Generating Automated Predictions of Behavior Strategically Adapted to Specific Performance Objectives

Katherine Eng[1], Richard L. Lewis[2], Irene Tollinger[1], Alina Chu[2], Andrew Howes[4], Alonso Vera[3]

[1]NASA Ames Research Center
Moffett Field, CA
irene.tollinger@nasa.gov

[2]Psychology
University of Michigan
Ann Arbor, MI
rickl@umich.edu

[3]Carnegie Mellon & NASA Ames Research Center
Moffett Field, CA
alonso.vera@nasa.gov

[4] School of Informatics
University of Manchester
Manchester, UK
HowesA@manchester.ac.uk

ABSTRACT
It has been well established in Cognitive Psychology that humans are able to strategically adapt performance, even highly skilled performance, to meet explicit task goals such as being accurate (rather than fast). This paper describes a new capability for generating multiple human performance predictions from a single task specification as a function of different performance objective functions. As a demonstration of this capability, the Cognitive Constraint Modeling approach was used to develop models for several tasks across two interfaces from the aviation domain. Performance objectives are *explicitly* declared as part of the model, and the CORE (Constraint-based Optimal Reasoning Engine) architecture itself formally derives the detailed strategies that are maximally adapted to these objectives. The models are analyzed for emergent strategic variation, comparing those optimized for task time with those optimized for working memory load. The approach has potential application in user interface and procedure design.

Author Keywords
User modeling, interface evaluation.

ACM Classification Keywords
H5.2. Theory and Methods.

INTRODUCTION
Modeling human performance in complex task domains has been approached using a number of methods, including engineering models (e.g. KLM, GOMS, CPM-GOMS [10]) and computational architectures (e.g. ACT-R [2], Soar [19], EPIC [14]). The domains modeled have been diverse, ranging from operating phone workstations [7], to playing video games [12], to piloting aircraft [14]. These models have been quite successful at both predicting and explaining observed behavior. However, a fundamental difficulty in developing such models is that there are multiple behavioral strategies available for people to accomplish their task goals, and the cognitive modeler must account for this strategic variability. In current approaches, this requires extremely detailed task analysis to identify or hypothesize the strategies that people might use. This is followed by careful programming or formal specification of those strategies (for example, in the form of a set of production rules) that are then used in conjunction with the architectural theory to yield predictions (see Kieras and Meyer, 1995 for explicit discussion [14]). Current modeling approaches provide no direct support for this task analysis and strategy specification; they simply admit of the possibility of strategic variation (itself a critically important and nontrivial feature).

We present here features of a new modeling approach [9, 17, 25] fundamentally based on the idea that behavior is shaped not only by basic task goals and the constraints on the cognitive architecture, but that it is also strategically shaped by specific performance objectives. These objectives can include speed, accuracy, minimization of memory load, maximization of perceptual attention/reactivity, or most likely a more complex balance among many performance criteria. What is novel in this approach is that these performance objectives are *explicitly* declared as part of the model, and the modeling engine itself formally derives the detailed strategies that are maximally adapted to these objectives. In particular, we will show how multiple strategies (and therefore behavioral predictions) may be derived from a single task specification, by varying the performance objective function. Tollinger, et al. reported work on automatically generating predictions at multiple levels of skill based on a single hierarchical task description [24]. The current work adds another dimension to the space of possible models that can be generated via this approach.

The rest of the paper is organized into the following sections: a brief review of the ubiquity of strategic adaptation across behavioral time scales; an identification of the applied potential for a modeling tool such as the one we have prototyped; a description of the technical methods and the underlying architecture used to automatically generate models; an illustration of the modeling technique using examples drawn from aircraft pilot tasks, in which we generate models that minimize both time and working memory load; and a conclusion summarizing what makes the approach distinctive, and pointing to future research directions.

THE UBIQUITY OF STRATEGIC ADAPTATION SHAPED BY PERFORMANCE OBJECTIVES

It is a commonplace of psychology that human behavior is adaptive, but what is perhaps a surprising result of research over the last 10-15 years is the extent to which strategic adaptation and variability manifests itself across behavioral timescales—even at the lowest levels of extremely rapid and routine behavior. We briefly review some of this evidence here. The clear import of this work for modeling is that there is no domain of interactive behavior or class of tasks that is immune to these strategic adaptations.

Recent work on elementary dual-task situations has demonstrated the effect of strategic variations at the level of primitive cognitive, perceptual, and motor operations [15]. For example, Schumacher, et al. [21] showed slowed task responses on the order of milliseconds on a basic choice reaction task in order to accommodate specific instructions about relative task priorities in a dual-task situation—in effect a precise modulation of low-level behavior in order to achieve a particular desired speed-accuracy tradeoff. In a similar spirit, Gray and Boehm-Davis observed variations in time on task caused by slight differences in user interfaces and proposed microstrategic variations as an explanation for the observed behavior. The behavioral adaptation demonstrates that users will vary low-level operations to optimize speed; the difference between the two posited strategies is only 150 milliseconds [6].

Strategic adaptations show up in complex applied domains as well. Wickens and Seidler [27] found that when aircraft pilots were told to prioritize a menu-driven information access task over an altitude monitoring task, they shifted to view the relevant target screens through two separate displays instead of one shared display. Because they were no longer sharing one display with the two screens displayed sequentially, this effectively reduced the strain on working memory. This work further supports the finding that people can and do optimize strategies to satisfy specific task priorities.

