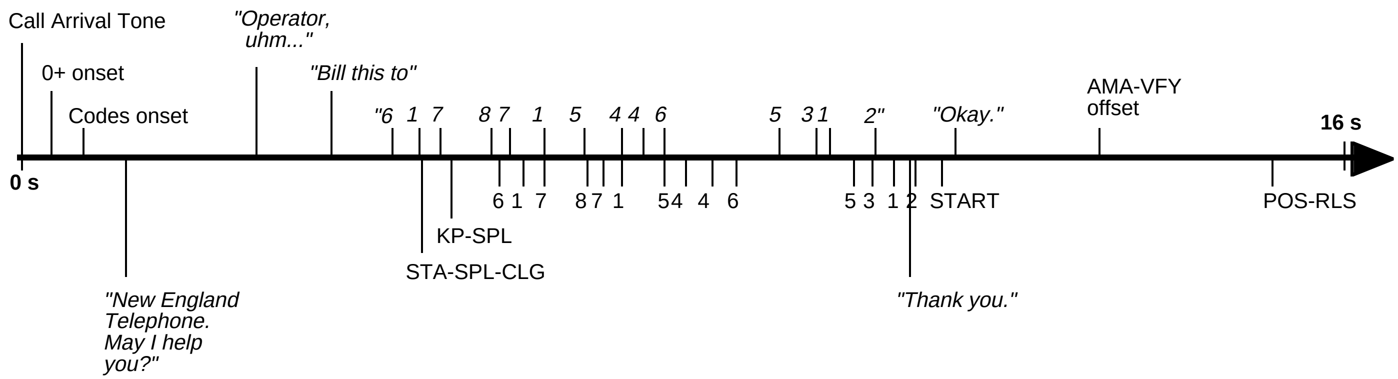

Figure 2. Timeline of a typical task instance from the data. The horizontal time scale covers the range from zero to sixteen seconds. Events above the line are the customer's speech (italics) and events displayed by the computer workstation. Events below the line are the operator's speech (italics) and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

times in the to-be-predicted total task time 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 on externally determined events in the to-be-predicted times.

Strong tests of the models. 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 0+ 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.

A Set of A-Priori Models

To simulate the operator's performance of the selected 0+ type of telephone operator tasks, the task environment model was programmed to generate simulated displays and customer input for this class of calls, and EPIC was "programmed" with a set of production rules capable of performing all possible instances of the task. Under direction of the cognitive processor rules, the perceptual and motor processors move the eyes around, perceive stimuli on the operator's workstation screen, and reach for and strike keys. The time these activities require is determined by the perceptual and motor processors, but the production rules can arrange to overlap some of the activities in order to complete the entire task as rapidly as possible. Some possible policies on this overlapping, which we included for testing alternative models, are listed in Table 1.

Since a variety of possible policies for overlapping are meaningful in the telephone operator task, our approach was to begin to develop these policies by proposing several models and then determining which could account for performance. Accordingly, a series of models, those listed in Table 1, was constructed to represent points on a policy continuum starting with a non-optimized 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 a set 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. Each of these models and the corresponding policies will be described with example production rules in the following sections. Since the operators in this task domain are normally 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 policies appears instead to provide the best fit.

As shown in Table 1, only a small subset of the possible combinations of policy features 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, Kieras, and Polson (1990) CCT production rule models, and so provides a conceptual baseline for the more complex models. The other Hierarchical models differ in one policy feature each to help understand the effects of each feature. 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.

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 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, John, & Atwood, 1993; John & Kieras, 1994). These assumptions were as follows: (1) 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 adequately 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. (2) 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 and KP-SPL keys. If digits are heard before such an utterance, it is assumed that the same call type is intended. (3) Along the lines described above with Table 1, 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, but eye movements to individual digit keys are not required. However, because of the high frequency of the striking the POS-RLS key, no eye movement to it or visual acquisition of it is required. (4) The operator says "thank you" after the customer has finished speaking, regardless of whether other actions are still under way. (5) The operator knows that the call has been verified when a certain designated screen event occurs.

Hierarchical Method Models

The Hierarchical Fully-Sequential model. This first model and its production rule representation will be described in some detail, because it represents the simplest model policy; all of the other models were more optimized versions of it, and can be explained very briefly after this detailed introduction. As shown in Table 1, this model assumed hierarchical methods, but was strictly sequential in that the policies followed for both inter- and intra-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 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.

