

# Model-based Evaluation of Cell Phone Menu Interaction

Robert St. Amant and Thomas E. Horton

Department of Computer Science  
North Carolina State University  
Raleigh, NC 27695

stamant@csc.ncsu.edu tehorton@eos.ncsu.edu

Frank E. Ritter

School of Information Sciences and Technology  
The Pennsylvania State University  
University Park, PA 16801

ritter@ist.psu.edu

## ABSTRACT

Cell phone interfaces are now ubiquitous. In this paper, we describe concepts to support the analysis of cell phone menu hierarchies. We present an empirical study of user performance on five simple tasks of menu traversal on a cell phone. Two models we tested, based on GOMS and ACT-R, give very good predictions of behavior. We use the study results to motivate an effective evaluation process for menu hierarchies. Our work makes several contributions: a novel and timely study of a new, very common HCI task; new models for accurately predicting performance; novel development tools to support such modeling; and a search procedure to generate menu hierarchies that reduce traversal time, in simulation studies, by about a third.

## Categories and Subject Descriptors

H.5.2 [Information Interfaces and Presentation]: User Interfaces—Evaluation/methodology.

## Keywords

Mobile telephones, menu traversal, cognitive modeling, evaluation

## INTRODUCTION

There are at least a billion cellular telephones in use today, and this number is expected to double by 2007 [4]. Cell phones are used for more than making calls; they include tools for managing contact information, voice mail, and hardware settings, and often software for playing games, browsing the Web, and connecting to specialized information services. The market penetration of cell phones is much higher than that of conventional computers, which raises significant opportunities and challenges for HCI.

The focus of this paper is on techniques for evaluating specific aspects of cell phone usability, especially the hierarchical menus that provide access to most functionality aside from dialing and data entry. While cell phone menu interfaces may appear simple at first glance, they pose a non-trivial design problem. Consider the menu hierarchy for the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CHI 2004, April 24–29, 2004, Vienna, Austria.

Kyocera 2325 cell phone, shown in Figure 1. If we count as terminals those selections that open an application (e.g., a game), a list of data (e.g., recent calls), or the cell phone equivalent of a dialog box (e.g., for setting the ringer volume), then this hierarchy contains 98 terminals, reachable through 22 intermediate selections. The longest menu contains 12 items—all associated with the selection of different sounds. The shortest menu contains a single item, for entering a voice memo. Terminals in the hierarchy are up to four levels deep, and the mean number of actions to reach an item (scrolling plus selection), over all 98 terminals, is 13.3, taking on the order of 7 s for an experienced user.

This menu hierarchy is as large as that of a moderately-sized desktop application (e.g., Eudora 5.2 with 103 items). Designing menu systems for cell phones is made more difficult by several factors:

- Discrete selection actions in the form of button presses are usually needed to move from one menu item to any other, because most cell phones lack more direct selection capabilities (e.g., a mouse or touch screen).
- Cell phone displays are small, allowing only a few menu items to be displayed at a single time. Many cell phones lack functionality for paging up or down, making display limitations even more significant.
- There is less standardization in hardware supporting menu traversal for cell phones than for desktop machines. Some phones have two-way directional buttons, others four-way; some have a labeled “Menu” button, while others rely on a button with overloaded functionality. Button placement can vary significantly, with “Cancel” and “OK” buttons reversed from one phone to another. If interfaces are developed for the lowest common denominator, independently of specific hardware (which is common practice at the mobile application level), then even cell phones with elaborate interaction support become less efficient.

These factors suggest that cell phone menu interfaces deserve close analysis, and that they may need specialized techniques for their development and evaluation.

This paper is in two parts. We first describe a small empirical study of the traversal of cell phone menus, along with three models for predicting user performance: a Fitts’

law model, a GOMS model, and an ACT-R model. The latter two models give good to very good predictions of user behavior. In the second part of the paper, we turn to the issue of designer-level support for evaluating cell phone menus. We use our empirical results to define a novel evaluation metric for cell phone menus. We describe a search process that generates improvements to a menu hierarchy with respect to a given set of user profiles. We make several contributions: a novel and timely study of a very common new HCI task, new models for accurately predicting performance, and a simple, theoretically motivated search procedure that generates menu hierarchies that reduce traversal time in simulation studies by a third.



Figure 1. Kyocera 2325.

### A PERFORMANCE STUDY

Our interest is in expert (i.e., practiced and error-free) use of cell phone menu systems. For control purposes, however, it was not possible to collect data from experienced users on their own cell phones, with all their potential differences in hardware and software. As a compromise, we had users practice a small number of tasks (so that all tasks could be remembered easily) and then carry them out on a single cell phone. Though restrictive, these conditions give a reasonable starting point for an empirical study.