In fact, quite fine-grained strategic adaptations are sometimes explicitly taught as part of interface training. For example, when pilots have to input a radio frequency, they are taught to begin dialing without looking at the knob, which is located out of view beneath the pilot's seat. Only towards the end of the dial rotation are they instructed to look down to fine-tune the input. This strategic procedure maximizes the amount of time pilots are looking ahead, a primary objective in aircraft piloting (Michael Feary, personal communication).

APPLICATION AREAS: INTERFACE AND PROCEDURE DESIGN

There is a large and varied set of objectives that are relevant to real-world design areas. For example, aircraft pilots may not only be concerned with speed of procedure execution but also with maintaining a particular visual scan pattern and making as few errors as possible. Understanding the different strategies that can emerge from these objectives can provide insight into a number of applied domains.

For example, in the Wickens and Seidler study above, a number of pilots were unable to discover and use the optimal strategy of using two displays. The authors attributed this to the interface's failure to successfully represent the navigational and organizational structure of the user interface. Therefore, pilots who were unable to correctly perceive the system model were unable to discover and adopt the most effective strategy. This indicated a need to redesign the interface to better afford the discovery of optimal strategies. Because interfaces designers do not always have empirical data to guide their designs, *a priori* models of optimal behavior offer an approximation of this behavior to focus their efforts on key design elements.

Another application to interface design is the exploration of "what-if" scenarios and the quantitative comparison between alternative designs. Predictive modeling would provide an estimation of the relative costs between interfaces under specific performance objectives and would support designers in making more informed design tradeoffs.

However, redesigning an existing interface based on *a priori* predictions of human performance is not always a feasible option. For example, in the aviation domain any interface change requires a costly cockpit redesign. Models of optimal behavior could provide a alternative solution by providing an optimized procedure. For example, if pilots can be taught to look for pieces of information at more effective times, this will improve performance without the cost of interface redesign.

MODELING PERFORMANCE OBJECTIVE FUNCTIONS AND THEIR IMPLICATIONS

Our modeling approach, Cognitive Constraint Modeling (CCM), is based on the idea that skilled behavior can be understood as the solution to a constraint satisfaction problem defined by the conjunction of task, environmental, and cognitive/perceptual/motor constraints. Put another way, task and architecture jointly circumscribe the bounds on skilled adaptation, and strategic variation must exist within this bounded space. The approach is therefore a kind of *rational analysis*, but one that departs sharply from classical rational analysis in that it takes into account not only the structure of the task environment but also the structure of the human performance system [1].

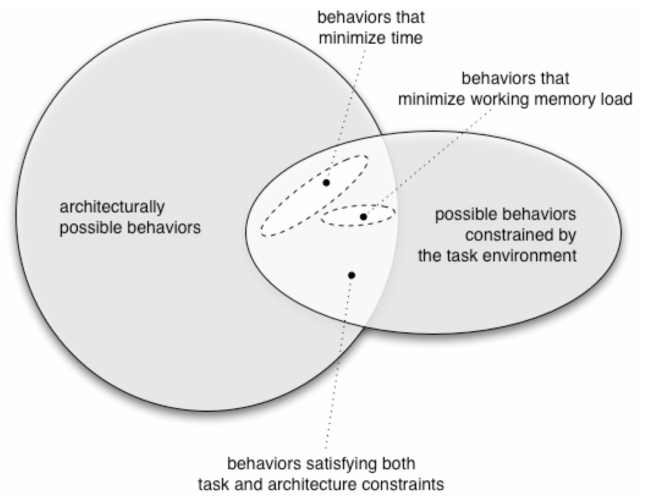

Figure 1. The possible space of strategic behaviors.

Within this bounded space of possible behaviors, further relevant subspaces or surfaces may be identified by selecting out those behaviors that maximize explicitly defined performance objectives. This situation is shown abstractly in Figure 1, which depicts a large space of architecturally possible behaviors intersecting with a smaller space of task-constrained behaviors. Within this intersection lie further subspaces that correspond to behaviors optimal with respect to different performance objectives. The figure shows subspaces corresponding to behaviors that maximize speed (i.e. minimize total time), and those behaviors that minimize working memory load, the two exemplar objective functions we explore below. The figure shows these two subspaces as disjoint, indicating that there is an *unavoidable* tradeoff in speed and working memory; this need not be the case however, and our analytic technique can be used to reveal whether such tradeoffs necessarily exist, or whether there is overlap in the optimal subspaces.

To achieve these analytic goals, some way is needed to explicitly represent the task constraints, the architecture constraints, and the objective functions. Additionally, it is necessary to formally reason with these constraints to automatically derive instances of the behaviors in the subspaces of interest identified by the objective functions. The modeling technology that we use to achieve this is CORE (Constraint-based Optimal Reasoning Engine) [9, 25, 17]. We now provide an overview of how the constraints and objective function are represented and used; for further details see the cited papers. The CORE architecture is also available on request from HowesA@manchester.ac.uk though it requires the purchase of a license for the SICSTUS implementation of the Prolog programming language.

