# Simple Cognitive Modeling in a Complex Cognitive Architecture

**Dario D. Salvucci**
Department of Computer Science
Drexel University
Philadelphia, PA 19104
+1 215 895 2674
salvucci@cs.drexel.edu

**Frank J. Lee**
Department of Cognitive Science
Rensselaer Polytechnic Institute
Troy, NY 12180
+1 518 276 4129
fjl@rpi.edu

## ABSTRACT
Cognitive modeling has evolved into a powerful tool for understanding and predicting user behavior. Higher-level modeling frameworks such as GOMS and its variants facilitate fast and easy model development but are sometimes limited in their ability to model detailed user behavior. Lower-level cognitive architectures such as EPIC, ACT-R, and Soar allow for greater precision and direct interaction with real-world systems but require significant modeling training and expertise. In this paper we present a modeling framework, ACT-Simple, that aims to combine the advantages of both approaches to cognitive modeling. ACT-Simple embodies a "compilation" approach in which a simple description language is compiled down to a core lower-level architecture (namely ACT-R). We present theoretical justification and empirical validation of the usefulness of the approach and framework.

### Keywords
Cognitive modeling, cognitive architectures, ACT-R.

## INTRODUCTION
Cognitive modeling of user behavior (or simply "user modeling") has emerged as a powerful technique for exploring how users interact with complex systems. For instance, recent investigations into the process of menu selection using detailed cognitive models [4, 8], have provided significant insights into how users coordinate eye and mouse movements for different types of menus. In addition to theoretical advances, cognitive modeling can be a useful and practical tool for in the development life-cycle of real-world systems. For example, the GOMS modeling framework and its variants [see 7, 10, 11] have been used successfully to predict user behavior and thereby speed up the process of testing and evaluating new interfaces [e.g., 7, 10]. In another example, integrated modeling of driving and secondary-task behavior (e.g., dialing a cell phone) has shown that modeling can help in evaluating interfaces with respect to their potential for driver distraction [20]. These and many other examples demonstrate both the theoretical and practical benefits of cognitive modeling for human-computer interaction.

### Levels of Modeling
Cognitive user models have typically addressed behavior at one of two levels of abstraction (or "grain sizes" of modeling). Many models have addressed behavior using higher-level frameworks that represent behavior as basic user actions such as moving a mouse or pressing a key. The GOMS framework and its variants [6, 10, 11] are by far the most popular tool for representing user behavior at this level. For instance, KLM-GOMS [6] provides a simple framework for describing expert behavior as linear sequence of steps, and its simplicity has benefited evaluations of straightforward interfaces and tasks—e.g., the telephone-operator task [7]. While easy to learn and use, such frameworks are limited to modeling expert behavior in simple tasks (excluding more complex GOMS frameworks like CPM-GOMS [see 11]) and say little about lower-level user behavior (e.g., eye movements).

As a different approach, some models have represented behavior using lower-level cognitive architectures that describe "atomic components" of behavior [e.g., 2, 16, 17, 18] with cognitive steps of roughly 50 ms and parallel perceptual and motor processes. Models developed within cognitive architectures comprise condition-action rules that gather perceptual knowledge, perform cognitive functions, and issue motor commands. For instance, in the menu models cited earlier [4, 8], investigators required a more detailed level of analysis to understand coordination of eye and mouse movements in menu selection, and thus turned to the increased power of the production-system cognitive architectures EPIC [17] and ACT-R [1]. The downside of the cognitive architecture approach is that developing detailed models typically requires significant training of the modeler, and even with this training, writing these models requires an order of magnitude more time than writing models in simple GOMS-like frameworks.

### Combining Levels of Modeling
In this paper we present a new framework, ACT-Simple, that combines the simplicity of higher-level frameworks with the power of lower-level cognitive architectures.

Salvucci, D. D., & Lee, F. J. (2003). Simple cognitive modeling in a complex cognitive architecture. To appear in *Human Factors in Computing Systems: CHI 2003 Conference Proceedings*. New York: ACM Press.

ACT-Simple is based on a "compilation approach" in which a model, comprising basic perceptual, motor, and cognitive commands (e.g., "look-at" or "press-key"), are translated into lower-level production rules in the ACT-R cognitive architecture [1]. We argue that the compilation approach provides several significant advantages over simply using a higher-level framework, namely that it facilitates theoretical consistency, inheritance of architectural features and constraints, model integration, and model refinement. We also demonstrate the accuracy of the ACT-Simple framework in the context of two sample application domains, namely entering database queries and dialing a cellular phone.

