

How Apex Automates CPM-GOMS

Michael Freed^{*◦}, Michael Matessa*, Roger Remington*, Alonso Vera*

* MS 262-4 NASA Ames Research Center / Moffett Field, CA 94035 / USA

◦ IHMC / University of West Florida / 40 South Alcaniz St. / Pensacola, FL 32501 / USA

Abstract

Although acknowledged to be a powerful technique for predicting skilled behavior, CPM-GOMS (John and Kieras, 1996) is not widely used in interactive system design. We hypothesize that this is because creating CPM-GOMS models requires extensive expertise and is tedious and error-prone. To address these problems, we used the Apex architecture (Freed, 1998b) to automate critical parts of the CPM-GOMS analysis process. This paper describes how modelers represent CPM-GOMS models in Apex and how Apex translates those representations into predictions of skilled behavior. This information should prove helpful in reproducing CPM-GOMS capabilities in other cognitive architectures.

Introduction

This paper describes an approach to applied human performance modeling based on the automatic scheduling of low-level cognitive, perceptual, and motor (CPM) resources that underlie actions such as moving and clicking a mouse, pressing a button, or speaking a phrase. Specifically, our computational architecture, Apex, automates a modeling technique known as CPM-GOMS (Gray, Atwood and John, 1993; John and Kieras, 1996). The practical value of CPM-GOMS is that it has been shown to provide very accurate zero-parameter predictions of human performance in highly practiced tasks. Its capacity for making accurate predictions for tasks of practical significance is well illustrated by its use in Project Ernestine (Gray, Atwood and John, 1993). A CPM-GOMS model was used to generate predictions of the average time for a telephone operator to handle a customer transaction using newly designed equipment and procedures. Accurately, but contrary to the system designer's expectations, the model predicted an average task duration .63 seconds *greater* than that required with existing equipment. With each second of transaction time costing NYNEX three million dollars annually, purchasing the new equipment would have been a costly mistake.

Given successes like project Ernestine, CPM-GOMS is acknowledged in the human-computer interaction community as a powerful predictive technique. However, CPM-GOMS is also considered an onerous and error-prone process that requires a great deal of specialized expertise to apply (John, Vera, Matessa, Freed and Remington, 2002). These factors have almost certainly inhibited CPM-GOMS from coming into widespread use.

Elsewhere (John et al., 2002) we have described the benefits of using the Apex cognitive architecture (Freed, 1998b) to automatically construct CPM-GOMS models, and demonstrated the validity of the resulting human performance predictions. Here we describe in detail how Apex automates CPM-GOMS. In keeping with its goal of providing a simplified modeling framework for engineering domains, Apex provides a flexible behavior representation language that permits the Apex modeler to represent behavior at a relatively high level, thus facilitating model development. Underlying this high-level language is a complex action selection architecture that selects and schedules tasks and resources. A more detailed treatment of how the Apex architecture interprets the high-level representation of task knowledge will be useful in developing human performance models in Apex. The description also attempts to highlight abstract properties needed for any automation of CPM-GOMS, and should facilitate its implementation in other computational architectures.

GOMS and CPM-GOMS

CPM-GOMS (John and Kieras, 1996) is an extension of GOMS (Card, Moran and Newell, 1983), a methodology for predicting how long a person will take to carry out a well-learned machine-interface task. GOMS represents tasks in terms of Goals, Operators, Methods, and Selection rules. For example, an analysis of the task of deleting a file from one's computer directory structure might start with a top-level **goal** such as (*delete-file oldpic.jpg*). **Methods** are generalized action sequences for decomposing goals into subgoals. For example, a method for goals of type *delete-file* might consist of the sequence: visually find target file icon, move mouse pointer to icon, hold mouse button, move mouse pointer to trash-icon, release mouse button. **Selection rules** are used to choose between alternative methods for accomplishing a goal – e.g. a rule might choose between the file removal method above and a text-based method depending on whether a graphical or command-line interface is being used. Method-based goal decomposition continues recursively on (sub)goals, stopping if the goal corresponds to a primitive **operator**. What constitutes an operator is decided by the modeler based on how fine-grained an analysis is desired. Usually, GOMS modelers define operators at the level of interface actions such as clicking a mouse or reading a value off a display.