We used a Kyocera 2325, as in Figure 1. The Kyocera display shows three menu items at a time, except for the first level, in which a single selectable item is shown. Each new menu list is displayed with the top item highlighted. The OK button selects the currently highlighted item. If the item is not a terminal, the result is a new list of items. The CLR button returns to the previous level in the hierarchy. On the four-way scrolling button, UP and DOWN move through the item list; except for the first menu, RIGHT and LEFT are inactive. Downward scrolling is incremental, with items appearing one at a time at the bottom of the screen.

### Procedures

We recruited twelve experienced cell phone users for our study, all male undergraduates in computer science, who took part for course credit. All were right handed. All but one used their right hand to hold the cell phone, and all used their thumb to press keys.

Participants started with a practice stage, in which they familiarized themselves with the cell phone and its menu system. We gave each participant a paper form describing how five target menu items were to be reached, as follows:

Menu >>> Settings >> Sounds > Ringer Volume

Menu >>>> Tools & Games >> Tip Calculator

Menu, Contacts, View All

Menu >>>> Tools & Games, Scheduler, View Day

Menu >>>>> Web Browser

Each “>” represents a scrolling action, with commas separating consecutive selection actions. Reaching each of the target items (those at the end of each sequence) constituted a task in the study. Participants practiced each task until they could carry it out three times in a row without error.

After the practice stage, we recorded the tone produced by each key press as transmitted through the earphone jack of the cell phone. Data collection was initiated by the first key pressed by the participant. The onset of each key press is detectable by a threshold test on the waveform, using software we wrote for this purpose. Each tone lasts approximately 0.095 seconds, during which time the display changes, before the key is released. System responses are treated as occurring within elementary key press actions and not contributing to the duration of user actions.

Each trial in the study required reaching one of the target items from the practice stage without access to the paper form. Tasks were presented to participants in a randomized order. We obtained five correct trials per participant (i.e., without errors or extraneous actions), discarding fewer than 10 trials across all participants, less than 3% of the data. This means that our cleaned dataset contains only OK and DOWN key press actions, 2,280 observations in total (2,280 = 12 users • 5 repetitions • (10 + 9 + 3 + 8 + 8 actions per task)).

Table 1 shows the mean duration per task, over all participants in the study. User performance is on the order of five times slower than for single-level menu selection with a mouse [3], which highlights the importance of specialized models for this task, as we discuss below.

### Models of user behavior

We predicted performance with three models, each supported by a considerable background literature. The models were developed independently of each other based on a preliminary pilot test with a single user and a single task. The models were also developed independently of the

| Task                  | N actions | Duration      |
|-----------------------|-----------|---------------|
| <i>Ringer Volume</i>  | 10        | 4.954 (1.077) |
| <i>Tip Calculator</i> | 9         | 4.027 (0.921) |
| <i>View All</i>       | 3         | 1.271 (0.412) |
| <i>View Day</i>       | 8         | 4.393 (0.971) |
| <i>Web Browser</i>    | 8         | 3.391 (0.827) |

Table 1. Task duration in seconds, with mean and standard deviation shown (in parentheses).

actual study data, and none reflect adjustment of model parameters to fit the data.

The first model is a Fitts' law model. We use MacKenzie's model of one finger typing for text-entry on mobile phones [8], in which movement time (in ms) for thumb input is

$$MT = 176 + 64 \log_2(D/W + 1), \quad (\text{Eq. 1})$$

where  $D$  represents the distance (amplitude) of the movement and  $W$  the width of the target. The value for  $D$  in our study was 14.5 mm, which separates the OK button and the DOWN button area, with widths of 6 mm and 10 mm, as measured on the physical device.

The second model is a GOMS model [6, 7]. GOMS methods for task analysis produce hierarchical descriptions of methods and operators needed to accomplish goals; some GOMS models have been strikingly successful in critical HCI domains [5]. In our model we define a method for each task in the study. Each step within a task corresponds to the selection of a menu item. Thus the Ringer Volume task involves the steps of selecting the Menu, Settings, Sounds, and Ringer Volume items, in sequence. Each of these steps in turn decomposes into a Select-X task, which involves scrolling until a given item X in the sequence is reached:

- S1 Look at location for X.
- S2 If (found-X and not on-OK-button),  
move-to OK-button; select-item; goto S4.
- If (found-X and on-OK-button),  
  select-item; goto S4.
- If (not found-X and on-scroll-button), scroll.
- If (not found-X and not on-scroll-button),  
  move-to scroll-button; scroll.
- S3 Goto S2.
- S4 Return with goal accomplished.

