A Constraint Satisfaction Approach to Predicting Skilled Interactive Cognition

Alonso Vera[1], Andrew Howes[2], Michael McCurdy,[1] Richard L. Lewis[3]

[1]NASA Ames Research Center
Moffett Field, CA
avera@arc.nasa.gov

[2]School of Psychology
Cardiff University
Cardiff, UK
HowesA@Cardiff.ac.uk

[3]Psychology
University of Michigan
Ann Arbour
rickl@umich.edu

ABSTRACT

In this paper we report a new approach to generating predictions about skilled interactive cognition. The approach, which we call Cognitive Constraint Modeling, takes as input a description of the constraints on a task environment, on user strategies, and on the human cognitive architecture and generates as output a prediction of the time course of interaction. In the Cognitive Constraint Models that we have built this is achieved by encoding the assumptions inherent in CPM-GOMS as a set of constraints and reasoning about them using finite domain constraint satisfaction.

Author Keywords
user modeling, tools for usability evaluation.

ACM Classification Keywords
constraint satisfaction, formal user-modeling.

INTRODUCTION

It is extremely expensive, if possible at all, to get assessments of skilled interaction by experts on routine tasks. Consider as an example, NASA's needs in designing a new interface for the shuttle cockpit. Ideally, it would be possible to bring in experienced astronauts, train them for tens or hundreds of hours, and then assess the impact of the new interface on performance time, error potential, and so on. Researchers would then iterate on the interface design, bring in a new set of astronauts, and go through the process again. This cycle would be repeated several times in order to ensure that a substantive proportion of the consequences of the interface on performance are understood and addressed. This is practically impossible due to constraints on time and other resources. Researchers working on shuttle cockpit design struggle to get pilots (not astronauts) as test participants, and then just for a few hours. The return-on-investment for tools that predict the time course and potential for errors in skilled performance is therefore high.

Assessing novice behavior, in contrast, has much lower costs. Novices are easily available and, moreover, relatively little of their time is required as they do not need training. Designing walk-up-and-use systems that require close attention to learnability rather than skilled interaction therefore poses a much lower barrier to effectively applying the design process. The consequences of poor HCI decisions are, at least, no less critical for interfaces that support skilled performance than for those that support walk-up and use tasks. It can be argued therefore, that model-based interface evaluation has great scope for impacting the interfaces of devices intended for skilled users.

Fortunately there is a long tradition of work in Human-Computer Interaction that is aimed directly at improving the efficiency with which predictions of skilled performance can be generated [1,2,3,4,5,6,7,8,9,10,14]. Empirically, there is little question that the performance of routine tasks by skilled individuals involves the interleaving or promotion of operators such that they are executed as the opportunity arises. Predicting this complex, anticipatory behavior has historically been accomplished in one of two ways: By simulating the learning process (e.g., ACT-R) or by hand-crafting behavioral templates that ensure the appropriate sequencing (e.g., Apex-CPM). We will argue that it is both possible and beneficial (e.g., faster, easier) to generate a similar schedule from a set of architectural, strategic, and task constraints.

Parallel recent efforts in the field have begun to focus on trying to make model-based evaluation techniques easier to use by non-experts. In addition to the work reported here, these includes Apex-CPM [7], Glean [10], and ACT-Simple [14]. Apex-CPM is an implementation of CPM-GOMS-like assumptions in a run-time scheduling architecture. CPM-GOMS operators are described in terms of a hierarchy of procedures in which partial-orderings are established with explicit pair-wise relationships between

operators ACT-Simple [14] has been built to transform simple task specifications to non-learning ACT-R models. ACT-Simple generates Key-Stroke-Level models (KLM) of performance which, as Card, Moran, and Newell [8] intended, give a rough approximation of performance early in practice. Similarly, a current effort to integrate EPIC and GLEAN also aims at predicting behavior *early* in practice, in contrast to our goal of predicting skilled behavior (Kieras, personal communication).