The process of recursively applying methods to non-primitive goals (those that do not correspond to an operator) produces a goal hierarchy. Ordered by a depth-first traverse, the leaf nodes of this hierarchy form a sequence of operator-level actions ($o_1 \dots o_n$) whose total execution time (S_o) is the predicted total time for the task. This approach to predicting task duration makes strong assumptions about operator independence – that operators are executed in strict sequence and that the specific nature of an operator has no effect on the time required to execute other operators in the sequence. These assumptions do not hold for highly practiced sequences where the execution of adjacent operators may overlap in time and the degree of overlap can depend on operator order and identity (cf. Agre and Shrager, 1990). As a result, the sum duration of operators whose assigned individual durations are correct in isolation will tend to predict excessive overall task duration.

The CPM-GOMS method extends GOMS to account for overlapping execution. A CPM-GOMS analysis is derived from a GOMS analysis of the same task. However, the interface-level behaviors in the resulting sequence are no longer considered operators. Instead they are **template-level goals**. We denote the template-level goal sequence ($t_0 \dots t_n$). Unlike operators, which cannot be further decomposed, and unlike regular goals in the goal hierarchy which are decomposed using methods, a template-level goal is decomposed using a new structure called a **template**.

A template specifies decomposition of an interface-level goal into discrete cognitive, perceptual and motor (CPM) actions consistent with the Model Human Processor (e.g. Card, Moran and Newell, 1983); see also Gray and Boehm-Davis, 2000). Each of these actions is considered an operator and each is considered a user of a particular cognitive, perceptual or motor resource. Operators in a template can be carried out concurrently if they use different resources and are not order-constrained by the logic of the task. For instance, two eye-movement operators cannot be carried out concurrently, nor can an eye-movement and a hand-movement if the former is used to visually identify a target for the latter.

In CPM-GOMS, operators from one template-level goal in the sequence can sometimes begin before all operators from a preceding template-level goal have finished. Several kinds of constraints (described below) govern the interleaving of operators from different templates. The essence of the interleaving phenomenon is that activities specified by a template do not use all resources all of the time; idle time (slack) in the use of a resource by one template's operators represents an opportunity for operators from a later template to "slip back" and begin execution. Interleaving at the level of CPM-GOMS operator-level goals corresponds to overlap in the execution of higher-level goals – i.e. at the level of classic GOMS

operators – and thus accounts for the different predictions of the CPM-GOMS and classic GOMS approaches.

The problem of determining how to interleave operators from a sequence of template-level goals can be formulated as a scheduling problem with three kinds of constraints determining what constitutes a correct schedule. **Logical constraints**, where one action is required to specify or enable another and therefore must precede it, may apply between operators within a single template or across templates. Within a template, logical constraints can be represented as explicit orderings between operators. To enforce logical constraints across templates requires representing each operator's preconditions and effects. A logical constraint exists if the effect of executing an operator from an earlier template is required to satisfy a precondition for an operator of a later template. Since CPM-GOMS allows operators from one template to slip back into the temporal scope of earlier templates, enforcing cross-template logical constraints is required to guarantee correct behavior.

Unary resource constraints specify that when two operators use a non-sharable, non-depletable resource (e.g. the left-hand), they cannot be carried out concurrently. Within a template, operators requiring the same resource must be explicitly sequenced. Across templates, operators must follow a *template precedence rule* – i.e. given template sequence $t_1 \dots t_n$, operator A from template-level goal t_i that requires resource R, operator B from t_j that also requires R, and $i < j$, operator A has precedence. An important nuance in applying this rule applies in situations where the earlier template has an interval of slack time immediately prior to A that is not long enough for B to run to completion. The template precedence rule extends to this case; B cannot be scheduled for that interval.