Following guidelines established by Kieras in his work on GOMS and GLEAN3 [7], all the steps above have a 50 millisecond duration except S2, whose duration is computed from movements, following the Fitts' law model above, and button presses lasting 350 ms. Other models are reasonable, both faster and slower, but we built our model to be as fast as possible, given what we know about the domain. The model assumes negligible system response time, that there are no verification steps, and that the information on the display starts in focus and does not require re-acquisition during scrolling activity.

The third model is based on the ACT-R 5.0 cognitive architecture [1].<sup>1</sup> ACT-R integrates theories of cognition, visual attention, and motor movement and has been the basis for a number of models in HCI [10]. ACT-R models simulate internal cognitive processing, such as changes of

<sup>1</sup> We picked ACT-R as a representative cognitive modeling architecture and as a common choice in HCI work. All models described in this section, plus the study data, are at [www4.ncsu.edu/~stamant/papers/RSA-TEH-FER-chi04](http://www4.ncsu.edu/~stamant/papers/RSA-TEH-FER-chi04).

attention and memory retrievals, as well as external actions, such as movement of the fingers, producing changes in the state of the model and durations for the actions. Roughly speaking, ACT-R models provide details that can explain behavior in cognitive terms at a level not addressed by more abstract GOMS models [11].

In our ACT-R model, a simulated field of view is maintained that represents the current menu items in the hierarchy. This view changes depending on the selection and scrolling actions taken by the model. Before running, the model's memory is initialized with a set of chunks representing the hierarchical relationships between intermediate and terminal menu items. This allows the model to determine whether a given menu item (e.g., "Settings") is on the path to a target item (e.g., "Ringer Volume") by retrieving this information from memory.

The model starts with the goal of selecting a specific target menu item. The simulation environment shows a single highlighted item, which the model searches for in its field of view. Once found, the item is attended and then encoded, so that its text representation becomes accessible. If the text matches the goal item, then the model initiates motor actions to press the OK key. If the text does not match, memory is searched to determine whether the highlighted text is on the path to the target item. If it is, then the OK key is pressed, and the process repeats from the beginning with a new highlighted menu item. If the currently highlighted item is not on the path to the goal item, then the DOWN scroll key is pressed, and again the process starts over. In the model manual scrolling actions can trail behind visual processing by an unspecified amount (determined by processing in the model such as memory retrievals); the visual and manual modules become synchronized when a new menu is presented. User errors, such as pressing an incorrect key, are not modeled. Model execution is deterministic, with no noise parameters used.

### Model performance

Table 2 shows the predictions each model makes of user performance over all the menu selection tasks. Significant differences between the user data and model predictions, as given by Tukey's HSD (Honestly Significant Difference) test based on a single run of each model on all tasks, are shown as starred table entries. There are three different categories: all actions aggregated together over all tasks, only selection actions, and only scrolling actions. The Fitts'

|              | All actions    | Scrolling      | Selection      |
|--------------|----------------|----------------|----------------|
| <i>Users</i> | 0.547 (0.256)  | 0.515 (0.254)  | 0.610 (0.249)  |
| <i>GOMS</i>  | 0.542 (0.147)  | 0.505 (0.154)  | 0.616 (0.098)  |
| <i>ACT-R</i> | 0.550 (0.242)  | 0.561* (0.272) | 0.527* (0.174) |
| <i>Fitts</i> | 0.218* (0.050) | 0.202* (0.040) | 0.248* (0.057) |

**Table 2. Mean key press duration across all tasks, in seconds, with mean and standard deviation shown.**

law predictions are significantly different from user performance in all categories. The ACT-R model provides good predictions at the aggregate level, but differs in the more detailed categories, making a qualitative error in predicting that scrolling actions take longer than selection actions. The GOMS model gives predictions within 0.01 seconds (less than 2%) of the user means for all categories.

Figure 2 shows a less-aggregated view of the menu selection process. We define a “selection run” as a sequence of scrolling actions up to and including a selection action. The length of a run is the number of scrolling actions it contains, e.g., a run of length 0 corresponds to two consecutive selection actions. Figure 2 aggregates selection runs over all trials, showing the mean duration of runs from lengths 0 through 7 (by oversight our data contained no instances of runs of length 6.) As in the previous analysis, the GOMS model is closest to the user data, but here the ACT-R predictions are almost indistinguishable from the GOMS predictions. We see that both models produce larger overestimates of scrolling time for shorter runs than for longer runs. For ACT-R, we interpret this as being due to the additional visual processing needed when a new menu list is first seen, in addition to the balancing out of over/underestimates in scrolling/selection actions (the pattern in Figure 2 is more complex than can be attributed only to the latter factor.) For both models, the differences are gradually reduced as scrolling begins to dominate the longer sequences. The Fitts’ law model significantly underestimates all durations.