The approach to model-based prediction of skilled performance that is reported in the current paper owes most of its formulation to CPM-GOMS. Two key assumptions are shared. The first is that human psychological capability can be described in terms of temporal and resource properties of a distributed set of processors each with its own processing capabilities. The influence on CPM-GOMS derives directly from the Model Human Processor [8]. Each processor is defined with a set of parameters that capture information about, for example, the default rate at which operators are executed. The second assumption is that the start time of an operator is determined by its dependencies. The start time of an operator is dependent on the operators which, according to the theory, are hypothesized prerequisites for its execution. The earliest start time of an operator is the latest start time of all of the operators on which it is dependent.

For many years the only method for composing a CPM-GOMS model was a painstaking manual process that relied as much on the modeler's skill and experience as the underlying theory. A recent attempt to automate CPM-GOMS output is described in Apex-CPM [7]. This method relies on a greedy algorithm to schedule cognitive, perceptual, and motor operators based on a procedural description of the task.

This paper introduces a fundamentally different approach to generating predictions of expert behavior based on similar GOMS-like assumptions. The approach taken here is different in that it involves casting the problem of predicting skilled interactive performance as a constraint satisfaction problem. In the sections that follow, we first describe what is meant by constraint satisfaction and highlight some of its potential advantages; we then describe our approach to the application of constraint satisfaction to cognitive modeling. We call this approach, Cognitive Constraint Modeling (CCM). Two important features of cognitive constraint modeling are (1) that it supports the formal reification of the constraints underlying skilled human performance, and (2) it supports derivation of the implications of these constraints.

We report our explorations of the use of CCM to formally specify CPM-GOMS in a set of axioms and to model a set of tasks for the Collaborative Information Portal (CIP) (one of the tools being developed at NASA Ames to support the Mars Exploration Rover '03 mission task). We call this specification of CPM-GOMS, CCM-d because it is a formal specification of the hypothesis that interactive cognition can be modeled by specifying the *dependency* relations between processes. The results suggest that constraints can be expressed in such a way as to increase, relative to previous techniques, the reuse of code from one model to another. They also suggest that the dependency axioms under-constrain the set of possible schedules. The subspace of possible schedules generated includes those that are both cognitively plausible and implausible. This space has traditionally been narrowed by the expert CPM-GOMs modeler selecting by hand a plausible schedule and more recently with a set of additional assumptions implemented in Apex-CPM. In the general discussion possible responses to this finding are discussed.

CONSTRAINT SATISFACTION

A constraint is simply a logical relation between variables. Bartak [11] lists the following properties:

1. constraints may specify partial values. (E.g. X>5 constrains the value of X without uniquely specifying it.)
2. constraints are heterogeneous. (They can specify constraints between variables with different domains.)
3. constraints are non-directional. (E.g. X=Y+2 can be used to compute the value of Y as well as the value of X.)
4. constraints are declarative. (I.e. they specify a relationship without specifying how that relationship is to be computed.)
5. constraints are additive. (I.e. the order in which constraints are imposed does not matter.)
6. constraints are rarely independent. (They share variables.)

These properties are of interest because they suggest that constraint satisfaction has the potential to provide a formal framework for the specification of theories of interactive cognition, and thereby for the construction of mathematically rigorous tools for supporting the prediction of behavior. Of central importance is the fact that constraints are declarative and additive. These properties should allow theoretical assumptions to be expressed in a computable form that is relatively independent of the arbitrary constraints that are sometimes imposed by the machine, or software algorithms, with which computation is conducted.

The fact that constraints are additive also allows different psychological assumptions to be stated separately. They can be stated in such a way that a collection of well-formed hypotheses about the nature of interactive cognition can be reified, refined, and their implications tested, in a modular fashion. Constraint-based definitions of psychological assumptions are also generative in a way that procedural descriptions are not. Procedural descriptions only implicitly represent constraints, making them difficult to modify and manipulate consistently.

