Supporting Efficient Development of Cognitive Models at Multiple Skill Levels: Exploring Recent Advances in Constraint-based Modeling

Irene Tollinger[1], **Richard L. Lewis**[2], **Michael McCurdy**[1], **Preston Tollinger**[1], **Alonso Vera**[1], **Andrew Howes**[3], **Laura Pelton**[1]

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

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

[3]School of Psychology
Cardiff University
Cardiff, UK
howesa@cardiff.ac.uk

ABSTRACT
This paper presents X-PRT, a new cognitive modeling tool supporting activities ranging from interface design to basic cognitive research. X-PRT provides a graphical model development environment for the CORE constraint-based cognitive modeling engine [7,13,21]. X-PRT comprises a novel feature set: (a) it supports the automatic generation of predictive models at multiple skill levels from a single task-specification, (b) it supports a comprehensive set of modeling activities, and (c) it supports compositional reuse of existing cognitive/perceptual/motor skills by transforming high-level, hierarchical task descriptions into detailed performance predictions. Task hierarchies play a central role in X-PRT, serving as the organizing construct for task knowledge, the locus for compositionality, and the cognitive structures over which the learning theory is predicated. Empirical evidence supports the role of task hierarchies in routine skill acquisition.

Author Keywords
user modeling, tools for usability evaluation.

ACM Classification Keywords
H.1.2. Human Factors, H.5.2 Theory and Methods

INTRODUCTION
Starting with Card, Moran and Newell [4] and continuing through the present, modeling for *a priori* predictions of human performance on interface-based tasks has matured into an established subfield of Human-Computer Interaction. The GOMS family of methods have been validated by a large research pool (e.g. 5,6,11), and detailed computational theories of human cognitive architecture such as ACT-R, Epic, and Soar have been applied successfully to models of complex interactive tasks [1,15,17,20].

However, in practice, there has been little use of either the GOMS methods or cognitive architectures in an applied design context. There are many complex reasons for this state of affairs, but one plausible and commonly-assumed barrier to the adoption of current methods is that the cost of modeling using these methods is too high relative to the benefit of the human performance predictions generated. Recently the research community has responded with modeling support tools that simplify and to some degree automate the modeler's task including CogTool [9], Behavior Recorder [12], Act-Simple [18], ACT-Stitch [14], Apex-CPM [10], G2A [19], and User Modeling Design Tool [16]. These efforts represent important individual successes, and the work reported here builds on them to some degree. However, a common property of this work is that it introduces a set of unfortunate tradeoffs between ease of use and predictive power. For example, ACT-R contains detailed process theories of declarative and procedural learning, but these are not accessible via some of the modeling tools [9,12,18] based on ACT-R. Other tools are concerned with usability and provide advances by automating some aspects of modeling [14,10,19,16], but still require time, knowledge of psychology, and programming skills beyond the resources available in applied design contexts.

This paper provides an initial report on the development of X-PRT, a new tool for developing cognitive models that is intended to support modeling activities in service of goals ranging from interface evaluation and design to basic cognitive research. X-PRT departs from existing efforts to build affordable modeling tools in several significant ways. First, it supports the automatic generation of predictive models at multiple skill levels from a single task-specification. Second, it directly supports the compositional reuse of existing cognitive model components. And third, it was designed and implemented from the ground-up with user-centered design principles to support a comprehensive set of modeling subtasks. It is not an exploration of a single new piece of technology for making modeling easier, but rather a complete environment for model development.

The first two features exploit recent new technical advances in cognitive modeling made possible by CORE [7,13,21]. CORE stands for Constraint-based Optimizing Reasoning Engine and is an implementation of a new general approach to cognitive modeling based on constraint-reasoning and optimization. X-PRT is the graphical model development environment for CORE. Thus, X-PRT is jointly shaped by the capabilities of CORE and a ground-up design effort to support a comprehensive set of cognitive modeling activities.

A key distinction of X-PRT over other similar efforts is that it represents a significant attempt to preserve as much predictive power as possible (the "benefit" side of the cost/benefit equation) while at the same time lowering the barriers to entry in terms of usability and required modeling expertise (the "cost" side of the equation). This is an especially difficult tradeoff to manage. Typically as predictive power is increased, so is the complexity of the tool. Therefore, most attempts to date have made significant sacrifices to predictive power in an effort to decrease cost. However, X-PRT allows the user to take advantage of powerful advances in the underlying modeling engine (CORE), in particular the ability to simulate users of varying skill from a single model.

The remainder of this paper has the following structure. First, there is a task analysis of cognitive modeling, breaking it down into its necessary subtasks. In this analysis the difficulties of each subtask are identified individually (from the point of view of both design and basic research) and used to help derive a set of requirements for an effective modeling development environment. Next, the design of X-PRT is described, with a focus on how the capabilities of CORE are exploited to generate predictions at multiple levels of experience from a single model. Next, there is a brief overview of the essentials of CORE; the reader is referred to [7,13,21] for more complete presentations. Then the empirical human data supporting the psychological reality of the learning theory specified in CORE is briefly reviewed. Finally, comparisons to other approaches and future work are covered.

AN ANALYSIS OF MODELING WORK

Breaking down the task of modeling into subcomponent tasks provides an ontology for discussing current challenges, requirements for an effective modeling support tool, and the relative priority of these tasks for different user goals. These goals lie along a spectrum between modeling to conduct basic cognitive research and modeling to evaluate or design/redesign an interface. Later these tasks will frame the design of CORE and X-PRT as well as help in comparing them to existing tools.