## THE COMPILATION APPROACH

Our approach to integrating levels of modeling involves compilation of a higher-level modeling framework to a lower-level cognitive architecture. The process starts with a high-level description of basic, GOMS-like actions of several hundred milliseconds (roughly 100-1500 ms) in duration—for instance, move the mouse to a new location, press a key, or even think for given duration (to be explained further in the next section). The process then compiles, or translates, each command to a corresponding representation in the low-level cognitive architecture (in our case, condition-action production rules). In essence, the compilation is very much analogous to the compilation of a higher-level programming language such as Java or C into lower-level assembly language.

Higher-level frameworks typically do not require any such compilation process—they can run in simulation directly (such as for NGOMSL [see 10]) or, even more simply, be translated directly into run times (such as for KLM-GOMS [6]). However, there are several significant benefits to compiling higher-level models to a lower-level architecture, as outlined in the following sections.

### *(1) Theoretical Consistency*

Under the vision of a truly "unified theory of cognition" [18], modeling at different levels, while perhaps each having distinct benefits and uses, should be consistent with levels above and below. For instance, the ACT-R cognitive architecture [1] formulates knowledge as condition-action production rules, but it has been demonstrated to be consistent with higher-level unit task descriptions [e.g., 17] as well as lower-level neural implementations [e.g., 15]. The compilation approach helps to maintain such consistency by using lower-level knowledge as much as possible. For example, KLM-GOMS assigns a time of 280 ms to pressing a key (although this time can vary depending on skill level). In the compilation approach, we compile the "press-key" command down to a lower-level architecture and allow the architecture to dictate the parameters of the action. To continue our example, if we compile "press-key" down to an analogous production rule in the ACT-R architecture, ACT-R (particularly the perceptual-motor extensions known as ACT-R/PM [5]) dictates how preparation of various hand and finger movements affect the time needed to press the given key. By allowing ACT-R to determine the reaction time, we constrain the higher-level framework to be more consistent with the (presumably well-tested) predictions of the lower-level architecture.

It may not be surprising that difficulties may arise in the realization of the compilation process—namely, that we are forced to specify certain aspects of the model which the higher-level description does not include. In the "press-key" example, the reaction time depends on the actual key pressed, but the higher-level model may not specify the key; thus, the compilation process may need to make a default assumption about the key—in some sense, performing the "average" action over all possible keypresses. We will make such assumptions for the ACT-Simple framework that clearly work for our validation studies; nevertheless, we recognize that these problems may in the future require detailed analytic and empirical study.

### *(2) Inheritance of Architecture*

One of the most important advantages of cognitive architectures is that they incorporate built-in parameters and limitations of human cognition and performance; in other words, any model developed in the architecture necessarily inherits the features of the architecture and thus becomes more psychologically plausible. By translating a higher-level model into a lower-level model in a cognitive architecture, the resulting lower-level model also automatically inherits these parameters and limitations, thus adding an significant amount of new predictive power to the model. For example, a simple KLM-GOMS model can provide a total task time for a given interface, but makes no predictions about learning and speed-up, memory failures, etc. By translating this model (or a similar one) to a model in an architecture such as ACT-R, Soar, or EPIC, the model immediately inherits performance parameters for common actions, thereby increasing theoretical consistency as we have described earlier. In addition, the model also inherits learning and performance mechanisms built into the architecture, and thus can now parallelize perceptual and motor processes with cognitive processes, learn and optimize performance, make errors in a human-like way, show individual differences in behavior, etc. Compiling down to a lower-level architecture allows for *a priori* predictions of behavior well beyond that of the simpler model.

### *(3) Model Integration*

As cognitive models become increasingly complex, there is a burgeoning need for developing models as modules and for integrating these "model modules" into full-scale models of large, complex tasks. To facilitate such integration, the models require a common descriptive language with which to interact and communicate, and the easiest way to guarantee such a common language is to ensure that all models reduce to a single common framework. The compilation process allows models written at different levels of description to interact easily by transforming the higher-level model to a lower-level one. The lower-level model that results from compiling a higher-level model can easily interact with other lower-level models that have been written directly in the cognitive architecture or have themselves been compiled. The modeler thus has much greater freedom in utilizing the

framework that best suits the needs of the task. For instance, recent work has demonstrated that integrating a model of driver behavior with models of cell-phone dialing can predict the affects of driver distraction on steering and speed control [20]. While the driver model required the complexity of the full-fledged architecture, the models for cell-phone dialing (or other simple in-vehicle tasks) could more easily have been written in a higher-level framework. By compiling a simple dialing model to a cognitive architecture, we can interface the model with the complex driver model and run the two together in a simulation to predict driver performance while dialing.