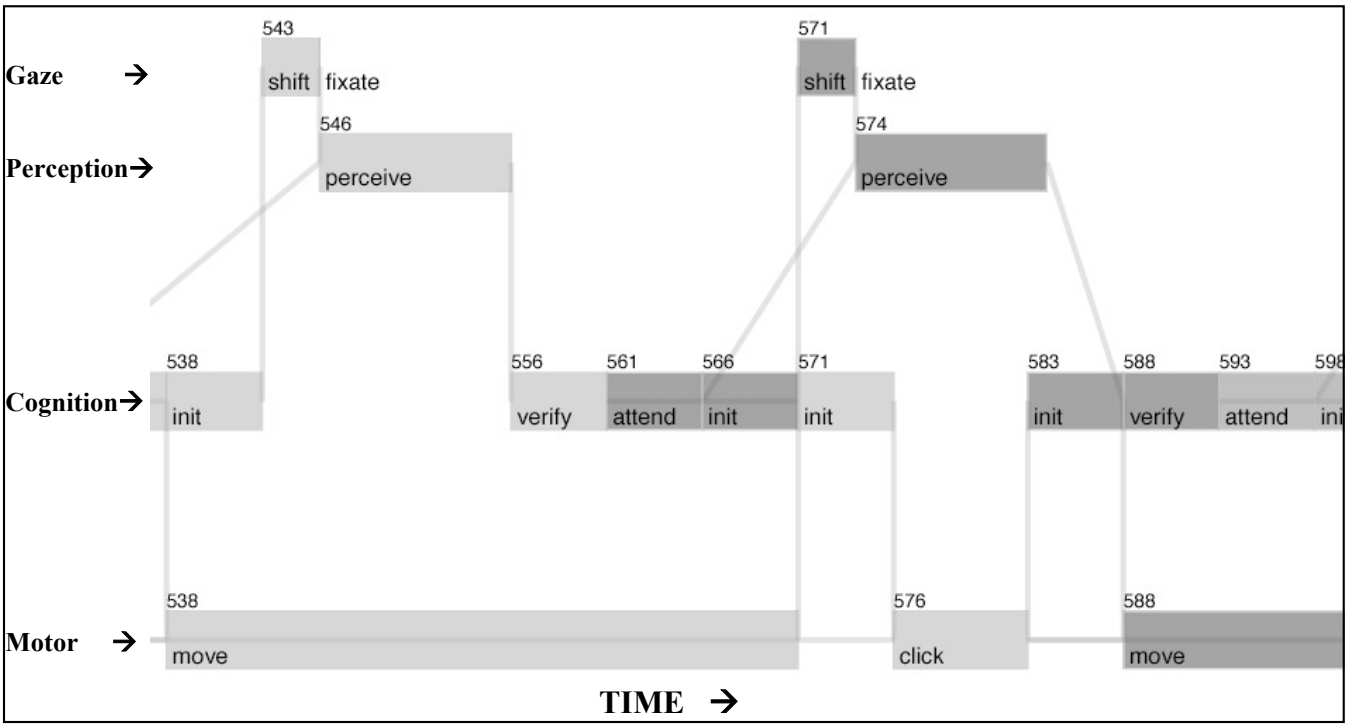

Figure 1: A CCM schedule of the cognitive, perceptual and motor operators required to click on the "person cell" and then click on the "person widget". Note the prediction that attention will be shifted to the "person widget" prior to the first click.

Lastly, because constraints allow the specification of what is to be computed without specification of how the computation is carried out (the algorithm), considerable flexibility is enabled in the desired properties of the schedule. In contrast, Apex-CPM models are intrinsically bound to Apex's greedy scheduling algorithm and ACT-Simple does not take advantage of ACT-R's learning mechanisms. In contrast, an approach to modeling cognition that is based on constraint satisfaction can, in principle, produce schedules that optimize some feature of the schedule. Of particular interest is the possibility of optimizing (i.e. minimizing) the overall performance time.

Optimal schedules are of particular interest in predicting interactive cognition in that they suggest the asymptotic bound on skilled behavior [16]. In contrast, while there is some favorable evidence [7] supporting the view that greedy scheduling approximates some level in the range of skilled human performance, it is not clear exactly what this level is.

