In Proceedings of the Sixth International Conference on Cognitive Modeling. July 2004.
Mahwah, NJ: Lawrence Earlbaum. [Best applied paper prize]

Automated GOMS–to–ACT-R Model Generation

Robert St. Amant (stamant@cs.ncsu.edu)
Department of Computer Science, North Carolina State University
Raleigh, NC 27695 USA

Frank E. Ritter (ritter@ist.psu.edu)
School of Information Sciences and Technology, The Pennsylvania State University
University Park, PA 16802 USA

Abstract

We describe a system, G2A, that produces ACT-R models from GOMS models containing hierarchical methods, visual and memory stores, and control constructs. Because GOMS is a more abstract formalism than ACT-R, a single GOMS operator might be plausibly translated in different ways into ACT-R productions (e.g., a GOMS Look-for operator might be carried out by different visual search strategies in ACT-R). Given a GOMS model, G2A generates and evaluates alternative ACT-R models by systematically varying the mapping of GOMS operators to ACT-R productions. In experiments with a text editing task, G2A produces ACT-R models with predictions that are within 5% of GOMS model predictions. In the same domain, G2A also generates ACT-R models that give good predictions of overall task duration for actual users (within 2% error), though the models are much less accurate at a detailed level.

Introduction

A recent trend has appeared in cognitive modeling for human-computer interaction: researchers are making the capabilities of cognitive architectures much more accessible to developers and designers who lack modeling expertise (Ritter et al., 2003). Salvucci and Lee's ACT-Simple (2003) automatically generates ACT-R models (Anderson et al., in press) from a language similar to KLM-GOMS. Techniques developed by John et al. (2004) automatically build ACT-R models (via KLM and ACT-Simple) from actions demonstrated by interface designers. A system by St. Amant, Horton, and Ritter (2004), in the domain of cell phone modeling, partially automates the generation of ACT-R models from the specification of a keypad and menu hierarchy.

The inspiration for the work described in this paper is ACT-Simple and to a lesser extent TAQL (Yost, 1993). ACT-Simple demonstrates the feasibility of automatic translation of a high-level, GOMS-like specification language into detailed ACT-R models—an exciting and significant achievement. That said, the ACT-Simple translation process and its results have limitations. Each operator in the source model is transformed into one or two ACT-R productions in a static translation process. These productions are chained together in a mostly linear fashion. The productions make almost no use of the environment (e.g., all visual processing is represented as shifts in attention between two fixed locations.) Essentially, the generated ACT-R models have relatively little that is "cognitive" about them: there is no input of information from the environment, there are no memory retrievals for information processing, and there is no decision-making. This is not to slight ACT-Simple—these constraints are mainly dictated by KLM-GOMS, which is intended to provide a simple, largely external description of performance.

Our research explores the same basic problem addressed by ACT-Simple, that of translating high-level specifications into more detailed cognitive models, but shifts the focus from simplicity to representational power. We have built a system, G2A, that translates models from GOMSL into ACT-R. GOMSL is an abstract but rich modeling language within the GOMS family (Kieras, 1999). Going beyond the capabilities of KLM-GOMS, GOMSL allows representation of mental objects, working memory storage, primitive internal and external operators, composite methods, and various flow-of-control constructs. G2A supports all of these capabilities in its translation process. Our goal is to allow the expression of models as simple as KLM-GOMS models, while letting modelers add more complexity if it is needed.

We chose ACT-R as a target language (e.g., rather than EPIC, Kieras & Meyer, 1997) to leverage existing work on model translation. We chose GOMSL as a source language because of a detailed manual documenting its structure and use (Kieras, 1999). G2A has been tested using 34 examples taken from the GOMSL manual (mostly verbatim, but with a few syntax corrections). The most complex GOMSL model given takes up about four pages—190 lines of code containing 15 mental object definitions, 11 methods, and two sets of selection rules. To match this model, the translation process generates an ACT-R model of five chunk definitions, 23 chunks, 79 productions, and various auxiliary constructs (over 1500 lines of formatted model code). The ACT-R model's predictions closely match the predictions of the source GOMSL model. Using the same GOMSL model as a starting point, G2A can also automatically generate different models that give reasonable predictions of actual user performance, based on different assumptions to match different evaluation criteria.