Figure 3 shows performance data broken down by task. One task, View All, involved only three actions and was omitted because it is similar to the other tasks. Again, we see that both the GOMS model and the ACT-R model give good approximations of user performance.

### Discussion

Many models of cell phone interaction, such as keypad dialing and one-finger text entry [8], have been based on Fitts’ law, which motivated this aspect of our evaluation. Our Fitts’ law model performs relatively poorly, however, despite the success of such models elsewhere. The model produces times that are about twice as fast as observed in users. This is actually not surprising—much of the activity of this menu selection task is outside the scope of the model. Silverberg et al. describe a comparable example of where Fitts’ law models break down, in a discussion of text entry on mobile phones [13]. For some cell phones, text entry is aided by lexicon-based word disambiguation. While typing the user ordinarily refers to the display in order to decide whether the system has correctly disambiguated the word being typed. In text entry, such cognitive processing may be rare for expert users familiar with the disambiguation system, but in menu selection, under the plausible assumption that users will not have memorized the linear position of the items in each menu, significant visual and cognitive processing is needed at each step in the process. This processing is not captured by Fitts’ law.

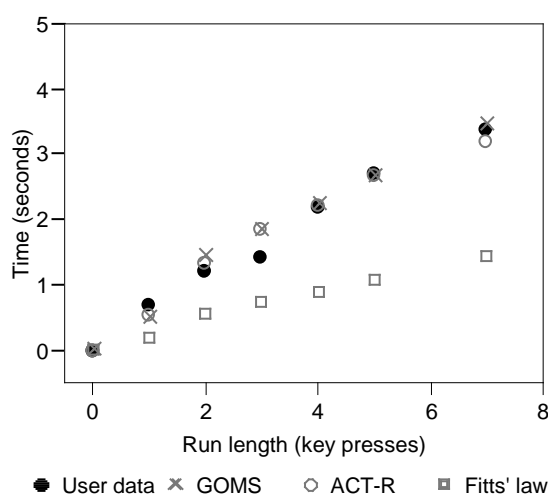
The GOMS and ACT-R models perform remarkably well in our study, with GOMS edging out ACT-R at a detailed level. As a baseline for performance, we can compare these models to a linear regression fit to all the data, where  $T$  represents time and  $n$  the number of key presses:

$$T = 0.531 n - 0.500 \quad R^2 = 0.834 \quad (\text{Eq. 2.})$$

Neither model accounts for as much variance as the least squares model, but both are close, with the GOMS model averaging an  $R^2$  of 0.786 and the ACT-R model 0.790 over the different tasks. Further, from a modeling perspective they have several compensating advantages over a post hoc model. First, they are a priori models—neither was tuned specifically to the data. Second, as seen in Figures 2 and 3, the models track some of the nonlinear aspects of user performance, at least qualitatively. Third, and most important, the GOMS and ACT-R models have theoretical underpinnings that give them explanatory power. In the case of GOMS, performance is explained by the specific tasks that are represented, the hierarchical structure in which they are combined, and dependence on a cognitive processing framework that provides specific timing predictions (e.g., for Fitts’ law movements.) The ACT-R model extends the level of detail in its explanations, in accounting for the interval between actions by explicit visual processing and memory retrievals, and in modeling visual processing and motor actions as proceeding in parallel for scrolling actions but synchronizing with selection actions. Like all model-generated explanations, these are provisional and subject to further testing, but our current results make them plausible, worth pursuing in more detailed experiments.

### SUPPORT FOR BUILDING AND USING MODELS

Building detailed models of use for novel interfaces is not yet a matter of applying simple, off-the-shelf tools. It requires support for specifying the properties of new platforms and tasks; models then need to be able to interact



**Figure 2. Selection runs. User data points (filled circles) are means of samples of size 240, 60, 120, 60, 120, and 60.**

with the platforms to carry out the tasks, which may require customization of modeling architectures. Our work has led us to develop an environment containing tools that facilitate this process for cell phone platforms and tasks.

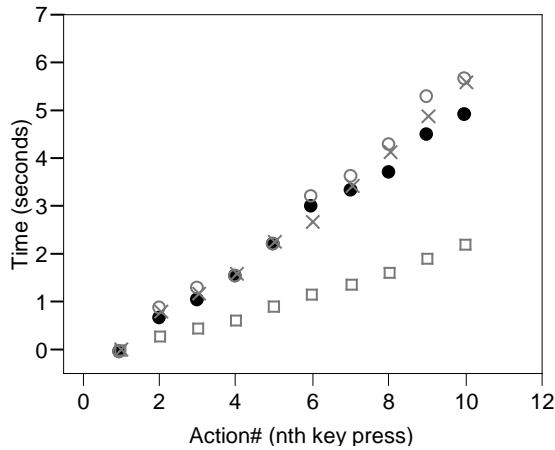
Specifying menu hierarchies in which menu traversal tasks can be defined is the easiest part. Our environment supports loading such specifications through a formatted data file. Once the information is loaded, the modeler can browse and potentially edit the result, through the simple interface shown on the left side of Figure 4.