One of the earliest reported examples of the application of constraint satisfaction was Waltz's work on the problem of labeling the vertices in a line drawing as either concave or convex [15]. More recent work, has led to the implementation of Constraint Logic Programming (CLP) environments [12,13]. This work has been successful in finding applications in scheduling and planning, but has not yet, to our knowledge, been applied to the problem of predicting the time-course of interactive cognition. Sicstus Prolog offers a CLP module for finite domains (CLP FD) that we are currently using to compute the implications of task and psychological constraints for skilled behavior.

A CONSTRAINT SATISFACTION APPROACH TO PREDICTING SKILLED PERFORMANCE

A fragment of a predicted schedule generated by CCM-d for a database entry task is given in Figure 1. The schedule consists of the cognitive, perceptual and motor operators required to click on the 'person cell' (yellow operators) and then click on the 'person widget' (orange operators) with a mouse. The basic structure of the output is derived from that produced by Apex-CPM [7] and earlier work on CPM-GOMS [2]. Each operator (process) is denoted by a box. In this case, the operators are organized into four rows, where each row represents a mental processor as defined in the Model Human Processor [8]. The top row is for eye movements, then visual perception; the next for cognition; and the bottom for right-hand motor actions. Time is represented from left to right. Boxes are elaborated with a start time and an end time and are connected by lines that represent unidirectional dependencies (an operator is dependent on connected operators to its left). The length of the boxes is to scale. Dependency loops are not possible.

A crucial feature of the schedule is that operators are interleaved. It is predicted that attention will shift to identifying the 'person widget' prior to the completion of the mouse move to 'person cell'. The model supports the

prediction that user will act as if to anticipate the next action in the sequence, by increasing the extent to which operators are concurrent, and reducing the overall time cost of executing the task.

The eye movement operator in Figure 1 overlaps with a click cursor operator. The prediction is that the eyes will move to target B while the hand is still acting on target A. Importantly, this overlap is not explicit in the input description rather it is calculated by combining the task, strategy, and architecture constraints together with the goal of minimizing performance time.

CCM is implemented in a Prolog-based tool called CORE (Constraint-based Optimal Reasoning Engine). There are two phases to the generation of predictions form a CORE model. In the first phase, an input description is translated into an internal set of CLP constraints. If operator B is dependent on operator A, then operator A must finish before operator B can start. In the second phase, durations are assigned to operators and, constrained by the dependencies, a schedule is computed.

Phase 1: Interpreting the description language

The input description is divided into four parts:

1. The task description. A hierarchal analysis of the interaction between a person and the task environment required to achieve a specified task or set of tasks.
2. The environment description.
3. The cognitive strategy description. Strategies for achieving simple routine tasks such as moving a computer mouse, turning a knob, or clicking a button.
4. The architecture description. Definitions of fundamental mental operators, resources and their parameters and dependencies.

The advantage to layering the description is the potential for ensuring that all templates conform to the same set of architectural constraints, and in providing a framework within which the role of further architectural constraints (e.g. memory capacity) can be explored. An accurate specification of the architecture and world layers should simplify the writing and comprehension of cognitive strategies (templates). The architecture description provides a specification of the obligatory relationships between operators, in terms of both their effects (e.g. an init causes a motor action) and their preconditions (e.g. a visual perceive(x) must be preceded by an eye movement to x.

In the paragraphs that follow, each part of the description language is illustrated for the task of typing a password into a mouse-driven interface, describing how constraints are expanded into a predicted schedule of primitive operators.

Proposition 1 describes a typical task interaction consisting of a sequence of dependent move_click (mc) subtasks. The mc subtask is defined with proposition (2). Proposition 2 expands into a sequence of two operators consisting of a mouse movement followed by a click. The click is dependent on the move (denoted with a minus sign '-'). Both of these operators represents a primitive interaction between the user and the task environment.

```
t( enter_password,
        mc( k4 )
        - mc( k9 )
        - mc( k8 )
        - mc( k0 )
        - mc( k1 )
        - mc( enter ) ).                                    (1)

t( mc( X ),
   motor(hand, move, X ) - motor(hand, click, X) ).
                                                            (2)
```

The propositions are translated into mathematical relations between the start times and durations of the specified operators and are then posted to a CLP constraint store.