The goal of developing G2A was not simply to translate GOMS models to ACT-R models, which turned out to be demanding in programming terms but conceptually relatively simple.[1] Our work provides insight into some

[1] G2A is available at www4.ncsu.edu/~stamant/G2A.

of the more interesting problems faced by the cognitive modeling community today. How does one go about extending simple models into more detailed ones? More generally, what is the space of models appropriate for a given task and performance data? (With G2A, the "degrees of freedom" are related to the flexible mapping of actions to productions, visual processing, and environment specification.) How can different, potentially complex models be integrated? (We argue that integrating high-level specifications is often easier than integrating detailed models themselves.) How can ACT-R modeling concepts be made accessible to novice modelers already familiar with GOMS formalisms? (G2A provides an explicit computational account of how some common ACT-R idioms are used to represent specific behaviors.) Finally, even if the feasibility of generating ACT-R models from GOMSL models is not an unexpected result, it is worthwhile to see such an expectation verified.

GOMSL translation

GOMSL syntax is comparable to that of a procedural programming language, as shown in Figure 1. For reasons of space, in our discussion we will assume basic familiarity with GOMS and ACT-R concepts, and we will not explicitly treat architectural differences; some details are also elided. G2A begins by parsing a GOMSL model into an intermediate representation[2] appropriate for further translation as described below.

Methods and flow of control. GOMSL methods are sequences of steps, where each step can contain one or more operators. For example, in Figure 1 the first step in the method Edit Document is the Store operator. G2A follows ACT-Simple in its representation of control: every production contains a condition test of the ?state slot of a Goal chunk and an action to update that slot. Sequences of operators are translated into productions that are ordered by appropriately sequenced ?state values. Step execution is not always sequential within a method; Decide statements support branching based on predicate tests Goto allows arbitrary transfer of control to labeled steps. These forms are translated to appropriately set ?state values in the relevant productions.

Each GOMSL method satisfies a goal; instead of calling operators, a method can "invoke" another method with an Accomplish-goal (AG) statement. This flow of control translates to the creation of a subgoal that is pushed on the goal stack.[3] When a method completes, it ends with a Return-with-goal-accomplished (RGA) statement; this becomes a pop action in ACT-R. Selection rules, which govern the choice of methods when multiple methods can accomplish a given goal, are translated similarly to Decide statements.

[2]For this we used an off-the-shelf LALR parser, written by Mark Johnson, from the online AI Repository at CMU.

[3]The models produced by G2A include a small number of ACT-R 4 constructs (Anderson & Lebiere, 1998) that are deprecated in ACT-R 5 (Anderson et al., in press); we are working to make the system ACT-R 5/6 compliant.

```
LTM_item: Cut_Command
   Name is Cut.
   Containing_Menu is Edit.
   Menu_Item_Label is Cut.
Method_for_goal: Edit Document
   Step. Store First under <current_task_name>.
   Step Check_for_done.
   Decide: If <current_task_name> is None, Then
      Delete <current_task>; Delete <current_task_name>;
      Return_with_goal_accomplished.
   Step. Get_task_item_whose Name is <current_task_name>
      and_store_under <current_task>.
   Step. Accomplish_goal: Perform Unit_task.
   Step. Store Next of <current_task>
      under <current_task_name>;
      Goto Check_for_done.
```

Figure 1: GOMSL object and method (partial).

Objects and working memory. GOMSL supports declarative object representations, including objects in long term memory (LTM-items), objects available through visual processing (Visual-items), and task descriptions (Task-items). Each object consists of a named collection of property–value pairs. GOMSL objects are translated directly into ACT-R chunks. Each generated chunk is given a unique identifier for reference.