The Hierarchical Fully-Sequential model was based on a straightforward GOMS model for the task using the NGOMSL approach (Kieras, 1988, 1994). This model describes task procedures as a hierarchical set of methods consisting of sequential executed actions, and which operate the motor processors strictly sequentially. Figure 3 shows the hierarchy of Goals and Methods for the GOMS model for the task, and Figure 4 contains an excerpt of the NGOMSL methods. Constructing this GOMS model was a routine activity needing no special explanation here (see John & Kieras, 1994), and was performed as part of the preliminary modeling work described by Wood, Kieras, & Meyer (1994). Then a set of production rules were written to implement this GOMS model in a style similar to the CCT templates described in Bovair, Kieras, and Polson (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, Kieras, and Polson (1990); each is in the format `(<name> IF <condition> THEN <actions>)`. The condition is a list of *clauses*

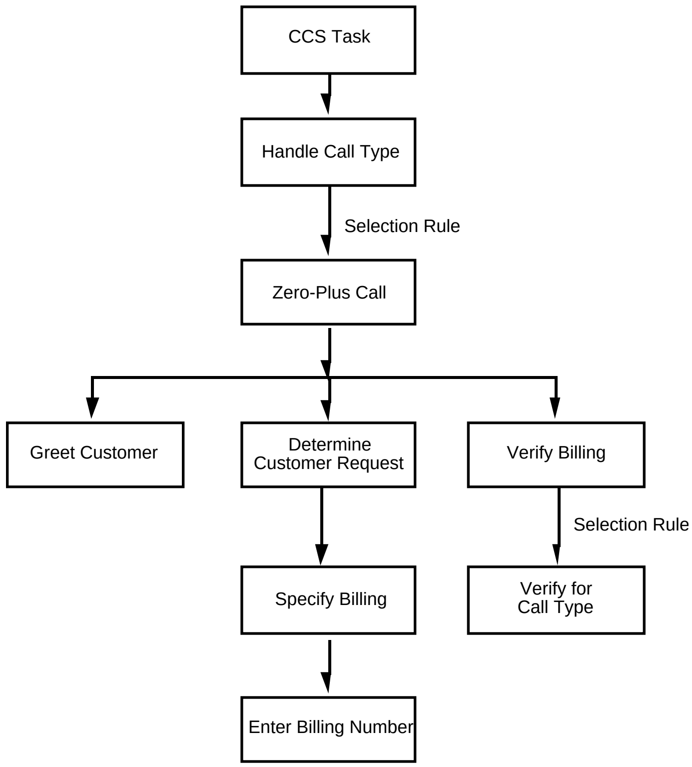

Figure 3. The hierarchy of goals and methods in the GOMS model for the telephone operator task. Connections labeled as selection rules indicate possible additional subgoals.

(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 (`???`) or a variable (a term denoted by a symbol with 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.

The actions in the rules usually include removing the current step clause from the working memory database with `DELDB` and adding with `ADDDB` the step clause corresponding to the next step in the procedure. 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 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 start-up 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 underway 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

```
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
```

Figure 4. Excerpt of NGOMSL Methods for a Subset of Telephone Operator Tasks

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

```
(*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))))
```

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

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 constitutes a policy for representing the GOMS methods in EPIC 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 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 & Kieras, 1994, for 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 (1988) 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 also updates a "pointer" in WM to the next speech item in auditory working memory to be processed. 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.

```
(*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))))
```

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

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 1, 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 takes advantage of the motor processor's ability to prepare the next movement while a prior movement is currently underway.

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>`

Fully-Sequential Policy

```
(*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 Policy

```
(*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))))
```

Figure 7. Example modifications made to transform the Hierarchical Fully-Sequential rules (top panel) into the Hierarchical Motor-Parallel form.

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 policy means that the operator 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.

The Hierarchical Prepared Motor-Parallel model. This next model took further advantage of the motor processors. As shown in Table 1, 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 ms) 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 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

```
(*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))))
```

Figure 8. Example rules for preparing a movement in the Hierarchical Prepared Motor-Parallel Model.

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.

The Hierarchical Premove/Prepared Motor-Parallel model. Table 1 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 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.

```
(*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))))
```

Figure 9. Example rules for making a premove to a key followed by advance preparation for a subsequent keystroke in the Hierarchical Premove/Prepared Motor-Parallel Model.

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, Rosenbloom, & Newell, 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, John, & Atwood, 1993), in that the methods consist simply of sequences of operators, with no hierarchical submethod structure and consequently no cognitive execution overhead (see John & Kieras, 1994, for more discussion of this distinction). As mentioned above, only two models of this type were developed here.

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 house-keeping 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 1 shows, the flattened methods are the only modeling policy feature 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 chained to a rule that invoked a submethod via a selection rule for the 0+ 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+, 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.

The Premove/Prepared Flattened Motor-Parallel model. Finally, as Table 1 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.

COMPARISON OF THE MODELS TO EMPIRICAL DATA

Observed and Predicted Times

The basic question concerns how well the *a priori* constructed models predict actual total task performance time. We selected 0+ call type task instances from the videotaped task performances of experienced operators collected, but not analyzed, during the Gray, John, and Atwood (1993) Project Ernestine. In these 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 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. The video and audio recordings of the selected task instances were digitized at full frame rate, and the times of the roughly 50