The dependency graph is further elaborated by the interpretation of propositions that specify constraints determined by the environment or device. The device used in Vera et al.'s experiment was mouse-driven and therefore it is a pre-requisite of clicking on a button that the cursor is at the button:

```
c( motor(hand, click, X ),
        at( cursor,X ) - motor(hand, click, X )).           (3)
```

Further propositions are required to represent micro-strategies. Propositions 4 and 5 define micro-strategies for achieving a click interaction between the user and the device. Proposition 4 defines the slow move click strategy previously presented in [1].

```
c(init(hand,click,X),
verify( X ) - verify( at(cursor,X) ) - init(hand,click,X) ).  (4)

c( [init(hand,click,X),in(X,[correct,card_2, ... ]) ],
        verify( X ) - init(hand,click,X) ).                 (5)
```

Proposition 4 states that if there is an init(hand,click,X) operator in the dependency graph then the init is dependent on a verification that the cursor is at the correct location, and that verification is dependent on the verification that X is the required target.

Proposition 5 defines the fast move click strategy also defined in [1]. This example is an elaboration of (4) that specifies that the verify(X) should be added only if the value of X is in the set [correct, card_2, …] (where '…' indicates that there are more names than printed above). This construction allows micro-strategy choices to be contingent on parameters such as target names.

```
(procedure
(index (slow-move-click ?target))
(step c1 (initiate-move-cursor ?target))
(step m1 (move-cursor ?target)
         (waitfor ?c1))
(step c2 (attend-target ?target))
(step c3 (initiate-eye-movement ?target)
         (waitfor ?c2))
(step m2 (eye-movement ?target)
         (waitfor ?c3))
(step p1 (perceive-target-complex ?target)
         (waitfor ?m2))
(step c4 (verify-target-position ?target)
         (waitfor ?c3 ?p1))
(step c5 (attend-cursor-at-target ?target)
         (waitfor ?c4))
(step w1 (WORLD new-cursor-location ?target)
         (waitfor ?m1))
(step p2 (perceive-cursor-at-target ?target)
         (waitfor ?p1 ?c5 ?w1))
(step c6 (verify-cursor-at-target ?target)
         (waitfor ?c5 ?p2))
(step c7 (initiate-click ?target)
         (waitfor ?c6 ?m1))
(step m3 (mouse-down ?target)
         (waitfor ?m1 ?c7))
(step m4 (mouse-up ?target)
         (waitfor ?m3))
(step t1 (terminate)
         (waitfor ?m4)))
```

Figure 2: A PDL description of a slow-move-click template (reproduced from John et al. (2002).

The next part of the descriptions consists of a set of propositions that define cognitive architectural constraints. In the case of this example the constraints are intended to make explicit the CPM-GOMS theory of the human cognitive architecture.

c(motor(E, A, X),
init(E, A, X) - motor(E, A, X)). (6)

c(verify(X),
perceive(X) - verify(X)). (7)

c(perceive(X),
attend(X) - perceive(X)). (8)

c(perceive(X),
motor(gaze,move,X) - perceive(X)). (9)

Proposition 6 states that if there is a motor action in the dependency graph then there must also be a cognitive init operator for the motor action, and the motor action must be dependent on the init. Proposition 7 captures the assumption that in order to verify(X), X must first be perceived. It states that verify is dependent on perceive. Proposition 8 captures the assumption that in order to perceive(X), X must first be attended to, and proposition 9 captures the assumption that in order to perceive(X), the gaze must first be moved to X. The intention here is not to make a commitment to these particular assumptions, but rather to provide an explicit and computable statement of the assumptions underlying one particular approach to modeling.

The resource requirement of an operator is defined using the 'resource' proposition, as in the following example:

resource(perceive(tone), audition). (10)

This denotes that perceive(tone) is executed by the audition processor. Every operator has a statement of its resource requirement.