Objects in GOMSL are brought into working memory as named tags, such as <current_task_name> in Figure 1. Properties of an object stored in a tag are also immediately available for processing. G2A captures this functionality via a slot in the Goal chunk for each tag in a GOMSL program. A change to the contents of a tag by a GOMSL operator translates to an update of the corresponding slot in a Goal chunk.

For convenience, GOMSL allows method arguments (pseudo-parameters) to be defined. These are implemented as implicit Store operators, which record values in global tags for the information being passed between methods in AG forms. When a production creates a subgoal, it copies tag values from the current goal to the subgoal before pushing it on the stack.

Operators. GOMSL defines a number of primitive operators that carry the basic load of modeling performance. Aside from the control-flow forms mentioned above, G2A handles Keystroke, Type-in, Click, Double-click, Hold-down, Release, Point-to, Home-to, Speak, Look-for-object-whose, Get-task-item-whose, Store, Delete, Recall-LTM-item-whose, Verify, and Think-of. These encompass all GOMSL operators except Wait-for-object-whose and Wait-for-auditory-object-whose, which we expect to add soon.

For most of these operators, G2A contains one G translation and one or more A translations:

- A-type translations use converted AC-R idioms to generate productions that perform activities equivalent to the GOMSL operator. For example, Point-to expands to a Hand-to-mouse production followed by a Move-cursor production, imposing appropriate conditions on the manual state.

- *G*-type translations use GOMSL-specified duration values (e.g., a Keystroke requires 280 ms). In most cases they generate the same productions as *A*-type constructs.

Think-of is one exception to this scheme; it only has a *G* translation, with the same fixed duration of other mental and visual operators, 1,200 ms. The creation of an ACT-R model using G2A requires choosing a specific *A* or *G* translation for each of the operators in a source GOMSL model. The more complex *A* translations are as follows.

Double-click/A, Hold-down/A, and Release/A are translated into ACT-R Mouse-click actions—ACT-R lacks a direct implementation of these low-level mouse operations as atomic actions. A Double-click/single-action, with duration comparable to a *G*-type translation, provides a partial workaround.

Point-to/A, as mentioned above, expands to two productions, taking a target as an argument. This target can be either a literal symbol, an object stored in a tag, or a literal or object stored in the property of a tag. The default G2A translation process generates a random location associated with the object or literal, which is stored for later pointing actions to the same target. This location is used in the Move-cursor production. (Numerical screen coordinates can be used if provided in the GOMSL model, but this requires specialized task-specific code.)

Recall-LTM-item-whose/A and Get-task-item-whose/A both take a list of predicates and a store tag as arguments, the first operator returning an LTM-item, the second a Task-item. The result of the translation is a production in which the identifier of a retrieved object that meets the predicate tests is stored in the given tag.
A predicate is a comparison between operands. Comparisons in G2A are limited to tests of equality or non-equality of symbols. The first operand to the comparison is a property of the object to be retrieved. The second is either a literal, a literal stored in a tag, or a property associated with an object stored in a tag. In the first two cases, the production that is generated includes a pattern that makes a direct comparison between the object property and the literal or tag. In the third case, an additional buffer is added to the production that allows the retrieval of the chunk corresponding to the identifier stored in the tag and access to its properties. Predicates are processed recursively, one at a time, until the list of predicates is exhausted. The last action carried out by the production is to store the identifier of the retrieved object in the tag slot of the current Goal chunk. Figure 2 shows a typical result for a retrieval with a single predicate.

Look-for-object-whose/search takes a list of predicates and a store tag as arguments. The translation generates productions that search through the set of visual objects present until the desired object is found. These productions use the standard ACT-R idiom of find/attend/harvest for visual acquisition. The harvest production is augmented via the same predicate processing as with Recall-LTM-item-whose.
If the visual module could be guaranteed to find an object that meets all of the predicate tests on its first try, then the translation would be similar to Recall-LTM-item-whose. Lacking this guarantee, however, productions must be generated for each of the predicates that may fail. In case of failure, these secondary productions cause the find production to be fired again, to visit another not-yet-attended visual location. When a visual object is found that passes all the predicate tests, its identifier is recorded in the tag slot of the current Goal chunk.

```
;;; Get-task-item-whose ((is name [task-name])) [current-task]
(p production86
   =goal>
      isa                 goal
      [task-name]         =[task-name]
      ?state              edit-document-3
   =match88>
      isa                 task-item
      ?id                 =temp87
      name                =[task-name]
==>
   =goal>
      [current-task]      =temp87
      ?state              edit-document-4)
```

Figure 2: Sample generated production.