Slack exclusion constraints are restrictions a template places on the use of slack time by operators that are brief enough to fit into a slack interval but have some "undesirable" property. Templates in the model described by John et al. (2002) employed a slack exclusion constraint on cognitive operators representing the initiation of a motor response. In particular, the model employed a *cognitive initiation exclusion rule* defined as follows: given a set of operators $A_1 \dots A_m$ from template-level goal t_i that all use resource R, operators B and C from template-level goal t_j where C uses R and B represents a cognitive action that initiates action in C, and $i < j$, B cannot execute until $A_1 \dots A_m$ are complete. This rule contributed to very accurate predictions in the referenced model; however, its generality and scientific basis are a topic of ongoing research.

So far, we have provided an architecture-independent characterization of the CPM-GOMS framework implemented in Apex. The remainder of

the paper will describe how task knowledge is represented in Apex and how these representations can be made to meet the specific requirements of a CPM-GOMS analysis as described above.

Apex

When performed by hand, the goal decomposition and operator scheduling process needed to generate a correct CPM-GOMS analysis is difficult and time-consuming. Apex provides a conceptual and computational framework for formalizing and automating some of the most demanding parts of the process. Apex is a software tool for simulating the behavior of intelligent agents, especially human agents (Freed 1998b). The conceptual approach taken in Apex is to treat the intelligent agent as resource-limited, and to provide capabilities needed to model how the agent allocates its limited resources to accomplish a set of tasks.

The agent architecture incorporates a plan execution system (Firby 1988; Pell et al. 1997) that provides capabilities needed for CPM-GOMS such as hierarchical task decomposition and enforcement of logical preconditions. An integrated dispatch scheduler (Zweben and Fox, 1994) and other mechanisms (Freed, 1998a, 2000) provide the ability to allocate resources based on priority determinations and constraints. This section describes how task knowledge in Apex invokes and parameterizes these mechanisms.

```
(procedure
(index (delete-file ?file using mouse))
(profile right-hand)
(step s1 (find-and-grasp mouse))
(step s2 (vis-locate ?file icon => ?icon))
(step s3 (mouse-move to ?icon) (waitfor ?s1 ?s2))
(step s4 (mouse-drag ?icon to trash) (waitfor ?s3))
(step ct1 (terminate ?self) (waitfor ?s4))
(step ct2 (reset) (waitfor (interrupted ?self))))
```

Figure 1: A procedure.

The central construct in Apex's task knowledge notation, the **procedure**, is used to represent different kinds of "how-to" including CPM-GOMS methods, templates and operators. Every procedure includes at least an **index** clause and one or more **step** clauses. The index identifies the procedure and specifies the class of goals for which it is appropriate. Each step clause describes a subgoal or auxiliary activity.

Steps are concurrently executable unless otherwise specified. A **waitfor** clause is used to indicate preconditions. Goals created with waitfor preconditions become eligible for execution (enabled) only when all the events specified in the waitfor clause have occurred. Thus, goals created by the steps labeled *s1* and *s2* in Figure 1 begin enabled and may be carried out concurrently. The remaining steps specify pending goals – i.e. goals that are created with

unsatisfied preconditions and therefore cannot execute right away.

Events arise primarily from two sources. First, perceptual processes produce a stream of events to represent new or updated observations of the external world. Second, the agent architecture generates events to reflect the status of goals it is executing or considering for execution. Two such events are particularly important for CPM-GOMS modeling. First, the architecture generates events of the form term (*terminated <goal>*) whenever a goal completes. Steps with waitfor clauses such as (*waitfor ?s4*), an abbreviation of (*waitfor (terminated ?s4)*), specify that the termination (completion) of one goal is a precondition for starting another. This constitutes an ordering constraint. In Apex CPM-GOMS models, task logic constraints are represented as explicit ordering constraints.

Another important kind of event signals that the execution of a goal has been interrupted in order to carry out a conflicting goal with higher priority. When an interruption occurs, the architecture generates an event of the form (*interrupted <goal>*). This can trigger contingency handling behaviors represented by steps that wait for specified goals to become interrupted. For example, the step labeled *ctl2* in Figure 1 specifies restarting the from the beginning of the procedure if the *delete-file* goal is interrupted, rather than trying to pick up at the point where the task was interrupted (Apex's default behavior). Interruption-handling behaviors are specified in Apex representations of CPM-GOMS operators.