*(4) Model Refinement*

With a typical higher-level model of behavior, the modeler starts and ends with this description and gleans whatever information possible out of it. However, there are times when the model raises issues that need further exploration, and the higher-level model is insufficient to describe behavior at that level—for instance, the coordination of mouse and eye movements in menu selection [4, 8]. The compilation process generates a lower-level model that has greater potential to explore issues further, perhaps by examining the lower-level model directly, or perhaps by refining it further into a more detailed and accurate model. As an alternative, the higher-level model may simply be viewed as a starting point for modeling: the modeler generates a first-pass description of behavior in the higher-level language, compiles it to the cognitive architecture, and then develops the model from there. Compilation thus allows for an extremely flexible process of progressive refinement in which the modeler can explore issues at a variety of levels of detail and for a huge saving in time for model development .

## THE ACT-Simple FRAMEWORK

ACT-Simple is a higher-level framework built on the ACT-R cognitive architecture [1]. ACT-Simple includes a set of basic perceptual, motor, and cognitive commands that compile down to ACT-R production rules. We first provide a very brief overview of the ACT-R architecture, and then outline ACT-Simple's command set as well as the compilation process that translates each command into a set of production rules. The full ACT-Simple system is publicly available for download, testing, and use.[1]

## ACT-R

The ACT-R cognitive architecture is a production-system architecture based on two distinct types of knowledge: declarative and procedural. Declarative knowledge comprises *chunks* of information that can represent factual information (e.g., '4+3=7') or encoded perceptual information (e.g., the word 'platypus' on a written page). Procedural knowledge comprises condition-action *production rules* that act on declarative knowledge. When the conditions of a production rule match, the rule can alter declarative knowledge, initiate perceptual actions, and/or issue motor commands. ACT-R incorporates cognitive, perceptual, and motor processes that allow non-competing resources to operate in parallel (e.g., shifting visual attention while moving a mouse). The perceptual and motor processes, through recent major extensions to the architecture [5], can interact with realistic simulations that provide architectural models with an increased sense of plausibility as accurate psychological models.

ACT-R has been applied to model behavior in an extremely wide array of domains, from primarily cognitive tasks such as choice, arithmetic, or scientific discovery [see 2] to complex, dynamic, interactive tasks such air-traffic control [17], driving [21], unmanned air vehicles [2], and real-time multiplayer computer games [3]. The development of even simple models requires at least several days to weeks[2] of experience using the system, and (arguably) takes months to years to become an expert in its use. The ACT-Simple framework aims to serve as a far simpler and easier framework that shortens the learning curve but still takes advantage of ACT-R's many powerful features.

Table 1: The ACT-Simple command set, with optional command arguments in brackets.

| | |
|---|---|
| (move-hand *position*) | (press-key [*key*]) |
| (move-mouse) | (speak [*string*]) |
| (click-mouse) | (look-at) |
| (press-mouse) | (listen [*time*]) |
| (release-mouse) | (think) |

## Command Set

Table 1 displays the ACT-Simple command set, where brackets indicate optional arguments. The commands map onto basic motor, perceptual, and cognitive processes as follows:

- (move-hand *position*) : moves the right hand either to the mouse (value 'mouse') or to the home position on the keyboard (value 'home'); uses the default ACT-R performance parameters for determining the timing of the movement (~650-800 ms).
- (move-mouse) : moves the mouse; requires that the right hand is on the mouse; uses the default ACT-R parameters for a mouse motor movement assuming a distance of 500 pixels (~650-750 ms).
- (click-mouse) : clicks the mouse button (i.e., presses and releases); requires that the right hand is at the mouse; uses the default ACT-R parameters (~200-350 ms).
- (press-mouse) : presses the mouse button (down only); requires that the right hand is at the mouse; currently

[1] http://hmil.cs.drexel.edu/projects/actsimple/

[2] The annual ACT-R summer school teaches the basic components of the architecture in two weeks of full-time study, by the end of which students can typically develop basic yet interesting models of given task domains.

uses the default ACT-R parameters for the click-mouse operator (~200-350 ms).