Look-for-object-whose/direct is another *A*-type translation. In many situations, an exhaustive visual search is not carried out, because the user either knows the location of a visual object already, or because the predicates rely on pop-out properties of the object. For such situations, the translation generates a production that simply moves the attention to a specific visual location (randomly generated by default, but recorded for later use, as with Point-to.)

Store generates a production that records a given value (either a literal value, the contents of a tag, or the property the contents of a tag) in a target tag. There is one special case for translation of Store (and Delete, below), as dictated by GOMSL. When these operators occur in the same step as other operators, their action is merged with the other operators, which means that they have no independent duration.

Delete generates a production that sets the contents of a given tag to a null (Empty) value.

Verify/last generates a production that moves attention to the visual location that was most recently visited by a Look-for-object-whose or a Point-to operator. Verify/none generates no production, taking no time.

The translations we have defined have minor limitations, which we are working to remove. We have also begun to examine different sets of translations; for example, we have reproduced several of the models of Salvucci and Lee (2003), using the ACT-Simple modeling language rather than GOMSL as a source language.

Evaluation

Comparing GOMSL and ACT-R models. We can directly demonstrate the accuracy of the GOMSL to ACT-R translation by choosing a GOMSL model, translating it using G translations for each of its methods, and running a comparison of execution times. To simplify our comparison here (and in the rest of this section), ACT-R models are executed using default values for all parameters, with no base-level learning.

For our comparison we used the Edit Document model described earlier. This GOMSL model includes four top-level methods describing activities in a mouse-based word processing environment, executed sequentially with no delays between them. Copy word involves selecting and copying a single word and pasting it elsewhere in the document. Copy arbitrary is a comparable task for a sequence of words. Delete word involves selecting a word and pressing the delete key. Move arbitrary is similar to Copy arbitrary. All editing actions are executed through selections from a pull-down menu. These lower-level activities are accomplished by the methods given in the Method column of Table 1.

The GOMSL column shows the durations per method as given by the GOMSL model (Kieras, 1999, p. 58), and the G2A/M_g column shows the corresponding values for the ACT-R model that G2A generated, using a G-type translation for every GOMSL operator, averaged over 20 runs. The numbers are very close, $r = 0.999$, with small discrepancies due to parallel execution of visual and motor actions in ACT-R, plus the automatic addition to the ACT-R model of hand movements between keyboard and mouse that are implicit in the GOMSL model (specifically, in Select-word and Select-insertion-point). On average, the predictions of the ACT-R translations are within 5% of the GOMSL method durations, and the overall task duration prediction is within 1%. This is a straightforward but non-trivial result: it shows that the hierarchical relationships between GOMSL methods, flow of control within the methods, and transfer of information between methods are captured in executing ACT-R productions, even if production durations are fixed.

Predicting user performance. An obvious next step, staying within the same task domain, is to see how well these models predict actual user performance. If we time users carrying out the Edit Document task in a standard word processing application, we find that it takes 20 seconds or less. This value is far from the predictions of the GOMSL or M_g model. If we believe that these models, despite their performance, correctly represent the basic structure of the task, we can develop explanations for the observed behavior in a systematic way with G2A.

The translation alternatives in the previous section can be thought of as ACT-R idioms for representing particular activities. We can then view G2A as facing the same issues as a human cognitive modeler in constructing an accurate model. Though G2A lacks actual domain knowledge, it can rely on the GOMSL model of the task, and in cases involving analysis of an existing system, G2A may have access to user data. In other words, G2A can carry out a model-fitting process, exploring different elaborations of the GOMSL task structure.

Table 1: Model predictions for the Edit Document task.