```
(*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))))
```

Figure 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.

individual events (display changes, words of speech, and keystrokes) in each task instance, were determined to the nearest video frame (1/30 sec, 33 ms). 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.

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. Only these internally-determined events were used to compare the models to the data.

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. For each 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. 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.

Each model performed all of the task instances and generated however many keystrokes the task instance required (e.g entering as few as four digits to as many as 14). The exact timing of the keystrokes can depend in subtle ways on exactly what happens to be on the critical path during a task instance, which depends in turn on the exact timing of the preceding input and output events.

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 (described above as *standard* values), and a single task-specific value determined during the preliminary model-fitting work in this task domain (Wood, Kieras, & Meyer, 1994), namely the time to recode a spoken digit to the name of a key (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.

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, Moran, & Newell, 1983; John & Kieras, 1994). 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 ms each) and homing operators for movements of the right hand to and from the keypad (400 ms each).

RESULTS

Prediction of Total Task Execution Times

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 key. However, this definition of total task time is a poor choice for testing the models. First, as mentioned both by our informants and Gray, John, and Atwood (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 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.

Goodness of fit measures. One way to assess the goodness-of-fit of the model predictions to the data is to calculate regression equations and coefficients. Despite the small sample size, all of the models, even the Keystroke-Level Model, accounted for a statistically significant ($p < .05$) 83% or more of the variance (r^2) in the task execution times. 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.

However, 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. A measure of the actual difference between the predicted values and the observed values is more informative. However, the mean error would be 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 were 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 that predictions accurate within 10-20% are useful for design purposes.

Model accuracy. Table 2 lists the average absolute error of prediction for each model; Figure 11 shows the results graphically with the dotted line showing 10% error. The average absolute error of prediction ranges from 7% for the Hierarchical Motor-Parallel model, to 14% for the worst-fitting EPIC model, to 28% for the Keystroke-Level Model. All of the EPIC models appear to be usefully accurate in predicting total task execution time 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), so they all do a reasonable job of predicting overall task execution time. In contrast, the Keystroke-Level Model is much less accurate because it does not overlap any activities. Even at the small sample sizes, the accuracy of the Keystroke-Level Model is significantly less than each of the EPIC models ($p < .05$).

The use of the absolute error of prediction does conceal an important fact: the Hierarchical Fully-Sequential Model and the KLM both seriously overpredicted the task execution times because of their failure to allow for overlapping, while the other models underpredicted the execution times. Underprediction of execution time is expected from an engineering model since the model will normally produce performance that is more efficient than that of a typical human (cf. Gray, et al.). Thus, both the relatively accurate Hierarchical Fully-Sequential Model and the inaccurate KLM, are fundamentally incorrect representations of how the task is done. This leaves the Hierarchical Motor Parallel model as the best contender, being both simple and the most accurate.

The major surprise in the results is that the Hierarchical Motor-Parallel model, which is only moderately efficient, appears to be more accurate than the remaining Hierarchical models that were highly optimized. Unfortunately, due to the small sample sizes for the total times, the differences in accuracy between this simple model and the more optimized models are not statistically reliable. Nonetheless, the optimized Hierarchical models are faster than the highly experienced human operators, and more so than the Hierarchical Motor-Parallel model. This result suggests that while operators take advantage of the parallel preparation and execution capabilities of their motor processors to speed up their performance, they make little use of pre-positioning the eyes and hands in advance. This could result from such efficiencies not being required in the task environment, as discussed more below, or else the assumed strategies for the anticipations could be incorrect (e.g., the STA-SPL-CLG key might not be the correct keystroke to anticipate).