Platform specification, in the form of building new keypads from scratch, is also straightforward. We have developed a simple direct manipulation interface in which modelers can create and name buttons, size them, and drag them into place. The visual layout of the keypad is automatically translated into a specification that can act as input to all the

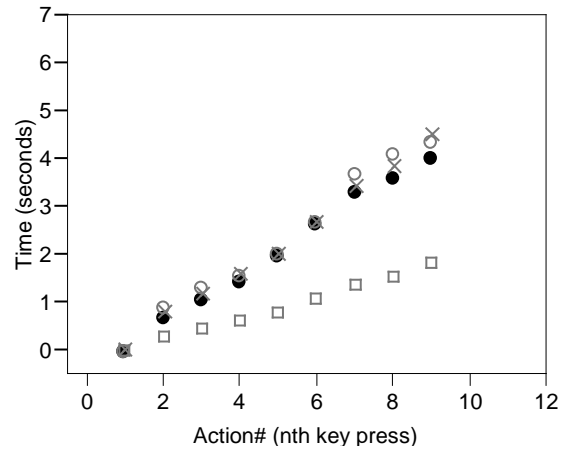
models in the previous section (though not all information may be used by all the models). The interface is comparable to conventional user interface development tools; we do not present it here because of space considerations.

Processing existing phone keypads poses a more interesting problem. It is possible to reproduce an existing keypad through the direct manipulation interface, but this is error-prone and more cumbersome than it need be. Given that images of most cell phones can easily be found online, another approach is possible. We have developed image processing algorithms to give perception capabilities to cognitive models such as ACT-R, Soar, and EPIC [12]. The algorithms are elementary but promise to be adequate for the task of interpreting images of real cell phone keypads.

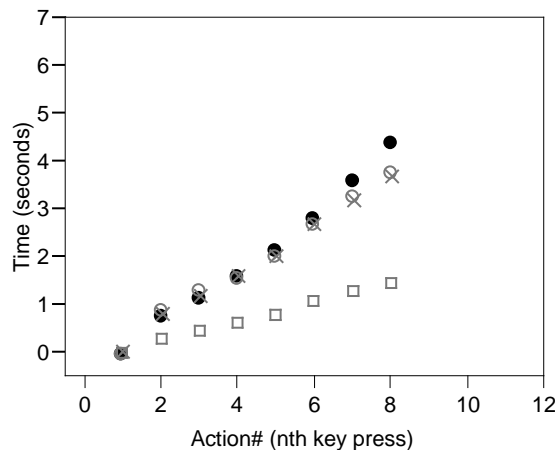
Keypad processing works as follows. The modeler loads a bitmap image of a keypad, which appears in a window



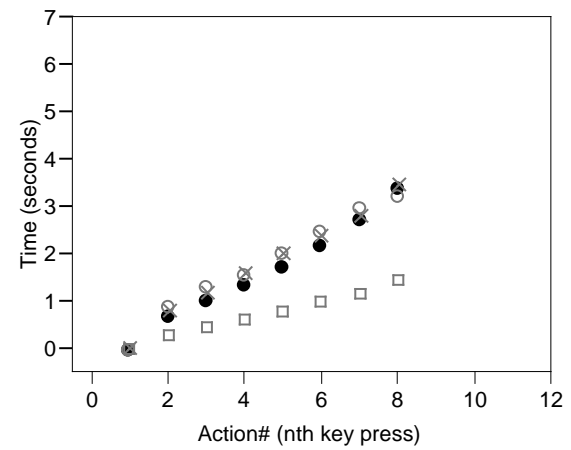
**Ringer Volume. Keys: OK >>> OK >> OK > OK**