- (release-mouse) : releases the mouse button (up only); requires that the right hand is at the mouse; currently uses the default ACT-R parameters for the click-mouse operator (~200-350 ms).
- (press-key [*key*]) : presses a key on the keyboard; requires that the right hand is at the keyboard for a right-handed key; if no key is specified, alternates between 'd' and 'k' (the middle-finger keys for each hand); uses the default ACT-R parameters (~200-350 ms).
- (speak [*string*]) : vocalizes a string; if no string is specified, defaults to saying "hello"; uses the default ACT-R parameters (~350-1000+ ms).
- (look-at) : attends to and encodes a visual object; uses the default ACT-R parameters assuming a distance of 500 pixels (~185 ms).
- (listen [*time*]) : listens to and encodes a sound; if no time is specified, defaults to 500 ms.
- (think) : performs a cognitive action; uses the GOMS value of 1.2 seconds.

It should be noted that in situations where default ACT-R parameters produce a range of duration values, the actual duration depends on the current context of the model, and especially whether the features of necessary motor actions have already been prepared; for instance, pressing the same key twice in row generates a smaller duration for the second press because ACT-R has already prepared the required movement features and needs only to execute them (see [5] for further information).

### Modeling

Given the ACT-Simple command set, a modeler can create a model by performing a task analysis of the desired task and writing out the sequence of commands that correspond to behavior in the task. In the following major section we will present several such models for the sample tasks of query entry and phone dialing. Overall, the ACT-Simple framework is fairly similar to the KLM-GOMS framework [6], and many of the basic techniques and issues for KLM-GOMS modeling carry over to ACT-Simple. Because of the high degree of similarity here, we refer interested readers to the extensive GOMS literature for this information, particularly the basics of task analysis and initial model development. In particular, we should note two important commonalities of ACT-Simple and KLM-GOMS. First, ACT-Simple is meant to represent expert performance — that is, well-practiced performance as would be observed from an expert user. Second, ACT-Simple models represent a linear, sequential behavior with no significant deviation, as imposed by the specification of a sequential list of commands. Nevertheless, after the compilation to an ACT-R model, a modeler can indeed take advantage of ACT-R's learning mechanisms to optimize behavior and choice (or "conflict resolution") mechanisms to incorporate decision making — one of the benefits of the compilation approach.

Table 2: Sample translation of (press-key *a*).

```
(p do-task-press-key-102
     =goal>
          isa do-task
          state 1
     =manual-state>
          isa module-state
          modality free
==>
     +manual>
          isa press-key
          key a
     =goal>
          state 2
)
```

### Compilation

Given an ACT-Simple model comprising a sequence of commands, the compilation process translates this sequence into a set of ACT-R production rules. At the implementation level, the compilation utilizes LISP macros to perform this (very much syntactic) translation. But more generally, the process can be characterized as follows. First, an initial production is generated that initializes the current goal to the first stage of running; in essence, the productions maintain a counter to ensure that the command sequence occurs in the correct order. Next, the process translates each command to either one or two corresponding productions, as described below. Finally, the production set is wrapped in ACT-R-specific code to initialize model state, declarative memory, etc., and the resulting ACT-R model is ready to run. Note that the final compiled model can be viewed directly and analyzed, edited, and run just as any "hand-crafted" ACT-R model.

The compilation of the individual commands reduces to re-writing the command to a production rule that an ACT-R expert modeler might write directly. For most of the commands, there is a one-to-one mapping from each command to a single production rule. For instance, as shown in Table 2, the compiled rule for the (press-key) command checks that the state of the manual module is free and, if so, issues a command to that module to press the given key (or a default key as described earlier); again, the rule is essentially the same rule that an ACT-R expert would write from scratch for pressing a key. Only one command is translated to two productions: (look-at) compiles to one production that issues the shift of visual attention and another that ensures that the attended object has finished being encoded; although we could avoid waiting for encoding to end, typically a model encodes a visual object because that information is needed, and thus we chose to lean toward the most common case. To allow ACT-R to produce realistic mouse and eye movements, the model includes a dummy window with two objects 500 pixels apart, and these objects are used as the targets of mouse and eye movements. The (listen *time*) command does not actually use ACT-R's audition module because of

the difficulty of creating simulated sounds at the right time, and thus its production rule simply stalls the cognitive processor for the given duration. Similarly, the (think) command does nothing other than stall the cognitive processor (taking 1.2 seconds to fire).