Method	GOMSL	G2A/M_g	Error
Select-insertion-point	3.60	4.14	15%
Select-word	4.40	4.71	7%
Erase-text	6.40	7.25	13%
Select-arbitrary-text	6.70	6.52	3%
Issue-command	9.05	8.55	6%
Paste-selection	12.80	12.84	0%
Copy-selection	14.85	14.36	3%
Cut-selection	16.20	15.79	3%
Copy-text	28.85	28.55	1%
Move-text	30.80	29.98	3%
Edit-document	101.20	100.63	1%

To do this, G2A treats the alternative translations of GOMSL operators as a search space. By varying the translations that are activated in generating ACT-R models, G2A explores this search space, using hill-climbing to identify the best translation. In hill-climbing, an evaluation function f is applied to a current state s. The function f is applied to each of the states neighboring s, and if any of these successor states produces a better value, the best of them becomes the new current state. The process repeats until no successor state produces an improvement in f.

A state for the G2A search is a set of translations, each chosen from the set of possible translations for a single operator: Look-for-object-whose/direct + Verify/none + Click/G + ... Each translation set produces a unique model. Successors to a translation set are those that differ in the translation of one GOMSL operator. For f, G2A executes the model corresponding to the current translation set 20 times, collects predictions of the total duration of the Edit Document task, and computes the difference from a target duration. (Notice that we cannot directly compare method execution times, because method boundaries are only implicit in user behavior—all we have access to is external events.)

To test this idea, we conducted a small user study. We implemented a simple, instrumented text editing application roughly equivalent to Microsoft Notepad that supports the Edit Document tasks. A pilot subject ran ten trials of four tasks (Copy word, Copy arbitrary, Delete word, and Move arbitrary) that the Edit Document model also performed. We then ran six users through the same procedure.

We used the mean total duration of the pilot user's trials as a target for the G2A search. G2A evaluated about 50 models in its search. The best model found, M_{td}, relies on A translations for all mental and visual activities and most motor actions, performs no visual search, directing attention to known locations, and does no verification. Comparing M_{td}'s predictions of total duration with the performance of the six users of the study shows good results. The grand mean duration over the

six users was 18.85 seconds, with a standard deviation of 1.82 over the six user means. M_{td}'s prediction of 19.18 seconds gives an error of about 2%.

This result is also worth noting: the ACT-R model automatically produced by G2A reflects straightforward decisons that a human modeler might make to capture human performance in this domain, and its prediction is much more accurate (and obtained with no further modeling effort) than that of the original GOMSL model. M_{td} has about the same predictive power as comparable generated ACT-R models described in the literature (John et al., 2004; Salvucci & Lee, 2003).

Refining a model for more detail. M_{td} has obvious built-in limitations. In particular, the duration of its mouse movements and visual attention shifts are based on random locations, rather than an actual environment, and thus its detailed predictions of user actions are unlikely to be reliable. Using the data from the pilot subject, however, it is possible to carry out a more detailed analysis by refining M_{td}.

We defined a new search evaluation function that measures the intervals between keyboard/mouse events (i.e., stripping out the cumulative duration to each event), and computes the mean squared error with the corresponding interval durations for our pilot user. We also altered the model generation process to use actual locations of objects, as measured in our instrumented application, rather than random values. The best model produced by the search, M_{id}, is almost identical to M_{td}: it does no visual search and no verification, and it relies on A translations for all mental operators. It varies only in that Double-click is implemented by a single action translation, rather than a sequence of mouse clicks.

M_{id}'s prediction of overall task duration is about the same as that of M_{td}, 19.03 seconds. (In general, we would not expect models generated by different search evaluation functions to be so similar, but in this domain it appears that several plausible models are clustered in the same region of performance.) The more detailed predictions of M_{id} are shown in Table 2, along with the mean intervals between mouse click and keystroke events for the six users in the study. Unfortunately, these predictions of M_{id} are not as good as we might have hoped for. The correlation between user intervals and M_{id} predictions is 0.754, and the average error in the predictions is 35%. There is no obvious pattern in the error values shown in Table 2, but if the two largest could be reduced, this would bring the model's error down to 24%—still high, but more respectable.

