In D. Reitter & F. E. Ritter (Eds.), Proceedings of the 14th International Conference on Cognitive Modeling (ICCM 2016). University Park, PA: Penn State.

Towards a General Model of Repeated App Usage

Sabine Prezenski (sabine.prezenski@tu-berlin.de)
Department of Cognitive Modeling in Dynamic Human-Machine Systems, Technische Universität Berlin
10587 Berlin, Germany

Nele Russwinkel (nele.russwinkel@tu-berlin.de)
Department of Cognitive Modeling in Dynamic Human-Machine Systems, Technische Universität Berlin
10587 Berlin, Germany

Abstract

The main challenge of implementing cognitive models for usability testing lies in reducing the modeling effort, while including all relevant cognitive mechanisms, such as learning and relearning, in the model. In this paper we introduce a general cognitive modeling approach with ACT-R for hierarchical, list-based smartphone apps. These apps support the task of selecting a target, via navigating through subtargets positioned on different layers. Mean target selection time for repeated app interaction, learning and relearning behavior was collected in four studies conducted with either a shopping app or a real-estate app. The predictions of the general modeling approach match the empirical data very well, both in terms of trends and absolute values. We also explain how such a general modeling approach can be followed. The presented general model approach requires little modeling effort to be used for predicting overall efficiency of other apps. It supports more complex interface, as well.

Keywords: ACT-R; usability; apps; cognitive modeling; learning; relearning; updates; general model

Introduction

Numbers of smartphone apps are growing and so is the need for efficient usability testing methods. Cognitive models simulate human behavior and can in theory be utilized either as a supplement to, or instead of real user testing. To achieve this aim, especially in terms of costs and effort, it is crucial to develop valid cognitive models for specific tasks and app characteristics. These models should be written in a general manner, in order to minimize the effort to transfer them to other similar apps. Such general models could then be used to predict specific usability measures like efficiency. This would be particularly helpful in the prototyping phase of apps, where these models could be used instead of user tests. Moreover, the model traces could provide evidence on potential (cognitive) causes of usability problems that are not achievable with user tests.

Theory

Smartphone apps often support a limited amount of tasks and although apps for a great variety of tasks exist, the structures and functionalities of these apps are similar. Consequently, predicting usability of such apps with a general cognitive modeling approach would be useful and worthwhile. In order to provide meaningful predictions, user behavior should be depicted accurately by such general cognitive models. It is crucial that these models are written in a manner that transferability to other similar, but not identical apps, in terms of content and structure, is feasible with minimal effort. Such an approach implicates that not all cognitive mechanisms of users are represented by the models. In respect to transferability, simplifications are necessary for such an approach.

ACT-R has a modularized structure, resembling the architecture of the human brain. Specified modules handle different types of information, called chunks. Each chunk has slots; this is where the smallest pieces of information are stored. The different modules interact via specialized buffers. Visual information is processed by the visual module and its two buffers (visual-object and visual-location). Motor movement is controlled by the manual module and its manual buffer. The declarative module serves as the systems memory and retrieved information from memory is stored in the systems retrieval buffer. The imaginal module and its correspondent buffer are required for learning new information. The steering of the model is governed by the goal module and buffer. The procedural module connects the modules and selects (production-) rules that steer the model behavior. A production is selected and executed, if the states of the buffers are met. The production then alters the states of the modules. Subsymbolic processes are also addressed in ACT-R. If a production requests a chunk and two chunks match the request, then the chunk with the higher activation level is selected. The activation level of a chunk depends on how long ago the chunk was created, on how often it was used and on when it was last accessed. Other parameters are the latency factor which influences the duration until a retrieved chunk is available in the retrieval buffer and the duration until a retrieval failure occurs. The later is also manipulated by the retrieval threshold parameter.

Box 1: A brief introduction to ACT-R

Hierarchical, list style apps are a common type of apps. They are often designed to support the task to find and select a target by navigating through different layers and selecting a subtarget on each layer. This paper presents a general cognitive model of a user interacting with hierarchical list style apps. The model covers repeated interaction, thus investigating learning and relearning effects.