T1. Specifying Task Knowledge

Task knowledge (T1) is the specific information the user must know to accomplish the task. The central problem from the modeler's point of view in current approaches based on cognitive architectures (ACT-R, Epic, Soar) is that there is a mismatch between the level at which task knowledge must be coded (the 50 ms production-rule level) and the level at which the task is naturally specified. Programming these systems also requires learning an idiosyncratic syntax, and the code can be difficult to debug. These problems are exacerbated in learning systems in which the architecture itself is generating new code. As noted earlier, approaches that do offer more abstract and easier task specifications also offer reduced predictive power (e.g., they do not support learning).

In conducting observational user research (Contextual Inquiry method [3]), with modelers doing cognitive research tasks, the authors found that users often start with similar models and use them as a basis for creating new ones. For example, a modeler developed a model of a cockpit keypad based on a model of an Automated Teller Machine (ATM). However, current systems do not provide a systematic way to build more complex models from existing components. For example, in the telephone operator domain there are many subtasks that require listening to an utterance and entering the result into a field but the parameters (the utterance itself and the appropriate field) are different. Modelers currently accomplish some degree of reuse by copying and pasting.

The ability to effectively represent the task is likely to be equally relevant for users with the goal of conducting cognitive research as interface evaluation and design. Perhaps it is slightly more important in the case of interface evaluation and design because real world interactive tasks are more likely to be relatively complex.

Thus, requirements for the improvement of task knowledge specification (T1) include: a representation that abstracts away from the complexities of an architectural implementation, the ability to create general, reusable tasks, and the ability to compose more complex tasks from these reusable components.

T2. Specifying Operator Skill

The specification of operator skill (T2) is an explicit statement about the level of skill that an operator brings to a specific task (perhaps specified in terms of number of practice trials). In the GOMS framework, the specification of operator skill is accomplished implicitly by choice of method. If the modeler wishes to predict behavior early in practice, she chooses an appropriate method like the Keystroke Level Model (KLM) [4] and develops a model. If on the other hand a modeler wishes to predict behavior late in practice, she chooses a method like CPM-GOMS. This is the type of strategy Baskin and John used to predict behavior early and late in practice [2]. Kieras used a different rule set in each of the two models to predict different levels of skill [8].

The alternative is to build a model in a learning architecture such as ACT-R and use the architectural mechanisms to generate predictions at multiple levels of practice. There is an enormous cost to doing this: it is far more difficult to build robust, functioning learning models than it is to build non-learning performance models at some fixed skill level. The nature of the learning is extremely sensitive to the details of the knowledge representation, and it is more difficult for modelers to debug code that they did not generate.

There is also no easy way in any of the existing frameworks to model tasks that are heterogeneous mixes of routine, semi routine, and novice skill as are many real world tasks. To take an applied example from NASA exploration missions, the Mars mission uplink process is well-practiced (done on a daily basis) but the content of each day's mission plan is unique. Users pursing both cognitive research and interface evaluation and design will need to specify operator skill. However, the relevance of this functionality will vary more based on the particular user base, task, and domain being modeled rather than the modeler's focus on research versus design.

In sum, the requirements for the specification of operator skill (T2) include: the ability to generate learning models with no additional cost beyond specifying the performance model, the ability to generate multiple levels of skill from a single task specification, and the ability to specify tasks composed of subtasks that are heterogeneous with respect to skill level.

T3. Specifying The Environment (Interface)

The specification of the environment or interface (T3) is the information about the device the model needs in order to make predictions. This specification can be simple or complex. However, in order to support efficient modeling, the goal should be to specify the minimum information necessary (i.e., only the points at which the user interacts with the environment). Otherwise, the modeler is in the position of having to code the interface or at least hook up the running code to a model. On the simpler side, tools (e.g. Apex-CPM) often require either the inputs to Fitts's Law (distance and size) or the product of the Fitts's Law calculation [10]. Based on Contextual Inquires conducted by the authors of modelers specifying an interface in Apex-CPM [10], the process of measuring screen elements on a printout with a ruler can take several hours. This process is prone to measuring and input errors. CogTool, an existing system supports improved interface specification involving importing screenshots and indicating widgets in a WYSIWYG manner [9].

In addition to position and dimension information necessary to calculate targeting times using Fitts's Law, often it is necessary to encode some form of interactivity within a model. For example, many systems exhibit a delay between when an action takes place and when feedback is received. As in the case of target specification, solutions to describing interactivity range from encoding specific events in a lightweight way to connecting models to running interfaces in the world.

To generalize, requirements for the improvement of the specification of environment or interface (T3) include: a minimal description of the physical layout of the device abstracted above the level of code, automatic calculation of Fitts's Law, the ability to define widgets whether on screenshots of existing interfaces or new interfaces in a WYSIWYG manner, and a simple notion of interactivity.

T4. Specifying The Architectural Assumptions About Human Cognition

The specification of the architectural assumptions about human cognition (T4) is the encoding of a theory of the fixed information processing structure of the mind. In practice, existing tools (ACT-R, Soar, Apex-CPM, etc.) encode these assumptions in a manner that makes them difficult to inspect, understand, or modify. For example, in ACT-R the assumptions are buried in underlying Lisp code. It is not currently possible to compare two models with different architectural assumptions without building multiple models. The user would have to develop models in different systems (e.g. one model in ACT-R and one in Soar). Specification of architectural assumptions is a task squarely in the realm of cognitive research. Interface evaluation and design requires a reasonable set of assumptions that underlie any particular model.