Operators are assigned a duration using a simple inheritance mechanism. Some operators, e.g. motor actions that are dependent on Fitts's Law, are assigned values that are particular to the instantiation of the operator, and which in the case of Fitts's Law can be calculated using information about the arrangement and size of objects in the world. Other operators may have durations that are inherited from their processor (e.g. eye movements). The durations of operators are specified in propositions of the form:

duration(perceive(tone), 100). (11)

duration(motor(hand,move,A,B),fittss(A,B)). (12)

duration(processor, cognitive, 50). (13)

Where proposition 11 captures the assumption that perceive(tone) takes 100ms, proposition 12 that the duration is determined by Fitts's law and proposition 13 that cognitive operators have a default duration of 50ms.

Phase 2: Computing time predictions

The various sources of constraint on operator scheduling (task, environment, strategy, architecture) are brought together as a set of arithmetic constraints in the CLP FD constraint store. This requires interpreting the particular task, strategy, and architecture assumptions (as defined above) as statements that further constrain the axiomatic assumptions of CPM-GOMS. These axioms essentially state that a CPM-GOMS model consists of a set of processes, each with a resource, and each with a set of dependencies, that only one process may occupy a particular resource at any one time, and lastly, that a process must start after all of the processes on which it is dependent have finished:

Given a set of processes P_i ($i=1...n$) each represented by a set of attribute=value features, and each with a duration D_i, start time S_i, end time $E_i = S_i + D_i$ and resource requirement Ri the following must hold:

1. For all P_i with features {*name=start*}: $S_i=0$.
2. For all P_i: $S_i >= 0$.
3. For all P_i, P_j, where dependent(P_i,P_j): $E_j =< S_i$.
4. For all P_i, P_j, where $R_i = R_j$: $E_i =< S_j$ or $E_j =< S_i$

where dependent(P_i,P_j) is true if P_i is dependent on P_j according to the definitions in the previous section.

Constraints on start times and durations of the operators (whether cognitive, perceptual, or motor) required to

perform the task are represented along with their dependencies.

However, the values of start and end times are still to be calculated. Predictions have been generated from CCM with both greedy scheduling and optimal scheduling. An optimal scheduler was used to generate the pert chart in Figure 1. In contrast, with a greedy scheduler, like that used by Apex-CPM [7], operators are scheduled as early as possible, i.e. once all of their dependencies have been met. A discussion of the suitability of different kinds of schedulers is presented below.

APPLICATIONS OF CCM-d

Predicting interactive skill

Using CCM-d we have derived predictions for a number of interface evaluation problems. These include: (1) the analysis of a call-center interface built by Convergys Inc. that involved interaction between a customer, an agent, and the computer system. The CCM-d analyses were used to help selection between competing interface designs. (2) predicting how long it would take a skilled user to enter shift assignments into the Collaborative Information Portal (CIP), one of the tools being developed at NASA Ames to support the Mars Exploration Rover '03 mission. (3) predicting the performance of participants using the laboratory ATM task previously reported by John et al. (2002).

All three of these analyses involved the semi-automatic generation of interleaved schedules such as that illustrated in Figure 1. The first of the tasks (Convergys) offered particular challenges in that it required modeling the interaction of an operator with a customer and with the system (similar to the original work by Gray et al. [2]). CCM-d was used to predict a highly interleaved schedule that involved much concurrency in the interactions (e.g. talking, while moving the mouse, and waiting for a system response) but without much of the hand-crafting of the schedule that would be required without some form of automation.

The reification of constraints

One of the claims that we made for the advantages of a constraint satisfaction approach to predicting skilled interactive cognition was that it allowed a clear separation of different levels of constraint. This is evident through our description of the four levels of constraint specification in the previous section.

The consequences of separating these constraints are two-fold. First, they reify the underlying psychological and task theory. Second, each statement is a self-contained assertion of a universally quantified assumption about the nature of interactive cognition. Each statement, therefore needs stating only once.

To fully appreciate the advantages of the separation of constraints it is worth contrasting the CCM descriptions to Apex-CPM procedural descriptions. Figure 2 is a reproduction of the Apex-CPM specification of Gray and Boehm-Davis's slow-move-click template [1] taken from [7]. The template is described in a procedural form, where each operator is labeled (e.g. step m1) and if there is a dependency of an operator on a previous operator then a waitfor instruction is used to denote the dependency. The waitfor instructions are used to locally capture obligatory architectural and strategic dependencies. E.g. the dependency that links step m1 and step c1 is an obligatory architectural constraint. In contrast the dependency that links c4 to p1 is a strategic dependency in the sense that it is a matter of choice whether a verify operator follows perception.