### Discussion

The specification of the ACT-Simple command set required several difficult design choices with no clear correct answers. First, two commands, (move-mouse) and (look-at), could include specification of the location to which to move/look. However, knowing locations would require some rough prototype of the task interface; although we could assume such a prototype, or even provide a separate description language for the interface itself, this additional specification goes against the desired simplicity of the higher-level modeling language. Second, the (think) command could certainly be replaced with more specific thought processes—for instance, a command (retrieve) representing the memory retrieval of a declarative chunk. Again, we opted for the simpler option of a single cognitive operator with a duration taken directly from KLM-GOMS; for more specific cognitive processes, modelers can always go directly into the compiled ACT-R model and utilize its strengths as a powerful representation of cognition. In a sense, these choices correspond to a decision on the appropriate level of modeling, and thus the chosen command set will likely evolve (or expand into multiple command sets) as users find what level works best for different domains and applications.

Nevertheless, given our design choices, it is enlightening to consider how the ACT-R architecture provides ACT-Simple models with predictive power beyond that of a basic framework such as KLM-GOMS. One benefit is that ACT-R automatically parallelizes those processes that do not interfere with each other; for instance, if the ACT-Simple model performs the commands (move-hand) and (speak) consecutively, ACT-R will issue the hand movement and, 50 ms later (the cost of firing a production rule), will issue the speak command, and both commands can continue on from that point in parallel. Thus, the execution of the compiled ACT-R model begins to closely resemble that of a CPM-GOMS model [9] rather than the original KLM-GOMS-like high-level model. Another benefit is that, as mentioned earlier, the compiled models are sensitive to context in that their behavior is affected by the current state of the various cognitive, perceptual, and motor modules—for example, by predicting shorter times for consecutive keypresses on the same letter, or consecutive hand/finger movements more generally. These are just two examples of how the compiled ACT-Simple models inherit parameters and constraints of the ACT-R architecture, which in turn gives them significantly greater power as *a priori* predictors of behavior.

## EMPIRICAL VALIDATION

To validate the ACT-Simple framework, we conducted a study that compared model predictions and empirical data for two tasks: a query entry task described by Nielsen and Phillips [19], and a cell-phone dialing task described by Salvucci [20]. The goal of the study was to generate *a priori* predictions of total times for four conditions in each task. Ultimately, we hoped to demonstrate that the ACT-Simple framework facilitates rapid development of user models and, at the same time, reasonably accurate predictions of behavior comparable to those of GOMS and its variants [see 10, 11].

### Tasks

*Query entry*. The first task analyzed in the study is a query-entry task [19] used to compare various methods of predicting task times. Specifically, the task involved issuing query of one or two phone numbers to one of several databases. The user issued the query using one of two interfaces: a *dialog-box interface* and a *pop-up menu interface*. The necessary actions for each interface are listed in Table 3. The original study [19] examined the predictions obtained by "cold," "warm," and "hot" heuristic estimates, GOMS analysis, and empirical user testing. In our study we compare the Nielsen and Phillips task times obtained by GOMS analysis and user testing with new times obtained in the ACT-Simple framework.

*Cell-phone dialing*. The second task in our study is a cell-phone dialing task [20] used to study the effects of interface use and distraction on driver performance. The original study used a hands-free phone mounted on the dashboard and asked subjects to dial the phone both alone and while driving; we focus only on the case where subjects simply dial the phone (i.e., not while driving). The study included four methods of dialing the phone: *full-manual dialing*, typing all digits of a seven-digit phone number; *speed-manual dialing*, typing only a single "speed digit" representing the full phone number; *full-voice dialing*, speaking the entire number and listening to the recognized number to confirm correct recognition; and *speed-voice dialing*, speaking a single phrase representing the full phone number. Table 4 lists sample sequences for each of the four methods. Each method starts with the user pressing the **Power** button on the phone, and the manual dialing methods end with the user pressing the **Send** button. The "send" is implied for the voice dialing interfaces after the phone confirms that it is connecting.