Thus, requirements for the improvement of the specification of the architectural assumptions (T4) include: an explicitly stated set of assumptions, an abstraction above the code level, and an ability to run a single model with different architectural assumptions.

T5. Specifying Strategy (How Skills Are Put Together)

Task strategies are defined as the composition of existing behaviors or skills in service of the specific new task goals at hand. These existing behaviors or skills may be the lowest-level, architecturally defined primitives (such as the cognitive initiation of a button-press), or higher level existing skills, which are themselves composed of more primitive elements (such as the composition of a mouse-movement-and-click strategy from a set of lower-level cognitive, perceptual, and motor primitives). The difference between the task level and strategy level is that the task specification describes the high-level requirements in terms of the interface itself, while the strategies specify how a particular cognitive-perceptual-motor architecture is actually used to realize the task requirements. In short, strategies bridge the gap between the abstract task and a specific cognitive architecture.

Defined this way, strategies encompass both GOMS-like methods [4] and multi-tasking strategies sometimes associated with executive function, as realized in detail in EPIC work [15]. The latter may range from quite task-specific to fairly general. The EPIC work in particular has revealed the complexity of such executive strategies in even putatively simple multi-tasking situations.

Whether the strategies are task-specific methods or executive multi-tasking methods, in existing approaches, strategies are also represented as code with its attendant problems. An effective modeling tool should support reuse of strategies across tasks and models. This is both a methodological and theoretical injunction: the modeler will find modeling more efficient to the extent that existing code can be reused, and the human operator that is being modeled is also bringing to bear existing skills that will be composed somehow to accomplish the novel task demands.

Although specification of lower-level strategies may primarily be the province of basic cognitive research, it must be possible for a user involved in an interface design activity to specify new strategies as they develop novel interaction methods for which strategies do not yet exist. For the most part, users performing interface design activities should rely on reasonably complete libraries for standard interface strategies like move and click or click and drag.

In sum, requirements for the specification of task strategies (T5) are similar to those of T2, and include: the ability to develop libraries of reusable strategies that work across tasks and models, and the ability to compose these strategies in the service of more complex tasks.

T6. Extracting Meta-Information About Behavior From Model Output

The extraction of meta-information about behavior from model output (T6) is the task of interpreting the behavior predictions that the model produces. Many systems produce trace output in which each low-level model action is documented. This type of trace is often many pages long and includes system oriented data (unique ID tags, long time/date stamps) which coupled with the sheer volume of information render it not human readable. Earlier manual methods and Apex-CPM have provided users with Gantt charts that are very long to scroll through and offer information at a small grain size (individual motor, perceptual, and cognitive operators of 50 milliseconds). Some systems provide only the total predicted time. This interpretation of predictions is relevant to both research and design activities. Users doing both activities would likely benefit by a high-level overview of their model to check for mistakes. Users engaged in interface design would benefit from suggestions as to the problems with the current task/interface combination (a particular move takes a long time or many items are stored in working memory at the same time). Users engaged in cognitive research, on the other hand, are likely to go from a high-level overview into a detailed representation like a Gantt chart where low-level assumptions about cognition are made explicit and can be verified.

Thus, requirements for the improvement of the extraction of meta-information about behavior from model output (T6) include: a high level overview, an improvement suggestion facility, and appropriate detailed views.

T7. Performing Overhead Tasks

Performing overhead tasks (T7) is a broad category for the myriad of activities that do not directly contribute to a particular component such as the task or user interface but are nonetheless required in order to model. These tasks include: installation, file management, command line model execution, versioning, etc. Often these research toolkits are composed of multiple applications (e.g. development environment such as LISP, a text editor, a particular web browser) that are often costly or difficult to install. A task like file management may not appear to have a high cost, but as the number of models increases scalability becomes a concern. For example, users of CogTool were able to generate 300 models over a summer but found them difficult to manage (Bonnie John, personal communication). Managing hundreds of files by file system location and naming convention does not appear scalable. When a modeler works at the code level, using currently available tools, she must manage file versioning herself. Unless the modeler saves an explicit version (via "save as") after each model run, which is time consuming, she cannot return to previous version of the model at will. For example, often mistakes are introduced and not noticed until several saves and iterations later. In these cases, the cause of the error can be difficult to reconstruct.

Therefore, the requirements for the support of overhead tasks (T7) include: the ability to easily install and update a single application, a framework to manage the many model components, elimination of command line model execution, and a low overhead versioning scheme.

Summary of Tasks

Table 1 presents a summary of tasks (T1-T7) and their relevance to the disparate goals of cognitive research versus interface design and evaluation. Based on the above discussion, the tasks are broadly categorized as being of high, low, or variable relevance to research versus design.

Component Tasks	#	Research	Design
Task Knowledge	T1	High	High
Operator Skill	T2	Varies	Varies
Interface	T3	Low	High
Architectural Assumptions	T4	High	Low
Strategy	T5	High	Low
Model Output	T6	High	High
Overhead Tasks	T7	High	High

Table 1. Comparison of Task Importance to Research versus Design Goals

The tasks (T1-T7) and their associated requirements listed in this section are by no means an exhaustive list of either. These requirements, defined by a first-principles approach to design (asking what an effective modeling support tool should look like), drove the design of X-PRT. They also serve as an opening position intended to spur discussion within the research community.

DESIGN OF X-PRT

While the focus of this work is the ability to model at different levels of experience, it is necessary to introduce the design of X-PRT at a high level in order to place particular functionality and design decisions in context.