Task Constraints

Howes, Lewis, Vera, and Richardson describe an Information Requirements Grammar (IRG), a notation for representing the knowledge required to execute tasks [8]. A key feature of IRG as a task specification is that it focuses on representing the *information requirements* of tasks, subtasks, and primitive operations, rather than a single sequence of steps. For example, although there is a hierarchical breakdown of tasks into subtasks, the order of the subtasks in IRG notation is irrelevant; if there are no explicit dependencies among subtasks, they may execute in parallel—subject, of course, to architectural constraints. Thus, IRG is a natural notation for denoting *spaces* of behaviors, because it is possible to create task specifications that are underspecified with respect to the details of control flow.

The important aspects of IRG are most easily seen by example. Figure 2 is a high-level task specification in IRG that represents a piloting task. In this task, the pilot hears a description of the current situation and is instructed to climb to the specified altitude limit at the best rate of climb. The pilot must check the airspeed window to obtain the current speed and then check the airspeed mode to verify the current velocity mode. Based on the instructions and current state of the aircraft, the pilot sets the appropriate speed. Finally, the pilot verifies the changes by checking the airspeed window and airspeed tape.

In the IRG representation of the task, bold words represent function names, and other words are parameters. Uppercase parameters after a colon sign are variables bound to the identifier of the process. An IRG rule that refers to the identifier of another process requires that information before it can be executed. For example, the pilot cannot **set speed given** until he or she knows what the INSTRUCTION, CURRENT_SPEED, and MODE are. These information flow constraints, along with the resource constraints imposed by the lowest-level processes (described more fully below) constrain the possible behaviors in task execution. IRG task descriptions decompose into a set of sub-tasks, and eventually into the level of architectural primitives described next.

Architectural Constraints

The specification of the architecture consists of two main parts. The first part consists of a declaration of the available architectural resources (processors in the sense of the Model Human Processor [4] and EPIC [14], or processors and buffers in the sense of ACT-R [2]), their connectivity, and their temporal operating characteristics. The details of this specification are relatively straightforward but beyond the scope of the present article; see [9] for more details.

The second part consists of the identification of those partial pathways through the network of resources that represent the smallest composable units of processing. Put another way, we seek to represent the set of process cascades that form the invariant functional primitives of discrete combinatorial adaptation. We call these architecturally bound information process cascades *Architectural Process Cascades* (APCs) [26]. For example, the *motor-simple* APC includes a cognitive initiate, a buffer, and a motor operator representing a mouse-up, mouse-down, press key or other action that uses a parametric estimate.

```
pilot_task
  →
  comprehend situation yielding flightplan and last_clearance       : FLIGHTPLAN  LAST_CLEARANCE,
  comprehend clearance yielding instruction and altitude_limit        : INSTRUCTION LIMIT,
  check airspeed_window after INSTRUCTION LIMIT                       : CURRENT_SPEED,
  check airspeed_mode after INSTRUCTION LIMIT                         : MODE,
  set speed given INSTRUCTION CURRENT_SPEED MODE                      : SPEED_SET,
  check airspeed_window after SPEED_SET                               : WINDOW_CHECKED,
  check airspeed_tape after SPEED_SET    : TAPE_CHECKED.

vision_simple_APC TARGET after EVENTS :  SEEN_TARGET
  →
  attend visual after EVENTS                                          - ATTENDED,
  transmit ATTENDED in visual_buffer                                  - TRANSMIT,
  do fixate on TRANSMIT                                               - FIXATION,
  perceive visual TARGET on FIXATION TRANSMIT                         - PERCEIVED,
  hold PERCEIVED in visual_buffer                                     - SEEN_TARGET.
```

Figure 2. An IRG representation of a piloting task and a vision simple APC.

The reason these three units constitute an APC is because motor operators *must* be preceded by cognitive operators that initiate them, and *init*s by definition instantiate another process. Furthermore, information between processes must be mediated by resource-consuming buffers. These are architectural requirements; the three operators are architecturally bound together.

IRG serves as a natural representation for APCs as well. Figure 2 shows an APC for simple visual perception represented in IRG. All of the results of the rules in an APC which follow hyphen signs are bound to resource-consuming processes which represent primitive cognitive, perceptual and motor processes. APCs also specify buffers in which pieces of information are passed. For example, in the APC above, once the TARGET is PERCEIVED, that information is held in the visual_buffer until it is required by another APC or until the buffer process terminates.

It is important to emphasize again that although the task description and APCs create a complete specification of task resources and information constraints, the result is an under-constrained model of the task. For example, consider the last two sub-tasks in the piloting task in Figure 2. This task requires the model to check the airspeed_window and airspeed_tape after the speed has been set. While the specification requires SPEED_SET to have been completed before these checks can occur, it does not define the temporal order between the two; the check of the airspeed_window can happen before the check of the airspeed_tape, or vice versa. As a result, the task constraints do not produce a definitive model of the task, but rather a space of possible models.

Objective Functions

The objective function must be specified in terms of concrete features of the resulting behavior in a quantitative fashion. For example, to minimize time on task, the objective function minimizes the end time of the last process. The objective function is not limited to general architectural specifications but can also be task-specific, such as amount of time spent visually scanning in the cockpit.

In addition to minimizing time, the other objective function that we explore here is minimizing working memory load. There is an extensive body of literature supporting the effect of working memory capacity on errors, demonstrating that behavior on tasks of even modest complexity is shaped and constrained by working memory limitations [e.g. 3, 5, 13].