The flattened method models also appear to be too fast, but the difference is not large, being only about 200 ms of predicted time difference for the Hierarchical Motor Parallel model and the Flattened Motor Parallel model. A *post-hoc* 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, and so the cognitive overhead does not contribute much to the total task execution time. Thus the difference between the two method structures does not appear to be identifiable in these data. A larger sample would probably not help, since the problem stems from how the task structure relates to the strategies and modeling policies. A different type of task would be required to clearly distinguish these two families of models.

Table 2. Quality of fit between models and observed execution times.

Model Type	Error
Hierarchical Methods	
Fully-Sequential	9%
Motor-Parallel	7%
Prepared Motor-Parallel	10%
Premove/Prepared Motor-Parallel	12%
Flattened Methods	
Motor-Parallel	10%
Premove/Prepared Motor-Parallel	14%
Keystroke-Level Model	28%

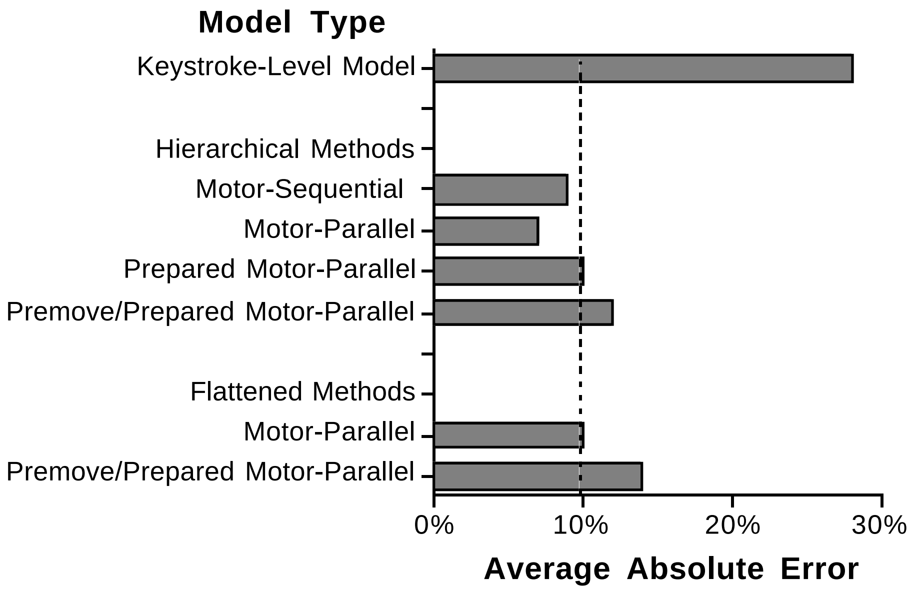

Figure 11. Average absolute error of prediction of total task execution time for each model as a percentage of observed execution time. A dotted line is shown at 10% error.

Comparison of EPIC models to CPM-GOMS. A natural question is how the EPIC models compare to the Gray, John, and Atwood (1993) CPM-GOMS models. A fundamental difference is that 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 generate predictions for each of the eight task instances. Thus the EPIC models attempt to account for all of the task instances with a single task strategy, and do so with an average absolute error of 7% to 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 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.

Implications of Individual Keystroke Time Predictions

Speech recognition delay and input rate. The EPIC models not only predicted total task times, but also 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 moderately well by the best-fitting Hierarchical Motor-Parallel model. Detailed examination shows that in the observed task instances, 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 parameter value is accurate enough make the total task time reasonably correct.

Unfortunately, empirical data on transcribing auditory digit strings to keystrokes has never been reported, meaning that an independent high-quality estimate of the auditory recoding time parameter is not available. The accuracy of the individual keystroke time predictions can be greatly improved by estimating the digit recoding time based on the shortest observed digit keystroke latencies, yielding a value of 400 ms rather than the 150 ms previously assumed, and by altering the task strategy so that the STA-SPL-CLG keystroke is not made until the first digit is heard, when the operator could be positive that the customer has finished specifying the billing type of the call. However, these changes render the model *post-hoc*, and thus unsuitable for the present goals of this paper. However, such modeling work is underway, and could shed light on important properties of this and related tasks.

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 the Gray et al. work, 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.

What controls the POS-RLS keystroke? As discussed above, POS-RLS, the last keystroke in the task, is problematic. Gray, John, and Atwood (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 the 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, John, and Atwood, 1993).

Given that the system verification process is relatively long, the operator should be idle at the time of the relevant screen event, and so the latency of the POS-RLS keystroke should depend 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 ms 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 pre-positioning or preparation, predicts a reaction time to this same event averaging only about 727 ms. 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 to 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-R:LS 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 a prediction 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.

CONCLUSIONS

Some EPIC models for a high-performance task were constructed using *a priori* task analysis, construction 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. Of particular interest was the result that the most accurate model was rather easy to construct because it followed very simple modeling policies. 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. Furthermore, the accuracy of one of the simplest models suggests that the level of optimization of the operator strategy is not be very high; although the operators are very practiced and execution speed is important, the task speed is limited in ways that may make highly optimized strategies of little value.