A template description in Apex-CPM consists of a mixture of architectural and strategic dependencies. This is not a general representation of the relationship between init and motor operators -- the same architectural rule must be recreated in each template in which it occurs. To put it another way, in Apex-CPM, there is no universally quantified assertion of the architectural relationships between operators. In contrast, and as we have illustrated, with constraint satisfaction and the constraints defined above, if it is specified that there is a motor operator in the schedule then it is required by constraint 6 that there is a corresponding init operator, irrespective of which strategy the motor operator is for. The constraint captures a universal assumption about the nature of cognition.

GENERAL DISCUSSION

We have reported current work on the value of taking a constraint satisfaction approach to formalizing the assumptions implicit in CPM-GOMS, and thereby providing a mechanism for predicting the time-course of skilled interactive cognition. A distinctive aspect of the approach is that it involves the specification of declarative and additive constraints on cognitive behavior followed by a process of reasoning about their implications. This is achieved with the implementation of a tool for translating declaratively specified task and psychological constraints into a Sicstus Prolog implementation of CLP FD. The approach has the following two strengths:

(1) As constraints are additive, a clean separation can be imposed between task specific, strategy specific, and psychological constraints. This separation ensures the reusability of the appropriate constraints as new models are built. It also helps ensure that an analyst can work at exactly the right level. It is possible, in contrast to Apex-CPM, to specify strategy constraints without knowledge of the underlying architecture.

(2) As constraints are declarative, a clean separation can be imposed between what is to be computed (in this case a prediction of the time-course of operators) and how it is to be computed. This is particularly important in the relationship between the constraints on cognition and the scheduling algorithm used. It is possible to make formal

derivations of both optimal and greedy schedule from CCM descriptions. The difference between these schedules is important (they represent different hypotheses as to the structure of skilled behavior), but the algorithm by which they are computed is not. It happens that the mechanisms we have used for reasoning about CCM descriptions takes advantage of a Sicstus Prolog CLP FD implementation of a branch-and-bound algorithm in order to produce optimal schedules. We could, however, have used an entirely different algorithm to compute the same optimal strategy. The algorithm is irrelevant to the particular theory of cognition. The constraint-based approach helps ensure that the statement of task and psychological constraints is uncluttered by irrelevant specification that is present in order to get the scheduling algorithm to work.

As discussed above, our application of constraint satisfaction techniques to a large-scale task (CIP Scheduler) provided evidence that supports the claimed strengths of the approach. However, it also exposed a weakness. The particular ontology that we have used to determine the entities between which relations were expressed is taken from CPM-GOMS. It is an ontology for expressing the relations between operators in terms of dependencies (if B is dependent on A then the start time of B is greater than the end time of A). However, the fact that it is not, for example, possible to specify the bounds on the delay between the end of A and the start of B means that operators may be scheduled much earlier than is cognitively plausible. Consider the following example. If a prerequisite of a motor action, click(x) is a cognitive operator init(x) then we could say that click(x) is dependent on init(x). Similarly we could say that click(y) is dependent on init(y). But the problem is that the specification of this pair of dependencies is insufficient to ensure cognitively plausible scheduling of the four operators. For example, the schedule ordering init(x), init(y), click(y), click(x) is legal but cognitively implausible.

In order to solve these problems, Apex-CPM included mechanisms to constrain greedy scheduling that were in addition to the standard CPM-GOMS dependencies between operators. John et al. [7] also provided Apex-CPM with information, for example, about virtual resources. Virtual resources were a mechanism that was deliberately, and successfully, introduced in Apex-CPM in order to prevent cognitively implausible scheduling. Other mechanisms added to standard CPM-GOMS assumptions include operator priorities (to ensure task ordering) and a mechanism to prevent interleaving of operators into gaps that were smaller than the operator duration.