Apex automatically allocates non-sharable resources among competing goals. A goal's resource requirements are determined when all of its non-resource (waitfor) preconditions have been satisfied and a procedure for carrying out the goal has been selected. The procedure's **profile** clause specifies what resources the goal needs before it becomes eligible for execution. Resource preconditions are satisfied when the architecture determines that the goal is either the sole competitor or highest priority competitor for all of the resources it needs. A profile clause specifies resources a goal needs from the moment it begins execution until the time it is complete. The step action **hold-resource** can be used to assert a resource requirement that arises during execution. For instance, if (*step spcl (hold-resource cognition) (waitfor ?s3)*) were added to the above procedure, this would cause the executing goal (*delete-file oldpic.jpg*) to require the "cognition" resource in addition to the "right-hand" resource (the latter requirement having been established by the profile clause). Adding a new resource requirement during execution triggers a new resource competition, possibly resulting in the goal failing to get the resources it needs and thereby interrupted. The step

action **release-resource** is used to remove an existing requirement prior to goal completion.

Apex can resolve goal competitions based on a number of factors including, e.g., proximity of goal-relevant deadlines, expected cost of interruption and, for repeated tasks, time since last iteration (see Freed, 1998a). However, for CPM-GOMS analyses, the most important factor is *precedence* which is specified (optionally) using a **rank** clause. Goal A has precedence over goal B, and thus has priority over B in competition for a resource if there exists goals A' and B' where A' is an ancestor of A, B' is an ancestor of B, A' and B' are siblings, both A and B have been explicitly assigned ranks (within the lexical scope of the procedure from which they originated) and $\text{rank}(A) < \text{rank}(B)$.

Specifying CPM-GOMS models in Apex

A CPM-GOMS analysis can be thought of as taking place in 3 phases. First, a modeler represents the task of interest in terms of methods, selection rules and a high-level goal. Task-independent templates and operators may also have to be represented, though in many cases, this step will not be necessary. Instead, required templates and operators may be available in a library of reusable behavior representations defined during previous modeling efforts (Matessa et al., 2002)¹. In the second phase, methods and selection rules are used to generate a goal hierarchy, with the leaf-level elements forming a sequence of template-level goals. Third, templates are used to decompose each goal in the sequence into a set of operators and operator scheduling constraints. The output of the analysis is a schedule that includes operators from all goals in the sequence, meets all constraints and is otherwise optimal (minimum duration). The schedule can be represented in the form of PERT chart, which graphically represents resource usage over time.

Apex automates the second and third phases and automatically generates a PERT chart, allowing the modeler to focus solely on representing task-relevant behaviors. This section describes how an Apex modeler represents methods, templates and operators. Examples of these structures, illustrated in Figure 2, are adapted from the Apex CPM-GOMS model of an experimental task (John et al., 2002) in which human subjects used a mouse to operate a simulated automatic teller machine. Templates developed for an entirely different task (Gray and Boehm-Davis, 2000) were reused in this model.

Representing Methods

In GOMS and CPM-GOMS, a method is a sequence of steps representing one way to achieve a specified