### Modeling

Given these tasks and the ACT-Simple framework, we wished to validate the framework by modeling the tasks and comparing their predictions to real-world data. First, we obtained two sets of models, a novice set and an expert set. The novice models were developed by an undergraduate student with one quarter-long course in cognitive modeling. The expert models were developed by two expert modelers, both faculty members with over seven years of cognitive modeling experience. The novice modeler and the expert modeler (the second author) were given the original papers describing the tasks [19, 20] and the list of ACT-Simple commands with basic instructions on creating a model; model creation consisted simply of generating a sequence of ACT-Simple commands for each task condition. Both modelers were also instructed not to read the empirical results of the studies before modeling was complete. The expert models were then checked by the second expert (the first author) for consistency and

Table 3: Query-entry task actions for each interface.

| *Dialog Box* |
|---|
| Pull down the query menu, find the desired database, open its submenu, select "query on telephone number"<br>→ opens a query dialog box<br><br>For each phone number, click in the editable text box (or select existing phone number), type the query phone number, click the "Add" button<br>→ adds number to query list<br><br>Click "Ok"<br>→ issues query |
| *Pop-up Menu* |
| For each phone number, click down on the visible phone number<br>→ brings up pop-up menu of databases<br><br>Find the desired database, release mouse button<br>→ issues query |

Table 4: Cell-phone dialing actions for each interface.

| *Full-Manual* | *Full-Voice* |
|---|---|
| Press **Power**<br>Press **5, 5, 5, 4, 2, 8, 3**<br>Press **Send** | Press **Power**<br>Say **5, 5, 5, 4, 2, 8, 3**<br>Listen for **5, 5, 5, 4, 2, 8, 3**<br>Listen for "Connecting…" |
| *Speed-Manual* | *Speed-Voice* |
| Press **Power**<br>Press **2** (speed number)<br>Press **Send** | Press **Power**<br>Say "**home"**<br>Listen for "**home"**<br>Listen for "Connecting…" |

assumption violations, which led to several minor edits: for query entry, a (think) command in the one-number dialog interface where it was for the two-number interface, and (look-at) commands to look at the phone number while typing; for cell-phone dialing, (look-at) commands before pressing the power button, (move-hand) commands to simulate the start position away from the keyboard, removal of (press-key) arguments because of mismatch between keyboard digit and phone digit positions, and revision of a mistaken (speak) command.

Table 5 shows the time involved in the development of the expert and novice models (not including time needed to read and understand the interfaces, which is assumed to be common to all other analytic methods). The novice models required only slightly more time to develop than the expert models, and all sets of models for each task were completed in a time span of less than one hour.

Table 5. Time needed to create models.

| | Query entry | Cell-phone dialing |
|---|---|---|
| Expert | 28 min | 34 min |
| Novice | 44 min | 36 min |

## Results

*Query entry*. We first analyze the results of the query-entry task, comparing the empirical data [19] to the expert and novice model predictions. Table 6 shows the empirical results for total time to complete each of the four tasks, labeled according to whether they represent the dialog-box or the pop-up menu interface ("Dialog" or "Popup") and whether they represent querying one or two telephone numbers ("1" or "2"). The table also shows the results for the novice and expert models in terms of task time, percent error, and correlation (the latter two with respect to the empirical data). The empirical data show that the dialog-box interface required significantly more time than the pop-up menu interface, and entering two numbers (not surprisingly) required significantly more time than entering one number. The expert models matched the empirical data extremely well: the percent errors are 12% or less and the correlation was excellent, $R$=.99. The novice models did not match the data nearly as well quantitatively — particularly for the dialog-box model, with errors up to 64% — but the correlation remained excellent, $R$=.99, and thus even the novice models successfully predicted the rank-order differences between the four interfaces. The biggest difference by far between the expert and novice models was the number of (think) commands: the novice assumed mental operators before every small group of commands (e.g., (think) + (lookat)), whereas the expert used the mental operators very sparingly given the presumed "expert" nature of behavior in the task.

*Cell-phone dialing*. Our analysis of the cell-phone dialing task involves comparison of the model predictions with the empirical data [20], as shown in Table 7. The empirical data show the pattern that the voice interfaces required more time than the manual interfaces and that the "full" interfaces (i.e., entering or saying all numbers) required more time than the "speed" interfaces (i.e., entering a speed number or saying a phrase). As for the query-entry task, the expert models closely matched the data with a maximum error of 12% and an excellent correlation, $R$=.98. The novice models were quite far off in terms of quantitative predictions, but also matched well in terms of correlation, $R$=.93; as before, we see that even the novice models generate reasonable rank-order predictions (with the exception here that the novice predictions for speed-voice and full-manual rankings are reversed). The novice models again included far more (think) operators than the expert models, and in addition, the novice assumed quite long times for listening in the voice interfaces (10 s for seven digits, 2 s for "home").