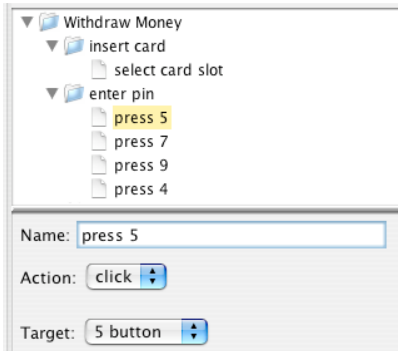
X-PRT, a cross platform Java application, is designed to fulfill the requirements that support the modeler's task as described above. X-PRT's interface metaphor, that of an Integrated Development Environment (IDE), was chosen for two reasons. First, the IDE-like multi-document project metaphor is appropriate for the large number of model components (task, user interface, architecture, strategy, and output) and the high likelihood that a model of a system will consist of multiple subtasks (e.g. withdraw, deposit, check balance, in the ATM context). Perhaps hundreds of tasks will comprise the task set for a particular domain to be drawn upon in composing larger tasks. Second, interface designers are likely to be familiar with the IDE metaphor based on knowledge of rapid prototyping tools such as Microsoft Visual Studio and web design tools like Macromedia Dreamweaver. The X-PRT interface is organized into 3 primary panes: a file pane to manage the model components (task, user interface, architecture, strategy, output), a view pane to view and edit the model components, and a run pane to run the model and generate predictions.

The modeling tasks (T1-T7) will be used as a framework to systematically cover X-PRT design elements. They are presented in the order discussed above for consistency. As of this writing the design elements covered represent functionality already implemented in a first version of X-PRT and user tested as described in the design process subsection at the end of this section.

Design of Task Knowledge (T1)

Given the requirement to represent the task as an abstraction above the code level, X-PRT provides a directed entry interface for the hierarchical specification of the task (Figure 1). The user explicitly specifies some of the same parameters that CORE requires and makes some parameters implicit. For example, using a tree representation for parent rather than the attribute value pair "parent=node ID." Other parameters such as presentation color or unique id's are automatically handled by the system and never presented to the user.

The choice of a file-structure-like hierarchy is based on users' familiarity with this cross-platform convention. Composing larger tasks from component tasks is supported by allowing hierarchy nodes (e.g. the "withdraw money" node below) to be copied and pasted along with all children such that a user can work on multiple tasks within a project and when ready can easily copy them into a larger task.


Withdraw Money
insert card
select card slot
enter pin
press 5
press 7
press 9
press 4

Name: press 5
Action: click
Target: 5 button

Figure 1: Hierarchical task knowledge in X-PRT (ATM task).

This design fulfills two of the requirements listed above: an abstraction above the code level and the ability to build larger tasks from component tasks. More sophisticated reuse, the ability to embed tasks within other tasks such that modifications to a task propagate to all embedded instances, is planned for future design and development as is designating and modeling from explicit templates. However, the current design's improved legibility over code and the explicit presentation of required fields associated with each action (such as "Action" and "Target" in Figure 1) serve to guide the user to enter allowable values and reduce errors.

Design of Operator Skill (T2)

X-PRT aggregates level of skill and similar parameters in a single "Person" dialog box associated with each project (see figure 2). The user sets a slider that represents the desired level of skill.

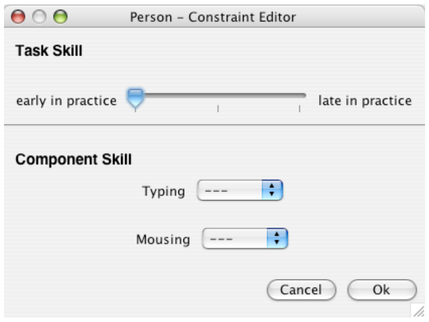

Person – Constraint Editor
Task Skill
early in practice
late in practice
Component Skill
Typing: ---
Mousing: ---

Figure 2: Dialog where level of skill parameters are set.

Though the representation of skill as a slider is a relatively simple design choice, the entire application works to support this level of simplicity in order to predict multiple levels of skill rather than build multiple or more complex models to access this type of valuable prediction. A more thorough presentation of the underlying psychological theory and advances in the CORE engine that make such a simple presentation possible are discussed later in this paper.

This design fulfills the above requirement to generate predictions at multiple levels from a single model. However, it does not yet support heterogeneous mixtures of these skills. The next version of X-PRT will provide the ability to specify the level of skill for each node in the hierarchy.

Design of The Environment or Interface (T3)

X-PRT supports the specification of the user interface in a minimal WYSIWYG manner. If the user has an existing screenshot of the interface, she imports that image and identifies only interactive widgets relevant to her task. She does this by dragging semi-transparent rectangles over them (much like specifying hot spots in Macromedia Dreamweaver). This is based on the method employed by CogTool, which has users import screenshots into Macromedia Dreamweaver, add actual hot spots, and export the HTML in order to get the positions and dimensions of the hot spots [9]. If on the other hand the user does not have a screenshot to import, the system labels each interactive widget with the name field on a plain background to distinguish it from other widgets. In both cases, each interactive widget must be named and can be precisely positioned by tweaking X and Y coordinates.

A user interface can be created consisting of many screens (with or without screenshots) which can be managed (added, deleted, reordered) in thumbnail form in a manner similar to the slide thumbnail pane within Microsoft PowerPoint. X-PRT then calculates the Fitts's Law estimates based on distances between widgets. A benefit of this approach is the modeler does not have to recalculate Fitts's Law if a widget is moved (as part of a redesign) or if the task is changed to press buttons in a different order (meaning the distances between one button and the next change).