type of goal. However, step-ordering in a GOMS method means something quite different from step-ordering in a CPM-GOMS method. In the former case, if step B follows step A, then all subgoals and operators descending from step A must be completed before B or any of its descendants can begin. This kind of ordering can be accomplished by making the completion of step A an enabling precondition for step B using the *waitfor* construct (see example Figure 1). For CPM-GOMS, this approach is too restrictive. Operators descending hierarchically from one step must be free to execute before those from a prior step have completed, subject to the constraints described earlier. The necessary effect is achieved using the *rank* clause. Whereas *waitfor* is a control construct in the programming language sense (like a loop or conditional-branch) and specifies order of execution, *rank* is a declaration (advisory construct) that attributes a property – rank value – to a given goal. If rank is specified for the steps of every method, then any two primitive operators arising from different templates can be compared to see which has the superior rank value and, thus, which has precedence in case of a resource conflict.

```
(procedure
  (index (perform withdraw transaction ?amt))
  (step s1 (fast-move-click withdraw-button) (rank 1))
  (step s2 (fast-move-click checking-button) (rank 2))
  (step s3 (enter-number-sequence ?amt) (rank 3))
  (step s4 (slow-move-click money-slot) (rank 4))
  (step ctl1 (terminate) (waitfor ?s1 ?s2 ?s3 ?s4)))

  (a)

(procedure
  (index (fast-move-click ?target))
  (step c1 (initiate-move-cursor ?target))
  (step hvr1 (hold-resource r-hand-block) (waitfor ?c1))
  (step m1 (move-cursor ?target) (waitfor ?c1))
  (step c2 (attend-target ?target))
  (step hvr2 (hold-resource vision-block) (waitfor ?c2))
  (step c3 (initiate-eye-movement ?target) (waitfor ?c2))
  (step m2 (move-eye ?target) (waitfor ?c3))
  (step p1 (perceive-complex-obj ?target) (waitfor ?m2))
  (step rvr2 (release-resource vision-block) (waitfor ?p1))
  (step c4 (verify-target-pos ?target) (waitfor ?c3 ?p1))
  (step c5 (initiate-click ?target) (waitfor ?c4 ?m1))
  (step m3 (mouse-down ?target) (waitfor ?m1 ?c5))
  (step m4 (mouse-up ?target) (waitfor ?m3))
  (step rvr1 (release-resource r-hand-block) (waitfor ?m4))
  (step ctl1 (terminate) (waitfor ?m4 ?rvr1 ?rvr2)))

  (b)

(procedure
  (index (mouse-up))
  (profile right-hand)
  (step s1 (start-activity right-hand release-mouse-button
    :object mouse-device :duration 100 => ?a)
  (step ctl1 (terminate) (waitfor (completed ?a)))
  (step ctl2 (reset ?self) (waitfor (resumed ?self))))

  (c)
```

Figure 2: (a) method (b) template and (c) operator

¹ This is one part of a broader effort to reduce the time and expertise needed to create Apex models while increasing the size and complexity of models that can realistically be attempted (Freed and Remington, 2000).

Apex can use precedence information to infer control (operator sequencing) decisions that comply with the unary resource and slack exclusion constraints. For example, the method represented in Figure 2a generates goals for moving the mouse to and clicking on the “withdraw” and “checking” buttons. These goals will be decomposed using the template in 2b, creating numerous operator-level goals including (*initiate-move-cursor withdraw-button*) and (*initiate-move-cursor checking button*).

As these both require use of the “cognition” resource, a unary resource constraint applies – the goals cannot be executed at the same time. Neither the method or template representation explicitly orders these steps. Instead, Apex automatically detects the conflict and determines that a precedence relationship exists between the goals, with (*initiate-move-cursor withdraw-button*) having precedence. On this basis, the architecture executes this goal first and thereby satisfies the constraint.

Representing Templates

A CPM-GOMS template specifies a set of operators, each representing a discrete cognitive, perceptual or motor activity, and a set of constraints on the execution of operators. Constructing templates involves consideration of the logical requirements of the task (e.g. moving a mouse to a target location must involve steps for finding the target), general principles of human cognitive behavior (e.g. reading a word requires visual attention (McCann, Folk, & Johnston, 1992)), and, in some cases, template-specific parameters (e.g. Fitt’s Law constants for mouse movement). Template construction is also guided by theory which places additional constraints. For example, CPM-GOMS incorporates the Model Human Processor constraint that motor actions are preceded by cognitive initiate operator to prepare the action (John 1996).