In our present model explorations, we operationalize working memory load with an extremely simple metric: the grand total of all working memory residence times of each item of information in the task. While the existing work on working memory includes many theoretical models, some implemented computational models, and myriad relevant behavioral phenomena (see Miyake and Shah [18] for a recent overview), for the moment we are not implementing any one theory. Rather, our representation and quantification of working memory should be viewed as an abstraction, and more importantly, a vehicle for relative comparison between models. There are a number of methods by which working memory could be quantified, such as the total duration of working memory use, the number of concurrent items held in working memory, the median amount of time items are held in working memory, modulated by similarity of representations, etc. We are not proposing that any of these metrics is more correct than the others, but we do hope to demonstrate a framework in which one can readily apply such metrics to the prediction of the interplay between working memory load and skilled performance. For the purpose of generating the models described here, we chose total duration of working memory use as the metric.

Automatically Searching the Strategy Space with CORE

CORE provides the mechanisms through which the constraints specified in IRG can be satisfied to generate an optimal schedule. To derive an optimal schedule, the system requires the IRG task (e.g. the piloting task) and human cognitive architecture specifications (e.g. the vision APC). Based on these constraints, the system performs a search to find the optimal behavior given a modeler-defined objective function. The underlying technology is an off-the-shelf finite-domain constraint-satisfaction package in Prolog.

After searching the strategy space, CORE outputs a single model with cognitive, perceptual and motor processes scheduled in the form of a *behavior graph*, a kind of Gantt chart (see Figures 3, 4, and 5). In order to model behavior with a different objective (e.g., performance speed or working memory utilization), the modeler must program which features in the model CORE will use to compute the objective costs. For example, to create an objective function which minimizes the amount of time items are held in working memory buffers, we define the cost as the sum of the durations of all working memory buffer processes in the model. Over time, a standard set of objective functions will emerge. This approach provides a significant benefit to the modeler: the ability to generate any number of models with varying objectives from a single task description.

EXAMPLE: MAXIMIZING SPEED AND MINIMIZING MEMORY LOAD IN COCKPIT TASKS

The domain of aircraft piloting offers particular interest because of the complex interface and high-workload tasks inherent to the cockpit environment. Reciprocally, such domains stand to benefit the most from *a priori* predictions of human performance as training and evaluation of new interfaces can be very costly and time-consuming [7].

Boeing has designed a new interface for the 777 cockpit called the Flight Deck of the Future (FDF). One of the primary goals of the redesign was to provide pilots more feedback on the auto-pilot settings (lateral navigation mode and vertical navigation mode). A related and explicit design goal was not to increase time on task in the new interface [Michael Feary, personal communication].

Boeing provided the authors with a set of tasks being used to evaluate the new interface. We verified the descriptions of the correct pilot actions for two of the tasks with a pilot to construct accurate task specifications in IRG. Each of the task specifications was modeled three times to simulate various objective conditions: a baseline, non-optimized condition and two optimized conditions for time and total working memory residence time.

In addition to varying the objective, these tasks were modeled on the two different interfaces (777 and FDF) to produce a total of 12 models. This setup provided the ability to compare both across the objective conditions as well as across different interfaces (a 3 x 2 comparison). The models are available upon request from the authors.

Strategic Variations

The optimized schedule outputs from CORE predicted a number of strategic variations among the non-optimized and the two optimal conditions. In the task modeled below, the pilot is instructed to change the plane's heading to 310 degrees. The pilot must check the current mode, dial the heading selector, and press the heading select button. In particular, we are interested in the scheduling of the check to the mode display, which is completed through a visual APC (a look to the mode display and a perception of its content). This look is constrained such that it must happen sometime after the instructions for the task are given (at the very start of the figures) but before the heading select button is pressed (at the very end of the figures.

Figures 3, 4, and 5 illustrate differences between the three optimization conditions for this task. Figure 3 is non-

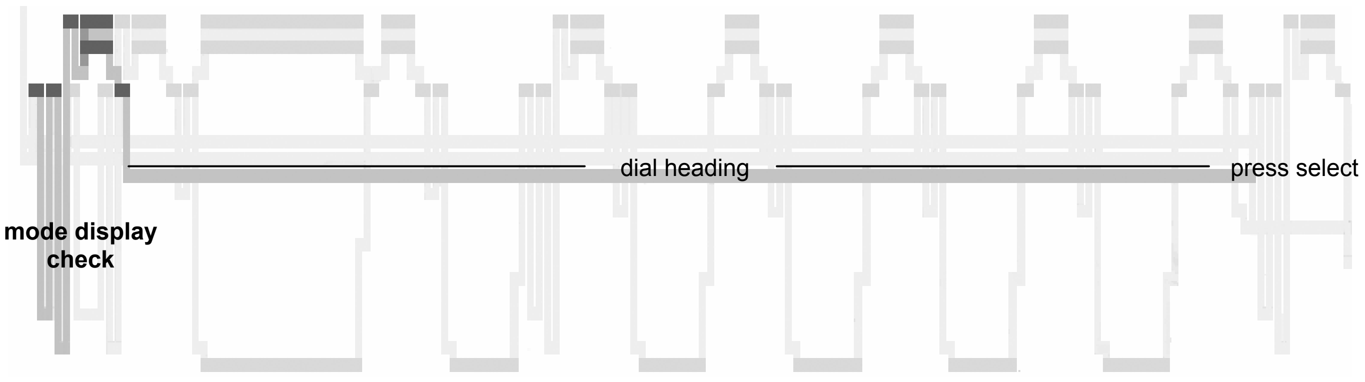

Figure 3. Non-optimized schedule of the piloting task. The mode display check has been scheduled on the critical path.