There are two ways we might improve M_{id}. First, as discussed in the previous section, some low-level mouse actions, such as Double-click, Hold-down, and Release, have no direct representations in ACT-R; our substitutions are inevitably inaccurate. We expect that once such actions are developed, validated, and added to the ACT-R architecture, we will see better results. Second, it is possible that the structure imposed on ACT-R productions by GOMSL forms (e.g., GOMSL methods entail bookkeeping overhead due to AG and RGA operators) degrades the accuracy of predictions of a generated model, compared with the predictions of a native ACT-R model. Translations that combine or parallelize GOMSL operators more effectively may help.

Table 2: User performance on the Edit Document task.

Action	User	M_{id}	Error
Double-click	0.25	0.20	20% (0.05 s)
Select Copy	1.48	1.40	5% (0.08 s)
Set insertion	0.53	1.01	91% (0.48 s)
Select Paste	1.28	1.18	8% (0.10 s)
Select sentence	2.16	2.02	6% (0.14 s)
Select Copy	1.81	1.16	36% (0.65 s)
Set insertion	0.76	1.01	33% (0.25 s)
Select Paste	1.62	1.18	27% (0.44 s)
Double-click	1.35	1.74	29% (0.39 s)
Press Delete	0.64	1.39	117% (0.75 s)
Select sentence	2.92	2.00	32% (0.92 s)
Select Cut	1.82	1.13	38% (0.69 s)
Set insertion	0.74	0.96	30% (0.22 s)
Select Paste	1.49	1.18	21% (0.31 s)

Despite the limited predictive power of M_{id} at a detailed level, it constitutes another interesting result. M_{id} was produced as a refinement of M_{td}, in a largely automated process that called for only the effort of supplying environment information. Otherwise the process depended mainly on the GOMSL model for domain knowledge and pilot user data to guide model construction. No existing work on model translation has tested model predictions at the same level of detail as M_{id}, and given this lack of experience we find its performance adequate.

Limitations. One important limitation in our evaluation of M_{td} is that it is among the fastest models in the search space explored by G2A. In other words, if we had chosen an evaluation function that simply minimized execution duration, our results would have been about the same. Our combination of task and subject pool is thus less informative than it might be, because our target performance lies at an extreme, allowing little flexibility in model structure.

We have also neglected one of the most interesting opportunities for G2A: exploring possible translations of the Think-of operator. John et al. (2004) have shown that the placement of mental operators in a keystroke-level model, a difficult task for human modelers, can be automated. An elaboration of our search process might insert ACT-R productions into a model to better match user durations, but determining exactly what these productions should do is a much harder problem that requires domain-dependent reasoning. This is an open issue for future research.

Discussion

G2A is a work in progress, but it is sufficient to demonstrate the promise of the approach. We expect that with further development and refinement by ourselves and others, G2A can benefit the research community.

There are several avenues worth pursuing. One is bracketing (Kieras & Meyer, 2000), a tactic that can guide the iterative modeling process. A fastest possible model and a slowest reasonable model (based on optional or inefficiently executed task components) are derived from a base strategy; results can then help a designer decide, for example, whether the performance demands of a system are likely to outstrip human capabilities. G2A could contribute to bracketing by generating different models given a base strategy represented as a GOMSL model. If the fastest and slowest models are within the search space of G2A's translations—an important consideration, but a reasonable expectation in some domains—then these might be identified automatically. Work along these lines could lead to a useful extension to G2A, in the form of translations of higher-level GOMSL methods as well as primitive operators.

An idea closely related to bracketing is that modelers might associate classes of translations with different classes of users. In the same way that a programmer sets compiler flags to generate faster or safer executable code, sets of translations can be identified with expert or novice behavior. These translations might go further than the ones described above, varying internal ACT-R parameters rather than only model structure. In such cases, the expert modeler would simply bypass G2A search to settle directly on a specific model.