Figure 2b illustrates how templates are represented in Apex. Every cognitive, perceptual and motor activity (operator-level action) is represented in a step clause. Logical constraints are represented using the *waitfor* clause. For example, the cognitive action to initiate motor behavior represented in step *c1* is considered a prerequisite for performing the motor behavior (*m1*). Thus, the latter step is explicitly constrained to wait for the former to complete. Unary resource constraints between operators within a template – i.e. where two operators from the template use the same resource – are also represented using the *waitfor* clause. Note that the rank-based mechanism described previously cannot enforce resource constraints since all operators from a template will have equal precedence.

Slack exclusion constraints are meant to prevent operators with certain “excluded” properties from executing during otherwise available slack intervals. The cognitive initiation exclusion rule described

earlier requires that templates represent constraints that prevent an operator from executing if (a) the operator originates from a different template with lower precedence, (b) the operator represents a cognitive action to initiate behavior in some resource R, and (c) there is at least one operator in the template that uses R and is not yet complete (including any that have not yet begun).

Representations in the template must test whether these conditions hold. For property (a), it is only necessary that an excludable operator produces a resource conflict. Apex will then automatically defer its execution and thereby enforce the exclusion constraint. However, this will work only if operator-level goals that use the contested resource, and thus produce a conflict, also have property (b). This requires extending the concept of a resource to signify any arbitrary property of an operator. In our model, cognitive operators that initiate action in resource R are represented by procedures defined to use a resource named *<R>-block* (e.g. “*r-hand-block*”, “*vision-block*”) ². Property (c) concerns *when* the template will prescribe use of the *<R>-block* resource, making a conflict possible. In particular, the temporal scope of potential conflicts is an interval running from the first cognitive initiation act for resource R in the template to the last operator in the template that uses R. Because this interval is less than the lifetime of the template-level goal, it cannot be specified by a profile clause in the template. Instead, the *hold-resource* and *release-resource* commands are used (see 2b) to precisely bound the interval.

Representing Operators

Operators are the most primitive, fine-grained level of behavior in a task representation. To represent a CPM-GOMS operator in Apex, three issues must be considered: resource requirements, duration and interruption handling. Every operator in a CPM-GOMS model represents a discrete cognitive, perceptual or motor activity. Each is thus a user of a single cognitive, perceptual or motor resource notated in a *profile* clause in the operator’s definition. In some cases, an operator will have a property making it eligible for exclusion under an exclusion constraint. If so, an additional resource (e.g. *vision-block*) will be listed in the operator’s profile clause to represent the property that makes it excludable.

An operator’s duration contributes to the predicted total time required for a high-level goal and also determines whether the operator can fit into a given interval of slack time. In Apex, duration is

² What “block” resources correspond to in psychological terms is still undetermined. One possibility is that each represents limited cognitive capacity to manage behavior in some other resource. Another possibility is that each such resource represents a learned inhibition against interleaving similar behaviors from different goals.

represented as a parameter of a *start-activity* step in the operator representation (see Figure 2c). Executing this step is what causes the operator to occupy a specified resource for a block of (simulated) time. When the prescribed interval has passed, an event of the form (*complete <activity>*) is generated to indicate that the operator is complete.

As described earlier, the unary resource constraint requires that operators not be executed in a given interval of slack time if the size of that interval is less than the operator's duration. Apex meets this constraint, not by preventing the operator from executing in that interval, but by aborting its execution when the window of slack time closes. Specifically, an operator will begin execution as soon as its waitfor preconditions (if any) are satisfied and it becomes the highest priority competitor for the resource(s) it needs. If an operator with conflicting resource requirements and higher priority (i.e. from a template-level goal with higher precedence) becomes enabled while it is executing, Apex interrupts the executing operator and reallocates resources to the new one³. By default, Apex resumes an interrupted goal where it left off once the goal again becomes the highest priority competitor. However, the correct behavior in this case is not to resume but to reset (start over). To override the default, a *reset* step (see Figure 2c) is included in the representation of each operator.