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 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 (see John & Kieras, 1994), as the EPIC architecture stabilizes and 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.

ACKNOWLEDGEMENT

This work was supported by the Office of Naval Research Cognitive Sciences Program under grant N00014-92-J-1173, and NYNEX Science and Technology, Inc., who also provided the data videotapes and support for the digitization and transcription of the protocols.

REFERENCES

Anderson, J. R. (1976). *Language, memory, and thought.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Anderson, J. R. (1993). *Rules of the mind.* Hillsdale, N.J.: Erlbaum.

Anderson, J. R. (1987). Skill acquisition: Compilation of weak-method problem solutions. *Psychological Review*, 94, 192-210.

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

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

Gray, W. D., John, B. E., & Atwood, M. E. (1993) Project Ernestine: A validation of GOMS for prediction and explanation of real-world task performance. *Human-Computer Interaction*, 8, 3, pp. 237-209.

Hallett, P. E. (1986). Eye movements. In K.R. Boff, L. Kaufman, & J. P. Thomas (Eds), *Handbook of perception and human performance*. Volume 1, 10-1 - 10-112. New York: Wiley.

John, B. E. (1988). Contributions to engineering models of human-computer interaction. Doctoral dissertation, Carnegie Mellon University.

John, B. E. & Kieras, D. E. (1994). The GOMS family of analysis techniques: Tools for design and evaluation. Carnegie Mellon University School of Computer Science Technical Report No. CMU-CS-94-181. Also appears as the Human-Computer Interaction Institute Technical Report No. CMU-HCII-94-106.

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

Kieras, D. E. (1994a). A guide to GOMS task analysis (Spring, 1994). Unpublished manuscript, University of Michigan.

Kieras, D. (1994b). GOMS Modeling of User Interfaces using NGOMSL. Tutorial Notes, CHI'94 Conference on Human Factors in Computer Systems, Boston, MA, April 24-28, 1994.

Kieras, D.E., & Meyer, D.E. (1994). The EPIC architecture for modeling human information-processing: A brief introduction. (EPIC Tech. Rep. No. 1, TR-94/ONR-EPIC-1). Ann Arbor, University of Michigan, Department of Electrical Engineering and Computer Science.

Kieras, D. & Meyer, D. (1995). Predicting human performance in dual-task tracking and decision making with computational models using the EPIC architecture. Proceedings of the *First International Symposium on Command and Control Research and Technology*, National Defense University, June. Washington, D.C.: National Defense University.

Kieras, D. E., & Polson, P. G. (1985). An approach to the formal analysis of user complexity. *International Journal of Man-Machine Studies*, 22, 365-394.

Kieras, D. E., Wood, S. D., & Meyer, D. E. (1995). Predictive engineering models using the EPIC architecture for a high-performance task. In *Proceedings of CHI*, 1995, Denver, Co, USA, May 7-11, 1995. New York: ACM.

Laird, J., Rosenbloom, P, & Newell, A. (1986) *Universal subgoaling and chunking*. Kluwer Academic Publishers: Boston.

Meyer, D.E., & Kieras, D.E. (1994). EPIC computational models of psychological refractory-period effects in human multiple-task performance. (EPIC Tech. Rep. No. 2, TR-94/ONR-EPIC-2). Ann Arbor, University of Michigan, Department of Psychology.

Meyer, D. E., & Kieras, D. E. (in press, 1996). Computational modeling of human multiple-task performance: I. The EPIC architecture and strategic response-deferment model. *Psychological Review*. (a).

Meyer, D. E., & Kieras, D. E. (in press, 1996). Computational modeling of human multiple-task performance: II. Executive scheduling strategies and psychological refractory period phenomena. *Psychological Review*. (b).

McMillan, G. R., Beevis, D., Salas, E., Strub, M. H., Sutton, R., & Van Breda, L. (1989). *Applications of human performance models to system design*. New York: Plenum Press.

Rosenbaum, D. A. (1991). *Human motor control.* New York, Academic Press.

Wood, S., Kieras, D., & Meyer, D. (1994). An EPIC model for a high-performance task. Poster presented at CHI'94 ACM Conference on Human Factors in Computing, Boston, April 25-28.

APPENDIX

The exact set of models, data, and evaluations of prediction accuracy appearing in the Gray, John, and Atwood (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. They then 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, calculating the average absolute error from their reported results gives about 26% and 25%error respectively. These relatively large prediction errors are due to benchmark selection error: the benchmark tasks on the current workstation turned out not to correspond very closely to the field data for the same workstation. That is, if the benchmark tasks were used to predict the observed field data times for the same call type and workstation, the average error was about 8%, and the average absolute error was just under 25%. 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 0+ calls as a separate type from ordinary 0+ 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.