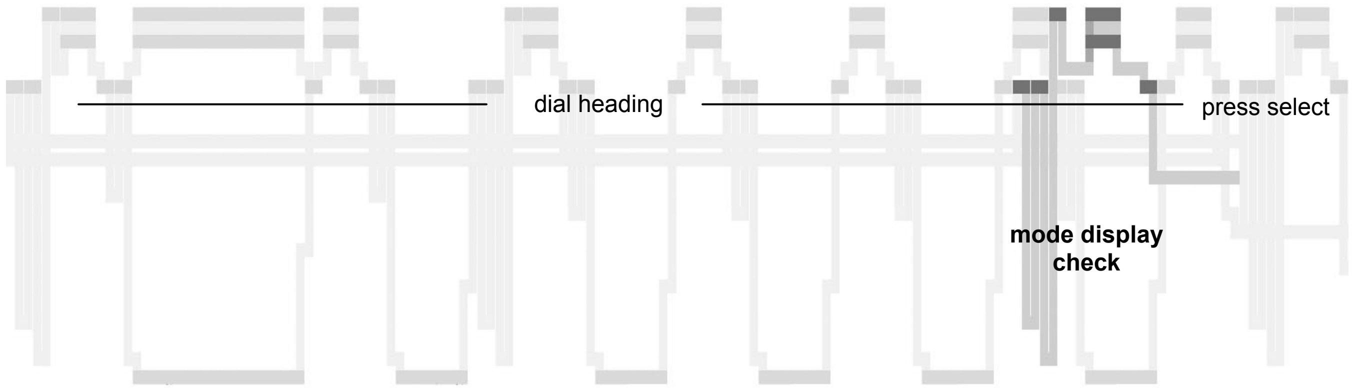

Figure 4. Optimized schedule for total working memory residence time. The mode display has been interleaved with the dialing towards the end of the schedule, resulting in the reduced horizontal working memory buffer trailing the end of the check.

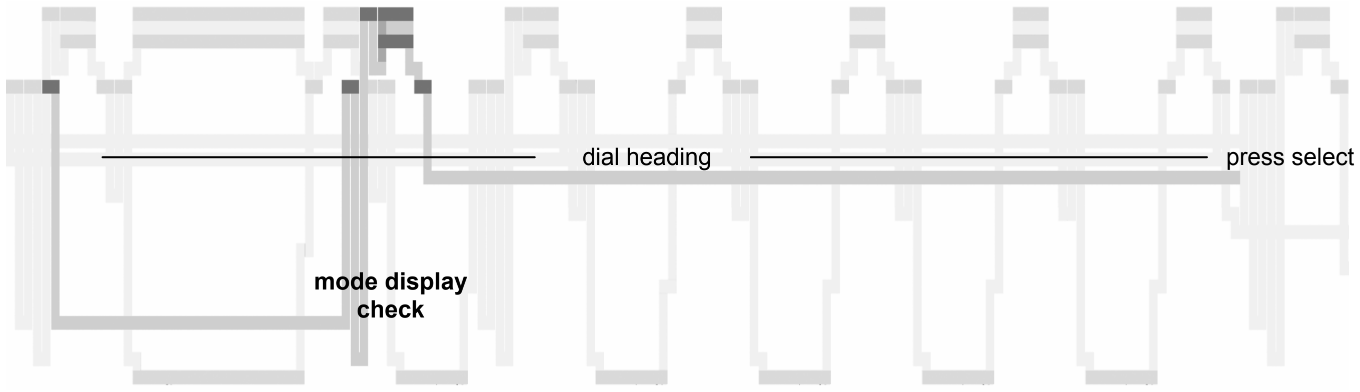

Figure 5. Optimized schedule for time. The mode display has been interleaved with the dialing towards the beginning of the schedule, resulting in the long horizontal working memory buffer trailing the end of the check for the majority of the task.

optimized, Figure 4 has been optimized for total working memory residence time, and Figure 5 has been optimized for time. The look behavior in question is the group of boxes in dark gray shading and labeled *mode display check.*

In the non-optimized model (Figure 3), CORE has selected a schedule that places the visual check at the earliest possible time and on the critical path. In other words, the check of the mode has not been interleaved such that it happens in slack time is part of the chain of dependent operators with the longest duration. However for both the working memory and time optimizations of the task (Figures 4 and 5), CORE has discovered a schedule where the look has been interleaved with other processes (i.e., the look is happening concurrently with the dialing of the heading). This interleaving was not explicitly programmed in the task specification; the parallelism emerged dynamically to optimally satisfy each of the different objective conditions.

Furthermore, it is important to note the exact placement of the interleaving in the context of the task. In the non-optimized and the time optimized models, the look occurs relatively early in the schedule, and the display value is held in working memory for the majority of the task. However in the working memory optimization, the look happens just-in-time for the display value information to be passed to the button push, towards the end of the figure. The just-in-time placement of the look under the working memory optimization required no effort on the modeler's part; this optimal arrangement was automatically found by CORE's systematic search of the strategy space.