**Tip Calculator. Keys: OK >>>> OK >> OK**



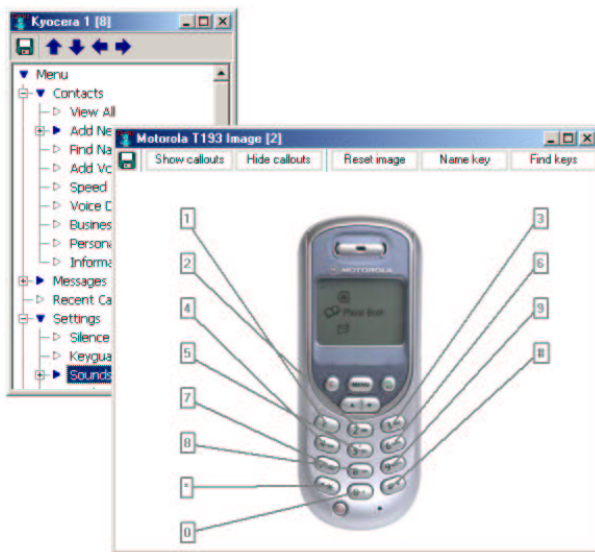
**View Day. Keys: OK >>>> OK OK OK**



**Web Browser. Keys: OK >>>>> OK**

● User data × GOMS ○ ACT-R □ Fitts' law

**Figure 3. Model prediction by task. Each user data point (filled circle) represents the mean of 60 sample points. Each graph title gives the target menu item and the sequence of selection (“OK”) and scrolling (“>”) actions.**



**Figure 4. Browsing a menu hierarchy (left); automatic detection and identification of keys (right).**

(Figure 4, on the right, shows a later stage of the process). The modeler then identifies an arbitrary button by “scribbling” over it with the pointer. This serves two purposes: specifying representative visual properties of a key, specifically its color, size, and position, and implicitly describing the background for all the keys. The user can then associate a specific symbol with this key, such as “1” or “#”. All keys can be specified in this way, or the system can be directed to detect and identify all keys automatically. Results can then be reviewed and modified by the modeler. Figure 4 shows the result for a sample cell phone, with call outs generated and laid out by the system.<sup>2</sup>

Currently the system can detect only buttons that are well-differentiated from the background, and can identify (i.e., associate a symbol with) only the 12 standard keypad keys (“0” through “9”, plus “\*” and “#”). Other keys can be detected but not automatically identified; naming these keys is left to the modeler. Once the relevant keys on the keypad have been identified, the system generates a specification of key names, sizes, and positions, identical in structure to that of the direct manipulation keypad builder interface.

In an informal test, we downloaded 12 images from the Web, of a quality comparable to Figures 1 and 4, of cell phones from different manufacturers and in different keypad styles. We ran the image processing algorithms on the keypads to identify the 12 standard keys, following the interaction described above. Results fell into three categories. For four of the cell phone images, the system failed to identify any keys at all. For two of the images, the

system identified 10 out of the 12 keys correctly. Errors involved the merging of two neighboring keys (e.g., “8” and “0”.) For the remaining six images, all 12 standard keys were identified correctly. The more successful results tended to have good resolution, low variation in background color, easily differentiated key borders, and few lighting artifacts. Figure 4, of a Motorola T193, is representative of an image processed successfully. Work needs to be done on making the image processing techniques more robust, but even in their current state they add value to the modeling process by reducing effort.

Once a menu hierarchy and keypad have been specified, model execution can proceed. Our modeling environment is designed as a set of extensions to the ACT-R system. The PM (perceptual/ motor) component of ACT-R has some bias toward desktop activities, such as selecting menu items with the mouse and navigating through windows and dialog boxes [1,3], so we added two extensions. Our first change involved increasing flexibility in the existing models of finger movements and the keyboard, to support finger movements to target keys of arbitrary size and placement. Our second change was a limited form of automated model generation from the menu hierarchy specification. ACT-R productions (rules) are not modified, but memory structures are automatically created (on the order of several hundred chunks) to reflect relationships between intermediate and terminal menu items.

The GOMS and Fitts’ law models co-exist with ACT-R in the same environment, sharing a common code base. We implemented a simple Fitts’ law simulator and a GOMS interpreter that use the geometry of the keypad specification for computing movement times. Fitts’ law movement operators are generated from the menu hierarchy, given a target menu item. Similarly, all of the methods and operators for the GOMS model except the Select-X method are automatically generated. Thus given a menu hierarchy, keypad specification, and target menu item, GOMS and Fitts’ law models can be built and run without further effort.

To summarize, the tools we have developed assist the developer in specifying new menu hierarchies, building descriptions of new or existing cell phone keypads, and automatically incorporating this information into specialized models. The process requires no programming or explicit model construction, from start to finish.

#### **User profiles and search**

Once models of menu traversal have been built, they can be applied toward improving menu hierarchies for end users. This is a key concern for developers who may be less interested in modeling details than in the pragmatic issues of increasing usability.