Table 6. Empirical, expert model predictions, and novice model predictions for the query-entry task, including percent error and correlation.

| | Dialog-1 | Dialog-2 | Popup-1 | Popup-2 |
|---|---|---|---|---|
| Data | 15.40 | 25.50 | 4.30 | 6.50 |
| Expert | 15.55 | 24.40 | 3.78 | 7.20 |
| (Error) | (1%) | (4%) | (12%) | (11%) |
| — *R* — | ——— .99 ——— | | | |
| Novice | 25.01 | 41.87 | 4.87 | 9.37 |
| (Error) | (62%) | (64%) | (13%) | (44%) |
| — *R* — | ——— .99 ——— | | | |

Table 7. Empirical, expert model predictions, and novice model predictions for the cell-phone dialing task, including percent error and correlation.

| | Full-Manual | Speed-Manual | Full-Voice | Speed-Voice |
|---|---|---|---|---|
| Data | 5.21 | 2.68 | 7.71 | 3.77 |
| Expert | 4.79 | 2.99 | 8.49 | 4.14 |
| (Error) | (8%) | (12%) | (10%) | (10%) |
| — *R* — | ——— .98 ——— | | | |
| Novice | 8.79 | 4.90 | 21.77 | 10.57 |
| (Error) | (69%) | (83%) | (182%) | (180%) |
| — *R* — | ——— .93 ——— | | | |

## GENERAL DISCUSSION

The model results are encouraging in several respects. First, the expert model predictions matched extremely well with the empirical data, suggesting that an expert ACT-Simple modeler (or team of 2-3 modelers) can generate very good quantitative predictions. In addition, we expect that minimal expertise in the ACT-Simple framework can be achieved with approximately 1-2 days of training and 1-2 weeks of practice — arguably, similar to the training needed to reach a useful familiarity with the GOMS framework. Second, while the novice model predictions mismatched at a quantitative level, the comparative predictions — that is, the predictions of the rank order of the various interfaces — was reasonably good, as indicated by the high model-to-data correlations. Third, the models took little time (less than one hour per model set) to develop, demonstrating that several interface options can be evaluated and compared in a matter of a few hours.

ACT-Simple also has some notable weaknesses, the most critical of which is the lack of constraint in using the (think) operator. As for similar frameworks such as KLM-GOMS, adding these mental operators is often the most difficult aspect of modeling, and at the same time can have a drastic impact on predicted times (as evident in the overpredictions of the novice models). Researchers have proposed more systematic methods of adding mental operators in the GOMS framework and these same ideas apply just as well for the ACT-Simple framework — for instance, determining the size of "thoughtless chunks" for novice vs. expert behavior, or specifying more detailed cognitive operators (e.g., simple retrieval) with distinct time values. There is clearly much room for further investigation on this front, but nevertheless, this work along with the large GOMS literature demonstrates that perfect modeling of cognitive processing is not always needed for fast and reasonable predictions of performance. However, unlike GOMS, the ACT-Simple framework has the advantage that a model compiled down to ACT-R can be augmented and further developed into a very rigorous model of cognitive processing with integrated perceptual and motor processing.

From a broader perspective, compiled ACT-Simple models are actually quite reminiscent of models developed in the CPM-GOMS framework [9]. In particular, the compiled models incorporate (typically) small bursts of cognitive processing that initiate parallel streams of perceptual and motor processing. The resulting execution in the ACT-R cognitive architecture generates, in essence, the same critical path that is the crux of the CPM-GOMS technique; in fact, one could imagine compiling a CPM-GOMS-like model into ACT-R using the basic compilation process presented here. Such a method could also take advantage of recent work tying CPM-GOMS and Apex to automate resource scheduling [12].

The proposed ACT-Simple framework represents only one possible point along a continuum of higher-level and lower-level frameworks, and there are clearly many other branches to explore. For instance, the commands that currently take arguments — e.g., (look-at) and (move-mouse) — could be expressed in more detail whenever possible to increase the task and cognitive plausibility. Or as another example, the basic ACT-Simple commands could be compiled down to another cognitive architecture — e.g., Soar [16, 18] or EPIC [17] — and could potentially be used as a metric of comparison between architectures. Such a technique would allow ACT-Simple models to be integrated with existing Soar or EPIC models, much like current ACT-Simple models can be easily integrated with large-scale ACT-R models — e.g., integrating the cell-phone dialing models with the ACT-R driver model [21]. Thus ACT-Simple serves as a first step toward a large set of useful cognitive modeling tools for rapid prototyping, evaluation, and comparison.