This design fulfills three of the requirements set forth for interface description: to provide a physical description of interface layout abstracted above the level of code, enable automatic calculation of Fitts's Law, and allow the use of existing screenshots or define a new interface.

Design of Architectural Assumptions (T4)

While most modeling methods and cognitive architectures embody a single set of architectural assumptions regarding human cognition, CORE encodes assumptions explicitly as constraints and provides the ability to run a given model with different assumptions. Users working on cognitive research have the option to encode different architectural assumptions and test their effect on behavior predictions. For example, Howes et al. encoded both ACT-R style assumptions as constraints and then modified those assumptions to encode EPIC style assumptions in order to explore the effect of the differences on the predicted behavior [13]. X-PRT supports and extends this capability by providing a level of abstraction that allows the modeler to reduce the work required to run a model using different architectures. X-PRT allows multiple architectures to have a common interface so that no task changes are required to run the task with a different architecture. The common interface defines what values the user needs to define in her task for that set of architectures to run. For architectures that are too different to support the same interface, X-PRT converts the task to run under the new architecture interface as much as possible, minimizing the number of changes necessary.

The current design fulfills the above requirement to provide an explicit representation of the architectural assumptions, and the capability to run the model under different architectural assumptions. The requirement to provide a representation above the code level is not addressed in the current version. There is an existing prototype of an architecture editor that must be tested, iterated, and integrated into the running version of X-PRT.

Design of Strategy (T5)

At present, the support for strategy specification primarily resides at the level of the CORE implementation; it is not yet completely supported in X-PRT. There are two fundamentally different ways that strategies may be specified in the model. First, X-PRT does support, in a limited fashion, making links between hierarchical task knowledge and specific strategies. This assumes the existence of a library of strategies from which complex tasks may be hierarchically composed. For each action defined within the task, the user is prompted to specify the specific strategy used (e.g. "click" as shown in Figure 1). Based on the strategy chosen, an appropriate set of parameters is presented. While this represents a significant advance over detailed coding, it still falls short of a general solution that systematically supports new strategy composition from architectural primitives.

One possibility that needs to be to explored, however, is exploiting the capabilities for hierarchical task specification to build the strategies as well. Even micro-strategies such move-and-click can be given meaningful hierarchical descriptions, with internal components that may be reusable. The challenge lies in making the cognitive architectural primitives accessible in the same way that the existing strategies like "click" are now accessible in the hierarchy editor (figure 1).

A second and novel way that CORE (and by extension X-PRT) supports strategy specification is by partially eliminating the *need* for strategy specification. This is possible because some detailed and complex aspects of behavioral control—for example, anticipatory eye movements or anticipatory hand positioning—emerge from CORE's automatic search for the optimal way to satisfy the task and architectural constraints. In short, users can focus on the higher task-level strategic composition ("perform these two task operations as quickly as possible in sequence"), and leave the CORE modeling engine to work out the precise details of how to stitch together the task-level operations. The details of this stitching-together are precisely what constitute multi-tasking executive strategies in a model like EPIC.

Design of Model Output (T6)

In terms of output visualization, X-PRT offers an improvement upon the traditional horizontally scrolling Gantt chart in the form a focus-plus-context style interface. The Gantt chart is presented in a 10% scale overview at the bottom of the screen (overview scrolls if necessary). In the overview area, the user can click and drag to move a rectangle that represents the view and can resize the overview area itself from the default 10% by increasing or decreasing its size with a standard pane resize bar.

This design provides a detailed view of the data and goes a small part of the way towards providing a high level overview with the context pane. However, the current overview is visual rather than conceptual. In practice, this requirement is not met nor is the requirement for a suggestion facility.

Design of Overhead Tasks (T7)

X-PRT separates and organizes each model component for the purposes of opening, editing, associating, copying and composing these components. Within X-PRT, the task is the central component around which other components are organized based on the understanding that the modeler's goal is to predict behavior for a particular task (projects can contain arbitrary numbers of tasks). For example, if a modeler is attempting to assess interfaces A and B, X-PRT manages the representation of both user interfaces and the linkage between the interfaces and the task. X-PRT does not rely on the user to remember this type of linkage, nor do files have to be explicitly manipulated in an external file system, as they are all stored within the context of a project.

To mitigate the problems with manual versioning, such as making a mistake and not being able to quickly restore the component's original state, X-PRT takes a snapshot of the model components that serve as inputs as well as the output of each run (currently in Gantt chart form). The modeler can view this run history with a single click and return to the earlier version of a model component with a second click.

Existing tools often rely on the command line to run models and generate behavior predictions as output. X-PRT provides a consistent, application-internal method of running models and generating behavioral predictions. The run pane allows the user to select a model, an associated architecture, and press the run button. X-PRT takes care of all the overhead necessary to make command line calls and pass arguments in order to run a model.

X-PRT is a standalone application installed by double clicking, as are commercial applications. It installs necessary elements transparently (e.g. the Sicstus Prolog environment). Standalone application implementation also allows the system to provide standard functionality like software update, which prompts users to download a new version.

This design addresses all the key overhead tasks observed to date, including file management, versioning, command line execution, and installation. However, overhead is a broad category and in extended use it is likely that users will uncover more of these types of challenges.