Figure 6 shows working memory load (measured as the total working memory residence time) across the two piloting tasks on the two interfaces. The graph shows that CORE consistently finds schedules with lower working memory load by optimizing the objective function. Overall, the models with the working memory optimization objective reduced working memory buffer duration by an average 2688 millisecond difference (2.7 seconds) compared to the time optimized objective and by 3288 milliseconds (3.3 seconds) compared to the non-optimized models.

Interface Comparisons: 777 and FDF

Table 1 summarizes the time and working memory costs across the tasks for each interface and optimization condition. The FDF performed better than the 777 in the non-optimized and both the optimized time and working memory conditions. Across both tasks, the FDF consistently supported a strategy that allowed for a lower working memory load compared to the best case working memory load in the 777 (in one task by 175 milliseconds and in another by 1375 milliseconds). The FDF also performed better on time on task than the 777 (in one task by 100 milliseconds and in another by 500 milliseconds).

These results validate the explicit design objectives behind the FDF interface. The new interface comes at no cost to the time required to complete tasks while enabling a better distribution of working memory load.

While any one of the four task conditions resulted in a schedule with the same time under either optimization, this was not a foregone conclusion—it is quite possible that a tradeoff could have emerged, and we have seen this happen in other models. For this particular combination of tasks and interfaces, it just so happens that it is possible to achieve maximal speed and working memory minimization (at least as defined by our simple metric) simultaneously.

CONCLUSION

This paper has illustrated how a new approach and tool for cognitive modeling systematically supports the automatic generation of a variety of behavioral strategies from a single task specification, as a function of different explicitly defined performance objectives (in the present case, minimize time and minimize working memory load). This approach and tool thus begins to address one of the fundamental challenges that has emerged in constructing detailed cognitive models: dealing with the fact that human behavior is under-constrained by task constraints and cognitive architecture, and people adopt different behavioral strategies in accord with their specific performance objectives.

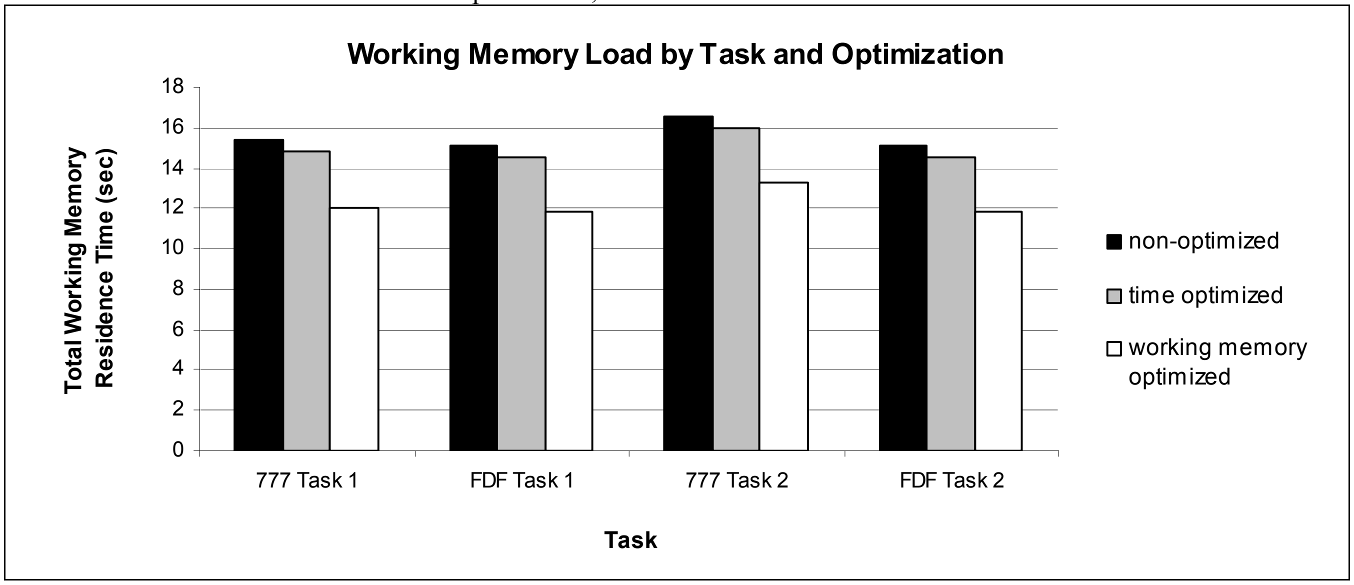

Figure 6. Graph of the total working memory buffer length (milliseconds) by task and for each user interface.

		Non-Optimized		**Optimized by Time**		**Optimized by Working Memory**	
	Cost	*time*	*working memory*	*time*	*working memory*	*time*	*working memory*
Task 1	777	5450 ms	15400 ms	5300 ms	14800 ms	5300 ms	12050 ms
	FDF	5350 ms	15100 ms	5200 ms	14500 ms	5200 ms	11875 ms
Task 2	777	5850 ms	16600 ms	5700 ms	16000 ms	5700 ms	13250 ms
	FDF	5350 ms	15100 ms	5200 ms	14500 ms	5200 ms	11875 ms

Table 1. The time and working memory cost across the non-optimized, optimized time, and optimized working memory models.