In contrast, the solution to the same problem (cognitively implausible orderings) was solved in CCM-d by using specifications of dependencies and no more. To solve the init(x) - click(x) problem in a CCM-d, rules were specified that, for example, constrain click(y) to be dependent on click(x) if init(x) must come before init(y). This solution has the advantage that it is expressed purely in terms of dependencies. However, it has the disadvantage that many such extra dependencies are needed to adequately constrain a schedule. This was demonstrated by the number of dependencies that were needed to constrain the excel task reported above beyond the small-set of template constraints borrowed from the work of Gray and Boehm-Davies [1]. Describing all of the constraints necessary to produce a cognitively plausible schedule with CCM-d is a problem that exposes the extent to which craft knowledge was required in the manual composition of a CPM-GOMS schedule.

One way in which this issue could be addressed would be to supplement CCM-d descriptions with statements that capture the mechanisms provided in Apex-CPM (i.e, a formal specification of Apex-CPM). However, we have moved in a different direction, directly addressing the fact that the specification of the relationships between processes in terms of dependencies is inadequate because it does not support the specification of:

1. The maximum duration of the gap between two processes.
2. Constraints on whether a process can be scheduled between two other processes.

Our current research focuses on specifying a set of entities and relations within a CCM framework that address these problems, and that therefore adequately constrain scheduling while at the same time maintaining the advantages of formal specification of cognitive theory. A crucial feature of this approach is that constraint satisfaction techniques provide a computational substrate for reasoning about these formal specifications.

REFERENCES

1. 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.
2. 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, 8* (3), pp. 237--309.
3. Scott E. Hudson , Bonnie E. John , Keith Knudsen , Michael D. Byrne, A tool for creating predictive performance models from user interface demonstrations, *Proceedings of the 12th annual ACM symposium on User interface software and technology*, p.93-102, November 07-10, 1999, Asheville, North Carolina, United States
4. John, B. E. (1988) *Contributions to Engineering Models of human-computer interaction*. Ph.D. Thesis. Carnegie Mellon University.

5. John, B. E. (1996) TYPIST: A Theory of Performance In Skilled Typing. *Human-Computer Interaction , 11* (4), pp.321--355.
6. John, B.E., Wayne D. Gray, GOMS analysis for parallel activities, *Conference companion on Human factors in computing systems*, p.395-396, April 24-28, 1994, Boston, Massachusetts, United States
7. John, B.E. Vera, A., Matessa, M., Freed, M., Remington, R. (2002). Automating CPM-GOMS. *Proceedings of Human factors in computing systems, 4*,1, 147-155.
8. Stuart K. Card , Allen Newell , Thomas P. Moran, *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Associates, Inc., Mahwah, NJ, 1983
9. Kieras, D. E. (1996). Guide to GOMS model usability evaluations using NGOMSL, *The Handbook of Human-Computer Interaction*, M. Helander and T.Landauer (Eds.), 2nd ed. North-Holland Amsterdam.
10. Kieras, D. E., Wood, S. D., Abotel, K., & Hornof, A. (1995). GLEAN: A Computer-Based Tool for Rapid GOMS Model Usability Evaluation of User Interface Designs. *International Journal of Human-Computer Studies, 22*, 365--394.
11. Bartak, R. (1999). Constraint programming- What is behind? *Proceedings of CPDC99 Workshop* (invited talk), Gliwice, June 1999, 7-15.
12. Jaffar, J., & Lassez, J.L. (1987). Constraint logic programming. In *Proceedings of the ACM Symposium on Principles of Programming Languages, ACM*, 1987.
13. Jaffar, J., Maher, M., Marriott, K., Stucket, P. (1994). The semantics of constraint logic programs. *Journal of Logic Programming, 20*, 1-47.
14. Salvucci, D. D., & Lee, F. J. (2003). Simple cognitive modeling in a complex cognitive architecture. In *Human Factors in Computing Systems: CHI 2003 Conference Proceedings* (pp. 265-272). New York: ACM Press.
15. Waltz, D.L. (1975). Understanding line drawings of scenes with shadows. In *The Psychology of Computer Vision*. McGraw Hill, New York.
16. Anderson (1990). *Rational Analysis*. LEA.