Design Process

Thus far, the discussion has covered the design of X-PRT itself and not the design process, which is also worth mentioning. The team has followed a standard HCI iterative design and test method. The first round of user testing is complete (number of users=5). Some design changes have been implemented while others are prioritized for development along with other work. The goal for this first round of user testing was to validate the X-PRT framework and IDE project metaphor. While the user tests uncovered many usability problems, they were primarily lower level problems like the fact that command line users expected to save explicitly rather than having the system do it automatically. Overall, the framework appeared to support the sample task of interface specification, task knowledge modification, model run, and output interpretation. As the work progresses, further iterations of the tool will be tested and metrics such as time on task will be tracked.

The Underlying Modeling Engine: CORE

In order to enable several of the features described in the design section such as easy description of operator skill and compositional reuse of tasks, X-PRT takes advantage of many theoretical advances of Cognitive Constraint Modeling (CCM) [7,13,21]. CCM represents a new approach to cognitive modeling that is characterized by three principles that distinguish it from existing simulation-based approach such as production system architectures. (1) Descriptions of behavior are derived via constraint satisfaction over explicitly declared architectural, task, and strategy constraints. One significant effect that this has on the practice of modeling is that architectural theory is uniformly encoded as explicit constraints that are inspectable and modifiable in the same way as task and strategy constraints. The generation of behavioral predictions is essentially an automated proof derivation from these explicit assumptions. (2) The details of behavioral control emerge in part from optimizing behavior with respect to objective functions intended to capture general strategic goals (e.g. go as fast as possible). A significant effect that this has on the practice of modeling is that the modeler is relieved of some of the difficult work of micro-programming detailed cognitive/perceptual/motor strategies for particular tasks. Rather, these details emerge from the behavior derivation process when it is guided by the explicitly declared objective goals. (3) The architectural building blocks are based on an ontology of resource-constrained cascaded processes. This provides a simple but extremely powerful way to construct complex behaviors from information processing primitives that specify the basic cognitive, perceptual, and motor processes and how they communicate. This ontology has been used to specify versions of cognitive architectures based on the Model Human Processor, ACT-R, and Epic [7,13,21]. All of these principles have been realized in an implemented computational modeling system called CORE (Constraint-based Optimizing Reasoning Engine), which uses constraint-satisfaction techniques that are guaranteed to yield optimal solutions.

EMPIRICAL DATA AND MODEL PREDICTIONS

Recently, CORE has been used to develop models of learning hierarchically structured tasks. Although it might seem that a modeling approach that generates optimal behavior would be suitable only for predicting highly skilled behavior, the approach can be applied quite naturally to predicting novice and intermediate skill levels as well. A key to this success is the discovery that hierarchical task structures not only provide a natural high level task description for modelers to work with, but are also the cognitive structures that mediate performance during task acquisition.

Consider first the role of task hierarchies as a high-level task specification. Goal hierarchies are ubiquitous in cognitive modeling and play a central role in GOMS-based methods. For many routine HCI tasks they are a natural specification because they directly reflect the task structure. See Figure 1, in the X-PRT design section, for part of a hierarchical description of a banking task.

The challenge in using such high-level specifications for developing millisecond level predictions of skilled performance is that humans organize their behavior in highly flexible ways that violate the putative encapsulation of the subtasks [10,13]. For example, humans exhibit *anticipatory* behaviors, where behaviors associated with a later subtask may intrude and intermix with behaviors associated with an earlier subtask. This kind of flexible scheduling is characteristic of human skill, but is at odds with traditional notions of encapsulated, reusable components.

How does the CCM approach, and CORE in particular, solve this problem? The answer is that CORE does not "execute" the task hierarchy. Rather, there is first a transformation of the hierarchy into independent task and strategy constraints, which, together with architectural constraints plus the optimizing constraint satisfaction, naturally yield the required flexible behavior. In this way, the hierarchy is taken as an abstract specification of task knowledge, rather than a control structure.

It is possible, however, to build CORE models that *do* treat the hierarchy as a control structure – in particular, as a structured memory that is retrieved piece-by-piece to control behavior. A natural assumption is that this kind of memory retrieval guides performance early in task acquisition, and skilled behavior emerges in part as a function of gradually eliminating these explicit memory retrievals [1]. ACT-R's production compilation mechanism provides a detailed process model of how this learning might take place. CORE is able to abstractly capture this kind of skill acquisition with two simple additions to the highly skilled model. First, memory retrieval processes are added to access the task knowledge, and behavior is contingent upon the results of these retrievals. Second, learning is modeled in an abstract way by simply flattening the task hierarchy. This is the mechanism that supports manipulation of operator skill level (T2) as discussed is the design section above. In all cases, behavior is generated in the same way via constraint satisfaction.

There are two major qualitative empirical predictions that this model makes. First, early in practice the hierarchical memory retrievals will serve as barriers to flexible scheduling of behaviors; thus, anticipatory behaviors will only emerge as these retrievals are eliminated. Second, early in practice the hierarchical memory retrievals will increase response latencies, and more specifically, will increase latencies as a direct function of hierarchy depth. This qualitative pattern is shown in Figure 3, which plots the model's reaction times to button presses in a mouse-driven ATM banking task.

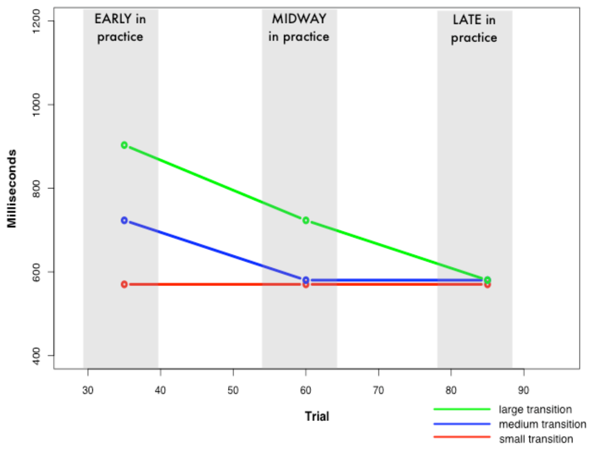


Figure 3: Model predicted times for button presses in an ATM task [13].