Distinctive Features of the Approach

It is worth reviewing a couple of the key differences and similarities between this approach and related modeling approaches available in HCI: programmable computational cognitive architectures such as EPIC and ACT-R, and CPM-GOMS. It should be clear that there are major points of commonality in all the approaches. All embrace the basic theoretical distinction between a fixed information processing architecture (including cognitive, perceptual, and motor components), and a variable program of behavior that can execute on the architecture. However, in current approaches, it is not yet possible to develop a specification of the task constraints that is separate from a detailed specification of the precise strategy that will be used to achieve the task. In particular, to explore the strategic space using present cognitive architectures such as ACT-R or EPIC, the modeler must develop hypotheses about the possible strategies users may employ and then develop production rule code to represent each strategy. Meyer and Kieras, [14] have done this for multiple task situations in the EPIC architecture. The space of possible strategic adaptations is scripted narrowly rather than searched broadly.

In contrast, the approach presented here achieves at least a partial separation of task specification and detailed strategy specification, as the examples above made clear; it was possible to generate multiple strategies from a single task specification by simply varying the objective function. But, it is important to note that the notion of objective functions is already implicit in current modeling work. It is performance objectives such as "go as fast as possible" and "minimize errors" that provide the rational motivation for the posited detailed strategies that humans use in their tasks. What CCM/CORE provides is a way to make this connection between strategy and objective a formal, deductive one. Another way to view this is that we are transferring, but simultaneously significantly reducing, the theoretical degrees of freedom from the large space of possible programs/strategies, to a much simpler specification of an objective function.

An alternative approach to reducing degrees of freedom due to strategic variation is to develop learning models, in particular, models that *learn from instruction* [15, 18, 21, 23]. Developing such models is a useful aim for cognitive science as well, because they serve to further close the loop on explanations of behavior. However, these efforts are complementary to the approach described here. What CCM/CORE provides is a way to analytically specify and explore the *bounds* on skilled adaptation, in a manner that abstracts away from the mechanisms of adaptation.

We believe this approach complements research on learning and instruction in two ways. First, it is important to understand the nature of the space that the learning mechanisms are moving through. Second, to the extent that observed behavior is consistent with the asymptotic bounds analytically derived from task, architecture, and posited performance objectives, then that behavior can be explained without reference to the learning mechanisms. In the learning and instruction-taking approach, the architectural learning mechanisms, the performance architecture, and the instruction taking theory are all jointly under test.

Future Work: Toward a Usable Applied Tool

We have presented a theoretical modeling result and a clear next step is to embark on a program of empirical test and validation, using explicit experimental manipulations of payoff functions to vary performance objectives. But there are a number of other steps that must be taken to improve the modeling capability itself, and we discuss these here.

Using the capability in an applied setting would still be difficult in a number of ways: (1) real world tasks are heterogeneous with respect to objective, (2) producing many more models per task adds overhead to identifying and organizing those models and their outputs, (3) comparing models is currently time and work load intensive, and (4) the novel capability must be integrated into a comprehensive model development system that addresses the broader difficulties associated with computational cognitive modeling. We cover each of these issues below.

(1) Real world tasks are often heterogeneous with respect to objective. For example, an aviation task may include an altitude entry subtask followed by an airspeed entry subtask. Altitude clearances are given to a precision of 100 ft (one click on the altitude dial). If a pilot is 300 ft from his designated altitude he can be penalized. Airspeed clearances are given to a precision of 10 knots (10 clicks on the airspeed dial). Therefore the objective for the altitude subtask would be accuracy while the objective for the airspeed subtask might emphasize speed more. The current implementation supports specifying an objective at the task description level. Functionality should be provided to assign different objectives at the subtask level and model output should identify the objective where it differs for subtasks. This also covers the case where there are varying objectives in dual-task situations. There will be varying priority between the two tasks as well as within a single task.

(2) The automatic generation of multiple models per task description requires the modeler to manage many more models than was necessary previously. In the work presented here, the generation of two models per task (time optimized and working memory optimized) requires the modeler to organize 12 models instead of 6 models. Future work will likely include additional types of objectives (e.g. maximization of visual perceptual attention, minimization of motor movement, etc.). For each type of objective there are likely to be multiple ways to measure a given objective (working memory total buffer length versus number of concurrent working memory buffers). Thus, the multiplier for number of outputs per task will increase. This will require software support to organize the models including: tagging each model (by task, objective function, measurement type) and grouping sets of related models by various criteria.

(3) The comparison of emergent strategies for different objectives is difficult. The current method is visual comparison of two schedules in PERT chart form, which is time consuming and highly dependent on working memory. This is compounded by the quantity of models that result from a large multiplier representing objective type as described above. In order to efficiently and accurately understand the strategic differences between objectives for a given task, computational support will be required. For example, algorithms must be developed to compare schedules such that the user is only alerted when the difference is above a certain threshold and then flag the appropriate area on both schedules.

(4) Even if these issues specific to objective setting are addressed, there are additional difficulties that have historically limited the use of cognitive modeling in applied contexts. The current work would have to fit into a larger tool suite that addresses the broader challenges of applied modeling. These range from providing a non-code based representation of task descriptions to a efficient method of specifying the user interface the model interacts with as described in Tollinger et. al. [24]. Several systems are under development to support applied modeling including: CogTool [11], X-PRT [24], and User Modeling Design Tool [20].