Another possible benefit is the use of G2A's abstractions in teaching modeling concepts to computer scientists. Operator translations generate sequences of productions comparable to those in ACT-R tutorial modules, though G2A's productions tend to be much more specialized. The abstraction of productions into methods can be thought of as providing a simple, high-level programming language for model components. Productions from different methods do not interact with each other except at the beginning and end of their translation sequences; information is passed through named, global slots. These factors make processing in a generated model much easier to understand. G2A might be extended to allow user queries (e.g., "What does an exhaustive visual search look like in ACT-R?") by showing how specific translations are performed.

A final potential benefit relates to model size and reuse. The largest models that G2A generates for the Edit Document task contain about 100 productions, and nothing prevents us from generating much larger models for more complex tasks. It is time-consuming and error-prone to generate models of this size by hand even if the modeled behavior is completely understood. Adding methods to a GOMSL model and examining their translation is straightforward in G2A and easier, in our experience, than writing ACT-R productions directly. If a task appears to be too large to be handled with conventional ACT-R development tools, or if model requirements outgrow GOMS (e.g., a model of visual processing or learning behavior may be needed), or if a high-level, procedural tasking language is required to control an extended experiment, then G2A offers a starting point.

Any potential benefits of G2A will involve tradeoffs that can only be determined by further research. For example, it may happen that abstract tuning of models via high-level compiler-like directives does not allow sufficient precision, but on the other hand it may provide more control. It may be that increasing model modularity, which makes larger models more feasible, involves a reduction in accuracy. This is suggested by our model refinement efforts, but it is not clear that this is a necessary implication of G2A processing. Still, we believe that G2A represents a promising approach. G2A advances the state of the art in cognitive model generation and points to several important areas for further research.

Acknowledgments

Thanks to Thomas Horton, who built the editing application and ran the user study. Thanks to Mike Byrne and two anonymous reviewers for suggesting significant improvements to this paper. This research was supported by NSF award IIS-0083281 and by ONR award N00014-02-1-0021. The information in this paper does not necessarily reflect the position or policies of the U.S. government, and no official endorsement should be inferred.

References

Anderson, J. R., Bothell, D., Byrne, M. D. & Lebiere, C. (in press). An integrated theory of the mind. *Psychological Review.*

Anderson, J., & Lebiere, C. (1998). *The Atomic Components of Thought.* LEA, Mahwah, NJ.

John, B. E., Prevas, K., Salvucci, D. D., & Koedinger, K. (2004). Predictive human performance modeling made easy. *Proceedings of CHI '04*, 455–462. ACM.

Kieras, D. E. (1999). *A guide to GOMS model usability evaluation using GOMSL and GLEAN3.* Technical Report, University of Michigan, Ann Arbor, MI.

Kieras, D. E., & Meyer, D. E., (1997). An overview of the EPIC architecture for cognition and performance with application to human-computer interaction, *Human-Computer Interaction*, *12*(4): 391-438.

Kieras, D. E., & Meyer, D. E., (2000). The role of cognitive task analysis in the application of predictive models of human performance. In *Cognitive Task Analysis*, Schraagen, J. M., Chipman, S. F., & Shalin, V. L. (eds.) Lawrence Erlbaum, Mahwah, NJ.

Ritter, F. E., Shadbolt, N. R., Elliman, D., Young, R., Gobet, F., & Baxter, G. D. (2003). *Techniques for modeling human performance in synthetic environments: A supplementary review.* Wright Patterson AFB: Human Systems Information Analysis Center. WWW: //iac.dtic.mil/hsiac/S-docs/SOAR-Jun03-Front.pdf

St. Amant, R., Horton, T. E., & Ritter, F. E. (2004). Cognitive modeling for cell phone menu evaluation. *Proceedings of CHI '04*, 343–350. ACM

Salvucci, D. D., & Lee, F. J. (2003). Simple cognitive modeling in a complex cognitive architecture. *Proceedings of CHI '03*, 265-272. ACM.

Yost, G. R. (1993). Acquiring knowledge in Soar. *IEEE Expert*, *8*(3), 26-34.