The task steps (in this case button presses) are grouped into large, medium, and small hierarchical transition, reflecting the amount of memory retrieval required to access the relevant bit of task knowledge to press the key. Early in practice, the operator must traverse the entire hierarchy as specified by the modeler. Midway through practice, assuming a four-tier hierarchy, the retrievals associated with third or lowest set of non-leaf nodes of the hierarchy are removed. Late in practice the second level of the hierarchy is removed and the operator represents what were once individual sub-goals as a single goal. Figure 3 shows this collapse, so that the end point converges on skilled behavior in which the hierarchical structure is no longer evident.

Recently, Lewis, Vera, and Howes [13] have developed an empirical paradigm that tests this qualitative prediction, by varying hierarchical structure (via instruction) while carefully controlling for other aspects of the task and interface. Results from an initial set of experiments reveal not just a single smooth learning curve for the overall task performance, but separate learning curves as a function of hierarchical task boundaries (Figure 4) [13]. Major subtask boundaries (at the start of major new subtasks) produced longer reaction times than minor boundaries within the lowest-level task groupings, and so on, as predicted by the model. In short, task hierarchies are not just convenient high-level notations but psychologically real control constructs which guide behavior during task acquisition. Unlike the model, there appears to be some continued effect of the task hierarchy quite late in practice, and the basis of this effect is currently being explored.

This is similar to the effects of production compilation in ACT-R (an ACT-R model of this task has been developed), but the approach does not depend on modeling the detailed processes of learning, and it makes no representational assumptions beyond that of the task hierarchy. The method will initially be restricted to modeling the effects of learning the task structure-it will not, for example, be able to model attentional learning (where to look for things on the interface). It is therefore restricted in scope, but represents an important step forward in incorporating learning predictions into easy-to-use modeling tools.

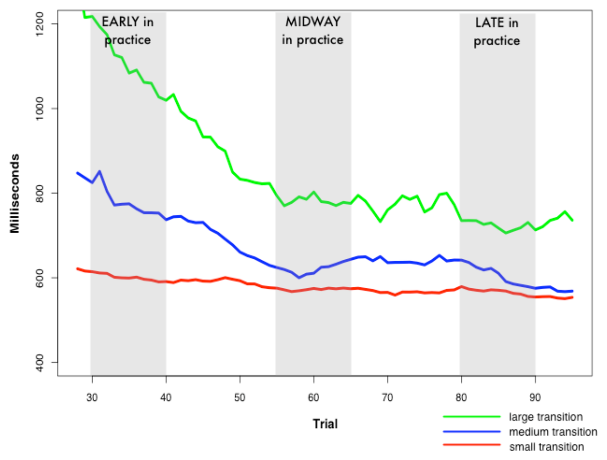


Figure 4: Reaction times for button presses an ATM task [13].

REFLECTIONS AND COMPARISONS

X-PRT represents a new approach to supporting cognitive modeling resulting from a combination of systematic user-centered design with recent developments in cognitive modeling based on constraint-satisfaction as implemented in CORE. It is worth summarizing here what we believe are its important features, highlighting what it has in common with other approaches and what makes it distinctive.

1. X-PRT was designed from the ground-up to support a comprehensive set of modeling activities in an integrated fashion. While it shares with both high-level languages (such as ACT-Simple, TAQL [18,22] and modeling toolkits (such as CogTool and user modeling design tool [9,16]) the goal of making modeling easier, we believe no other tool addresses the range of modeling activities represented in Table 1.
2. X-PRT provides rudimentary support for the easy generation of models at multiple skill levels from a single task specification. The ability to model at multiple skill levels is a feature of learning architectures such as ACT-R, but these approaches require considerable expertise in the details of the architectures and learning mechanism. Furthermore, none of the high-level languages that compile into ACT-R or Soar code systematically support learning. We believe that X-PRT and CORE represent an important and novel step toward developing tools that make the application of learning theory accessible to a broader user community.
3. X-PRT shares with GOMS-based approaches (and nearly all high-level modeling languages) the use and benefits of hierarchical task descriptions. But hierarchies play an even more extensive role in X-PRT and CORE: they are both the method of task description and compositional reuse *and* an important part of the cognitive theory that supports the generation of behavior at multiple skill levels. This is in large part what makes the learning model in X-PRT and CORE powerful and usable: it exploits the existing natural hierarchical specifications.
4. Finally, perhaps the most unique feature of X-PRT modeling is that some complex aspects of behavioral control (e.g., micro-strategies for making anticipatory movements) emerge *automatically* from the search for the most adaptive behavior given the posited objective function, combined with task, architecture, and partial strategic constraints. We believe this is a theoretical advance in cognitive modeling, but it is also a usability advance, because it reduces the burden on the modeler for strategy specification.

FUTURE WORK

Although we are clearly in the beginning stages of this project, the initial results are promising: X-PRT supports the rapid development of detailed models of routine GUI interactions at multiple skill levels, based on a single high-level hierarchical task specification. One of the early discoveries in this project is the surprisingly powerful set of roles that hierarchies play: they serve as both a natural task specification and a model of the cognitive structures that mediate task acquisition. This dual role is directly supported by CORE, and made easily accessible by X-PRT.