## ACKNOWLEDGMENTS

This work was supported in part by Office of Naval Research grant N00014-03-1-0036 to the first author, a Ford Motor Company gift to the first author, and an Exploratory Research Seed Project grant to the second author. We thank Alex Chavez for his extensive help in the modeling portion of this study.

## REFERENCES

1. Anderson, J. R., & Lebiere, C. (1998). *The atomic components of thought*. Hillsdale, NJ: Erlbaum.
2. Ball, J., Gluck, K., Krusmark, M., Purtee, M., & Rodgers, S. (2002). *Process and challenges in development of the Predator air vehicle operator model*. Eighth Annual ACT-R Workshop, Carnegie Mellon University, Pittsburgh, PA.
3. Best, B., Lebiere, C. & Scarpinatto, C. (2002). *Modeling synthetic opponents in urban combat simulations*. Eighth Annual ACT-R Workshop, Carnegie Mellon University, Pittsburgh, PA.
4. Byrne, M. D. (2001). ACT-R/PM and menu selection: Applying a cognitive architecture to HCI. *International Journal of Human-Computer Studies*, *55*, 41-84.
5. Byrne, M. D., & Anderson, J. R. (2001). Serial modules in parallel: The psychological refractory period and perfect time-sharing.. *Psychological Review*, *108*, 847-869.
6. Card, S., Moran, T., & Newell, A. (1983). *The psychology of human-computer interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates.
7. Gray, W. D., John, B. E., & Atwood, M. E. (1993). Project Ernestine: A validation of GOMS for prediction and explanation of real-world task performance. *Human-Computer Interaction*, *8*, 237-309.
8. Hornof, A. J., & Kieras, D. E. (1997). Cognitive modeling reveals menu search is both random and systematic. In *Human Factors in Computing Systems: CHI 97 Conference Proceedings* (pp. 107-114). New York: ACM Press.
9. John, B. E. (1990). Extensions of GOMS analyses to expert performance requiring perception of dynamic visual and auditory information. In *Proceedings of CHI 90* (pp. 107-115). New York: ACM Press.
10. John, B. E., & Kieras, D. E. (1996). The GOMS family of user interface analysis techniques: Comparison and contrast. *ACM Transactions on Computer-Human Interaction*, *3*, 320-351.
11. John, B. E., & Kieras, D. E. (1996). Using GOMS for user interface design and evaluation: Which technique?. *ACM Transactions on Computer-Human Interaction*, *3*, 287-319.
12. John, B., Vera, A., Matessa, M., Freed, M., & Remington, R. (2002). Automating CPM-GOMS. In *Human Factors in Computing Systems: CHI 2002 Conference Proceedings*. New York: ACM Press.
13. Kieras, D. E. (1988). Towards a practical GOMS model methodology for user interface design. In M. Helander (Ed.), *Handbook of Human-Computer Interaction* (pp. 135-157). New York: North-Holland Publishing.
14. Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1-64.
15. Lebiere, C., & Anderson, J. R. (1993). A connectionist implementation of the ACT-R production system. In *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Lawrence Erlbaum Associates.
16. Lee, F. J., & Anderson, J. R. (2001). Does learning of a complex task have to be complex? A study in learning decomposition. *Cognitive Psychology*, *42*, 267-316.
17. Meyer, D. E., & Kieras, D. E. (1997a). A computational theory of executive cognitive processes and multiple-task performance: Part 1. Basic mechanisms. *Psychological Review*, *104*, 3-65.
18. Newell, A. (1990). *Unified theories of cognition*. Cambridge, MA: Harvard University Press.
19. Nielsen, J., & Phillips, V. L. (1993). Estimating the relative usability of two interfaces: Heuristic, formal, and empirical methods compared. In *Proceedings of INTERCHI 1993* (pp. 214-221). New York: ACM Press.
20. Salvucci, D. D. (2001). Predicting the effects of in-car interface use on driver performance: An integrated model approach. *International Journal of Human-Computer Studies*, *55*, 85-107.
21. Salvucci, D. D., Boer, E. R., & Liu, A. (2001). Toward an integrated model of driver behavior in a cognitive architecture. *Transportation Research Record*, 1779.