An evaluation of a menu hierarchy independent of usage patterns would be uninformative: different users choose different items, and items are chosen with varying frequency. In other words, different usage patterns favor

<sup>2</sup> This side of our research ran in parallel with but slightly later than the study. The image processing algorithms handle the Kyocera keypad without problems, but the numbers used for the study were measured by hand.

different designs. We can incorporate information about different user categories into an evaluation. We define a *user profile* to be a probability distribution over the set of terminals in the hierarchy (explained below). The *coverage* of a profile reflects the ratio of users it represents with respect to all users of the cell phone (some profiles will be more common than others), and the *usage* of a profile gives the average frequency with which items are chosen (e.g., two profiles may contain similar probabilities, but one may apply to users who access their cell phones several times as often as the other set of users.) As an example, imagine that 20% of users access two items equally, "Recent Calls" and "View All Contacts", on average three times a day. These users would have a profile such that in its probability distribution, those two item have probability 0.5 (all others 0.0); its coverage is 0.20; and its usage is 3 (a value that only becomes meaningful in the context of the usage values of other profiles.)

These concepts can be used to produce a very straightforward performance measure that represents the cost of traversing a menu hierarchy, over different possible user profiles, for all menu items that are accessed. Specifically, we define the expected cost of making item selections in a menu hierarchy as follows. Let  $T_h$  be the set of terminals in hierarchy  $h$ . Then

$$EC(h) = \sum_{t \in T_h} c_h(t) p(t) \quad (\text{Eq. 3.})$$

where  $p(t)$  gives the probability of the occurrence of a particular terminal  $t$  in hierarchy  $h$ , and  $c_h(t)$  gives an estimate of the cost of reaching that terminal in the hierarchy. This measure is used by an automated search algorithm to identify alternative designs of the menu hierarchy that improve user performance.

The starting point for the search algorithm is provided by the designer, who enters information about terminal item occurrences. The designer can enter these directly (e.g., such-and-such an action occurs four times per day, week, or month) through a simple fill-in form that is generated from a given menu hierarchy. Alternatively, the designer can provide such information for specific user profiles. The system searches through modifications to the menu hierarchy, producing alternatives that can be interactively browsed, as shown earlier in Figure 4, to be accepted, rejected, or modified.

To compute  $p(t)$ , one component of the expected cost measure, the system uses the information from the designer to construct a probability distribution of the occurrence of terminals in  $T_h$ . This distribution may be directly supplied by the designer as described above, in the form of item access frequencies. If profile-based information is provided instead (i.e., distributions of terminal occurrences conditioned on different user profiles), then an overall distribution is computed using Monte Carlo sampling, based on the usage and coverage values of the profiles.

To compute  $c_h(t)$ , the other component of the expected cost of traversing the menu hierarchy, we use our study results. For pragmatic reasons, we use the easiest metric we have available to compute cost, the linear regression given in Equation 2 (the GOMS or ACT-R model could have been used, though with a significant increase in processing time.) The factors that make the linear regression less appropriate for modeling do not apply here. Our choice for  $c_h$  means that  $EC(h)$  produces the expected duration of choosing an arbitrary menu item in  $h$ .

An automated, search-based modification of a menu hierarchy cannot arbitrarily rearrange its structure purely for efficiency. Changes must respect the semantic relationships between the items. That is, "Ringer Volume" is under the "Settings" category rather than vice versa for good reason. To avoid the difficulties of representing and reasoning about menu item semantics (we leave this for future work), we rely on two search operators that produce only small changes. For a target item with non-zero probability in some user profile, these operators can be applied:

- *Promote item*: This operator moves a target item to the beginning of its menu list, to reduce scrolling time.
- *Promote subtree*: This operator moves an ancestor of the target item up one level in the hierarchy, to reduce selection of intermediate items to reach the target.

An item or subtree rooted at an ancestor may only be promoted once. Even with these constraints, the search space size is exponential in the number of target items with non-zero probability in any profile (e.g. if all non-zero items in a user profile are in one menu list, then all permutations of these items will be considered.) Exhaustive search is thus impractical. A best-first search algorithm, however, gives good results after as few as 100 steps.

Lacking real user profiles, we can only illustrate the search procedure in practice, but our results are promising. Based on the Kyocera menu hierarchy, we defined random profiles of different sizes, where size refers to the number of non-zero probability menu items contained in the profile. The probabilities for each profile were drawn from a uniform random distribution and normalized. Because these profiles were randomly generated, we used only a single profile for the search, rather than composing arbitrary probabilities from different random profiles. Table 3 shows the results for user profiles of size 20, 30, and 40, each given 10 runs in a best-first search bounded at 500 steps. The cost values are means of the time estimates produced by the linear

| Profile size | Initial cost | Final cost | Savings |
|--------------|--------------|------------|---------|
| 20           | 7.325        | 4.530      | 37.5%   |
| 30           | 6.962        | 4.762      | 31.5%   |
| 40           | 7.009        | 4.940      | 29.4%   |

**Table 3. Improvement of menu traversal times.**

model. The last column gives the time savings in traversing the reordered menus, as a percentage of the duration of the traversals in the original menu hierarchy. Because these results are based on random probabilities of accessing menu items, rather than actual user experiences, they can only be viewed as suggestive.