There is significant work remaining on both the systematic evaluation of the tool (and comparison to existing modeling tools), and extending the functionality with respect to tasks T1-T7 identified earlier. This work highlights just one interesting future direction that bears on learning models. As discussed earlier, skill level (T2) is currently set at the project (i.e. model) level rather than at the individual subtask level, but it is clear that most tasks of applied interest will be heterogeneous mixtures of routine and novel components. The current plan is to again exploit the hierarchical task specifications by permitting modelers to specify skill parameters for any individual subtask in the overall task, as well as permitting modelers to specify performance parameters associated with cross-cutting skills such mouse-based target tracking. The development of X-PRT along these lines will continue to push the state-of-the-art in both basic research and applied cognitive modeling.

REFERENCES

1. Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (in press). An integrated theory of mind. *Psychological Review*.
2. Baskin, J. D., John, B. E. (1998). Comparison of GOMS Analysis Methods Late Breaking Results: Ubiquitous Usability Engineering. *Proceedings of Conference on Human Factors in Computing Systems*, v.2 p.261-262.
3. Beyer, H., Holtzblatt, K. Contextual Design. *Contextual Design: A Customer-Centered Approach to Systems Designs*, Morgan Kaufman, 1998.
4. Card, S. K., Moran, T.P. and Newell, A. The Psychology of Human-Computer Interaction. Lawrence Erlbaum Associates, Hillsdale, NJ, USA (1983).
5. Gong, R. (1993). Validating and refining the GOMS model methodology for software user interface design and evaluation. Ph.D. dissertation, University of Michigan.
6. Gray, W. D., John, B. E., & Atwood, M. E. (1993). Project Ernestine: Validating GOMS for predicting and explaining real-world task performance. *Human Computer Interaction*., *8*(3), 237-309.
7. Howes, A., Vera, A. H., Lewis, R. L., and McCurdy, M. (2004). Cognitive constraint modeling: A formal approach to supporting reasoning about behavior. *Proceedings of the Twenty Fifth Conference of the Cognitive Science Society,* Chicago, IL.
8. John, B., E., Kieras, D., E. (1996). The GOMS family of user interface analysis techniques: comparison and contrast. ACM *Transactions on Computer-Human Interaction*, Volume 3, Issue 4 pg. 320 – 351.
9. John, B. E., Prevas, K., Salvucci, D. D., Koedinger, K. (2004). Predictive Human Performance Modeling Made Easy. *Proceedings of the conference on Human factors in computing systems*, Vienna, Austria.
10. John, B. E., Vera, A., Matessa, M., Freed, M., Remington, R., (2002). Automating CPM-GOMS. *Proceedings of the Conference on Human factors in computing systems*, Minneapolis, MN.
11. Kieras, D. E., Wood, S. D., Abotel, K., and Hornof, A. (1995). GLEAN: A Computer-Based Tool for Rapid GOMS Model Usability Evaluation of User Interface. *Proceedings of the ACM Symposium on User Interface Software and Technology*, ACM Press, pg 91-100.
12. Koedinger, K. R., Aleven, V., Heffernan, N., McLaren, B. M., and Hockenberry, M. (2004). Opening the Door to Non-Programmers: Authoring Intelligent Tutor Behavior by Demonstration. *Proceeding of Intelligent Tutoring Systems Conference*, Maceio, Brazil.
13. Lewis, R.L., Vera, A. H., and Howes, A. (2004). A constraint-based approach to understanding the composition of skill. *Proceedings of the Sixth International Conference on Cognitive Modeling,* Pittsburgh, PA.
14. Matessa, M. (2004) An ACT-R Framework for Interleaving Templates of Human Behavior. *Proceedings of the Twenty-sixth Annual Conference of the Cognitive Science Society*, Chicago IL.
15. Meyer, D. E., & Kieras, D. E. (1997). A computational theory of executive cognitive processes and multiple-task performance: Part 1. Basic mechanisms. *Psychological Review*, 104, 3–65.
16. Ritter, F. E., Van Rooy, D., & St. Amant, R. (2002). A user modeling design tool based on a cognitive architecture for comparing interfaces. *Proceedings of the 4th International Conference on Computer-Aided Design of User Interfaces* 111-118. Dordrecht, NL.
17. Ritter, F. E., & Young, R. M. (2001). Embodied models as simulated users: Introduction to this special issue on using cognitive models to improve interface design. *International Journal of Human-Computer Studies*, 55, 1-14.
18. Salvucci, D. D., & Lee, F. J. (2003). Simple cognitive modeling in a complex cognitive architecture. *Proceedings of the conference on Human Factors in Computing Systems* (pg. 265-272). New York, NY.
19. St. Amant, R., & Ritter, F. E. (2004). Automated GOMS to ACT-R model generation. In submitted to the *International Conference on Cognitive Modeling*. 26-31. Mahwah, NJ: Lawrence Erlbaum.
20. Taatgen, N.A. & Lee, F.J. (2003). Production Compilation: A simple mechanism to model Complex Skill Acquisition. *Human Factors*, 45(1), 61-76.
21. Vera, A. H., Howes, A., McCurdy, M., and Lewis, R. L. (2004). A constraint satisfaction approach to predicting skilled interactive cognition. *Proceedings of the Conference on Human Factors in Computing Systems,* Vienna, Austria.
22. G.R. Yost, Taql: A Problem Space Tool for Expert-System Development, doctoral dissertation, Carnegie Mellon Univ., Pittsburgh, Pa., 1992.