Discussion

We have provided a detailed description of the requirements for automating CPM-GOMS and shown how Apex implements these requirements. The use of Apex to automate CPM-GOMS analyses allows user-interface designers and engineers to simulate human performance on HCI tasks with little effort and minimal expertise in cognitive modeling or psychology. For example, in a recent class at Carnegie Mellon University, all students, none of whom were psychology students, were able to go from simple keystroke-level models they had built in Apex to correct CPM-GOMS models in less than half an hour. In previous classes and CHI tutorials, the process took six times as long and often resulted in incorrect models. These experiences raise hopes that Apex will allow UI designers and engineers to easily evaluate human performance on any given interface using a powerful modeling method previously inaccessible to all but expert cognitive modelers.

³ Enforcing unary resource constraints by interrupting and restarting operators is inefficient for "ballistic" tasks where all operators needed for the high-level goal can be specified at the outset. It is appropriate for "reactive" tasks where new goals and operators may arise in response to unknown or unexpected features of the environment.

References

- Agre, P. and Shragar, J. (1990) Routine evolution as the microgenetic basis of skill acquisition. Proceedings of the 12th Annual Conference of the Cognitive Science Society.
- Card, S. K., Moran, T.P. & Newell, A. (1983). *The Psychology of Human-Computer Interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Firby, R.J. (1989) Adaptive Execution in Complex Dynamic worlds. Ph.D. thesis, Yale University.
- Freed, M. (1998a) Managing multiple tasks in complex, dynamic environments. In *Proceedings of the 1998 National Conference on Artificial Intelligence*. Madison, Wisconsin.
- Freed, M. (1998b) Simulating human performance in complex, dynamic environments. Ph.D. thesis, Northwestern University.
- Freed, M. (2000) Reactive Prioritization. *Proceedings 2nd NASA International Workshop on Planning and Scheduling for Space*. San Francisco, CA.
- Freed, M. and Remington, R. (2000) Making Human-Machine System Simulation a Practical Engineering Tool: An APEX Overview. In *Proceedings of the 2000 International Conference on Cognitive Modeling*. Groningen, Holland.
- Gray, W. D., & Boehm-Davis, D. A. (2000). Milliseconds matter: An introduction to microstrategies and to their use in describing and predicting interactive behavior. *Journal of Experimental Psychology: Applied*, 6(4), 322-335.
- Gray, W. D., John, B. E. & Atwood, M. E. (1993) Project Ernestine: Validating a GOMS Analysis for Predicting and Explaining Real-World Task Performance, *Human-Computer Interaction*, v.8 (3), pp.237-309.
- John, B. E. (1996) TYPIST: A Theory of Performance In Skilled Typing. *Human-Computer Interaction*, 11 (4), pp.321-355.
- John, B. E. & Gray, W. D. GOMS Analyses for Parallel Activities. Tutorial materials, presented at CHI, 1992 (Monterey, California, May 3- May 7, 1992), CHI, 1994 (Boston MA, April 24-28, 1994) and CHI, 1995 (Denver CO, May 7-11, 1995) ACM, New York.
- John, B. E. & Kieras, D. E. (1996). The GOMS family of user interface analysis techniques: Comparison and Contrast, *ACM Transactions on Computer-Human Interaction*, v.3(4), pp. 320-351. New York: ACM Press.
- John, B., Vera, A., Matessa, M., Freed, M. and Remington, R. (2002) Automating CPM-GOMS. In *CHI 2002 Conference Proceedings*.
- Matessa, M., Vera, A., John, B., Remington, R. and Freed, M. (2002) Reusable templates in human performance modeling. In Proceedings of the 24th Annual Meeting of the Cognitive Science Society.
- McCann, R. S., Folk, C. L., & Johnston, J. C. (1992). The role of attention in visual word processing. *Journal of Experimental Psychology: Human Perception and Performance*, 18, 1015-1029.
- Pell, B. Gat, E., Keesing, R., Muscettola, N. and Smith, B. (1997) Robust Periodic Planning and Execution for Autonomous Spacecraft. In *Proceedings of the Fifteenth Joint Conference on Artificial Intelligence*, Nagoya, Japan.
- Zweben, M. and Fox, M. (1994) *Intelligent Scheduling*, Morgan Kaufman.