#### FUTURE WORK

One aspect of our future work will involve testing with ACT-Simple [11], to see whether the simpler GOMS model might be automatically translated into an improved, more detailed cognitive model based on ACT-R. As obvious as this direction may seem now, we did not pursue it initially, partly because we did not expect the ACT-R and GOMS models to perform as well as they did, and partly because we thought of GOMS as a baseline against which we would demonstrate an ACT-R improvement. (That the ACT-R model produces inferior detailed predictions is disappointing, indicating that our understanding of the task is incomplete at the lower levels. We expect to address its shortcomings in follow-on work.) A comparison of different detailed models, including ACT-R, ACT-Simple, and other possibilities, is now appropriate at this stage.

#### CONCLUSION

In this paper we have described a set of evaluation concepts and tools to support cell phone menu design. Our work has both theoretical and practical implications. Novice user actions, learning, error recovery behavior, and generality across different devices are now areas ripe for further exploration. From a practical standpoint, developers have models that are ready for use—they are general enough that they do not require cognitive modeling expertise or programming skill to apply them to different traversal tasks, in different menu hierarchies, or on different cell phones. Our longer-term goals for this research include building an integrated environment to support the definition of new interactive environments, the application of modeling techniques to provide insights into usability issues [9], and the application of intelligent tools that can contribute information to aid design. We believe that as modeling concepts and techniques become more accessible to HCI developers, they will become increasingly significant in their contribution to improving interfaces.

#### ACKNOWLEDGMENTS

The authors wish to thank David Golightly and five anonymous reviewers for their helpful comments. This effort was supported by the National Science Foundation under award IIS-0083281 and by the Space and Naval Warfare Systems Center, San Diego. 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

1. Anderson, J., & Lebiere, C. (1998). *The atomic components of thought*. Mahwah, NJ: LEA.
2. Byrne, M. D., & Gray, W. D. (2003). Returning human factors to an engineering discipline: Expanding the science base through a new generation of quantitative methods—Preface to the special section. *Human Factors*, 45, 1-4.
3. Byrne, M. D. (2001). ACT-R/PM and menu selection: Applying a cognitive architecture to HCI. *Int. J. Human-Computer Studies*, 55(1):41-84.
4. Charney, B. (2003). Cell phone use to double. CNET News.com, <http://news.com.com/2100-1039-5060745.html>, August 6, 2003.
5. Gray, W. D., John, B. E., and Atwood, M. E. (1993). Project Ernestine: Validating a GOMS analysis for predicting and explaining real-world task performance. *Human-Computer Interaction*, 8(3): 237-309.
6. John, B. (2003). Information processing and skilled behavior. *HCI models, theories, and frameworks*, Carroll, J. (ed), SF, CA: Morgan Kaufman.
7. Kieras, D. (1999). A Guide to GOMS Model Usability Evaluation using GOMSL and GLEAN3, [ftp://www.eecs.umich.edu/people/kieras/GOMS/GOMSL\\_Guide.pdf](ftp://www.eecs.umich.edu/people/kieras/GOMS/GOMSL_Guide.pdf).
8. MacKenzie, I. S. (2003). Motor behavior models for human-computer interaction. *HCI models, theories, and frameworks*, Carroll, J. (ed), SF, CA: Morgan Kaufman.
9. Nichols, S., & Ritter, F. E. (1995). A theoretically motivated tool for automatically generating command aliases. In *Proceedings of CHI*, 393-400. NY, NY: ACM.
10. Ritter, F. E., and Young, R. M. (2001). Embodied models as simulated users: Introduction to this special issue on using cognitive models to improve interface design. *Int. J. Human-Computer Studies*, 55(1):1-14.
11. Salvucci, D. D., and Lee, F. J. (2003). Simple cognitive modeling in a complex cognitive architecture. *Proceedings of CHI*, 265-272. New York, NY: ACM.
12. St. Amant, R., and Riedl, M. O. (2001). A perception/action substrate for cognitive modeling in HCI. *Int. J. Human-Computer Studies*, 55(1):15-39.
13. Silfverberg, M., MacKenzie, I. S., and Korhonen, P. (2000). Predicting text entry speed on mobile phones. *Proceedings of CHI*, 9-16. New York, NY: ACM.