ACKNOWLEDGMENTS

We would like to thank Michael Feary, an experienced pilot, for his contribution to our understanding of these tasks. This research was supported by the NASA Space Human Factor Engineering and Intelligent Systems Programs and by ONR Grants N000140310087 and N0001404IP2002.

REFERENCES

1. Anderson, J.R. *The Adaptive Character of Thought*. Lawrence Erlbaum Associates, Hillsdale, NJ, USA (1990).
2. Anderson, J.R., Bothell, D., Byrne, M.D., Douglass, S., Lebiere, C., & Qin, Y . An integrated theory of the mind. *Psychological Review 111,* 4 (2004), 1036-1060.
3. Byrne, M.D. and Bovair, S. A working memory model of a common procedural error. *Cognitive Science 21*, 1 (1997), 31-61.
4. Card, S. K., Moran, T.P. and Newell, A. *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates, Hillsdale, NJ, USA (1983).
5. Engle, R.W. Working memory capacity as executive attention. *Current Directions in Psychological Science 11,* 1 (2002), 19 - 23.
6. Gray, W.D. and 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.*
7. Gray, W.D., John, B.E., and Atwood, M.E. Project Ernestine: Validating GOMS for predicting and explaining real-world task performance. *Human Computer Interaction 8*, 3 (1993), 237-309.
8. Howes, A., Lewis, R.L, Vera, A.H., and Richardson, J. Information-Requirements Grammar: A theory of the structure of competence for interaction. In *Proc. Of CogSci 2005*, Lawrence Erlbaum Associates (2005).
9. Howes, A., Vera, A., Lewis, R.L., and McCurdy, M. Cognitive constraint modeling: A formal approach to reasoning about behavior. In *Proc. of CogSci 2004*, Lawrence Erlbaum Associates (2004).
10. John, B.E. and Kieras, D.E. (1996) The GOMS family of user interface analysis techniques: Comparison and Contrast. ACM Transactions on Computer-Human Interaction., 3 (4), pp. 320-351.
11. John, B.E., Prevas, K., Salvucci, D.D., and Koedinger, K. Predictive Human Performance Modeling Made Easy. In *Proc. of CHI 2004,* ACM Press (2004).
12. John, B.E., Vera, A.H., and Newell, A. Toward real-time GOMS: A model of expert behavior in a highly interactive task. *Behavior and Information Technology 13*, 4 (1994).
13. Kane, M.J. and Engle, R.W. Working-memory capacity and the control of attention: The contributions of goal neglect, response competition, and task set to Stroop interference. *Journal of Experimental Psychology 132*, 1 (2003), 47–70.
14. Kieras, D.E. and Meyer, D.E. Predicting human performance on dual-task tracking and decision making with computational models using the EPIC architecture. *International Symposium on Command and Control Research and Technology*. National Defense University, Washington D.C. (1995).
15. Kieras, D.E. and Meyer, D.E. A computational theory of executive cognitive processes and multiple-task performance. *Psychological Review 104*, 1 (1997), 3-65.
16. Lewis, R.L., Newell, A., and Polk, T. Toward a Soar theory of taking instructions for immediate reasoning tasks. In *Proc. of CogSci 1989*, 514-521.
17. Lewis, R.L., Vera, A.H., and Howes, A. A constraint-based approach to understanding the composition of skill. In *Proc. of ICCM 2004*.
18. *Models of Working Memory: Mechanisms of Active Maintenance and Executive Control*. Akira Miyake and Priti Shah (eds). Cambridge University Press, New York, NY, USA, 1999.
19. Newell, A. (1990). *Unified Theories of Cognition.* Harvard University Press. Cambridge, Massachusetts
20. Ritter, F.E., Van Rooy, D., and St. Amant, R.A. User modeling design tool based on a cognitive architecture for comparing interfaces. In *Proc. of the International*

Conference on Computer-Aided Design of User Interfaces 2002, 111-118.

21. Schumacher, E.H. Seymour, T.L., Glass, J.M., Fencsik, D.E., Lauber, E.J., Kieras, D.E., and Meyer, D.E. Virtually perfect time sharing in dual-task performance: Uncorking the central cognitive bottleneck. *Psychological Science 12*, 2 (2001), 101-108.
22. Simon, H.A., & Hayes, J.R. (1976). Understanding complex task instructions. In D. Klahr (Ed.), Cognition and instruction (Chap. 14). Hillsdale, NJ: Erlbaum.
23. Taatgen, N.A. Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science* (2005), 421-455.
24. Tollinger, I., Lewis, R., McCurdy, M., Tollinger, P. Vera, A., Howes, A., Pelton, L.. Supporting efficient development of cognitive models at multiple skill levels: Exploring recent advances in constraint-based modeling. In *Proc. of CHI 2005*, ACM Press (2005).
25. Vera, A.H., Howes, A., McCurdy, M., and Lewis, R.L. A constraint satisfaction approach to predicting skilled interactive cognition. In *Proc. of CHI 2004,* ACM Press (2004).
26. Vera, A.H., Tollinger, I., Eng, K., Lewis, R., and Howes, A. Architectural building blocks as the locus of adaptive behavior. In *Proc. Of CogSci 2005*, Lawrence Erlbaum Associates (2005).
27. Wickens, C. D., Seidler, K. S. Information access in a dual-task context: testing a model of optimal strategy selection. *Jounal of Experimental Psychology: Applied 3,* 3 (1997), 196-215.