Some cognitive modeling approaches addressing the usability of HMI already exist. The most prominent is CogTool (John, Prevas, Salvucci, & Koedinger, 2004). This is a rapid prototyping tool that enables the creation of cognitive models and predicts execution times for predefined task. But important aspects for the usability of apps such as version updates and learning behavior cannot be modeled with CogTool. A main objective of our work is to develop a model that learns through experience with the interface. A modeling procedure that is strong in

representing learning mechanism is the cognitive architecture ACT-R (for a brief outline of the mechanisms of ACT-R see box 1). Successful ACT-R models of menu and mobile interaction exist. The following aspects of these models are used our model. In an eye-tracking study using desktop computers, Byrne (2001) showed that menu search can be modeled with ACT-R. The fact that this model reads the menu from top to bottom is adopted in our model. In a study on learning of mobile phone usage for elderly novice users (Das & Stuerzlinger 2007), the performance increase in the model was due to the successful recall of locations of keys. Our model also uses the retrieval of locations as a learning mechanism. St.Amant, Horton, & Ritter (2007) developed a model that predicted time on task for expert users searching in hierarchical menus with a feature phone. Their assumptions that experienced users navigate with parent child chunks through hierarchies is implemented in a specialized chunk of our model.

The aim of this paper is to develop and test a general cognitive model with ACT-R that predicts user behavior during repeated interaction. Thus, the model incorporates learning and relearning mechanisms. The modeled task is repeated target selection with different hierarchical list apps.

Methods

Four studies were conducted with either a shopping app (*shopping 3-4, shopping 4-3)* or a real-estate app (*house apartment, apartment house*). Both apps are custom-designed android apps. The empirical studies are presented elsewhere in greater detail (Prezenski, Lindner, Moegele, & Russwinkel, in preparation.; Prezenski & Russwinkel, 2014). Since this paper focuses on the modeling approach, only a brief outline of the apps and the study procedure will be given. See figure 1 for an overview of the apps.

The main functionality of the shopping app is to compose a shopping list. To place products on the list, navigation through various stores and product categories in the menu is required. The real-estate app allows the selection of search criteria for real-estates, such as the number of rooms or the city district. Again, the criteria can be found by navigating through different categories. Both apps are multi-layer hierarchical list apps with variations in menu depth. The main functionality of the apps is target selection via navigating through a number of layers (see figure 1). On each layer a subtarget has to be preselected. For each target, there is only one correct path of subtargets leading to the target. Two versions of the shopping app are used: one with three and one with four layers of menu depth for all targets. As illustrated in figure 1, the path leading to the target e.g. *alcohol free beer* differs between the two versions. The real-estate app has a mixed number of layers (either three or four layers per target). Furthermore, the real-estate app is adaptive. Depending on preselection the paths leading to targets and the position of some subtargets changed. As can be seen in figure 1 the path leading to the target *lawn* differs if either house or apartment has previously been selected.

Figure 1: Screenshots of the apps with the modified paths leading to the targets for the different versions (shopping app) or different previous selections (real-estate app).

Task

Participants repeatedly selected targets using the apps installed on a Google Nexus 5 smartphone, running android 4.1.1. Targets were read to the participants and after selecting the target, participants were required to navigate back to the first layer of the app. In all four studies there were four runs, each run required the participants to select a number of targets. Participants of the studies *shopping 3-4*, and *shopping 4-3* had to select nine targets (products) per run. The same targets were used for all four runs. After the second run the version of the shopping app was updated, either a layer was added (*shopping 3-4*) or removed (*shopping 4-3*). Thus, the paths leading to the targets were the same for the first and second layer but were altered from the third layer on. Participants of the studies with the real-estate apps had to select six or seven targets (criteria) per run. Some of the targets were the same for all runs, e.g. numerical criteria such as *the rent* remained the same for all four runs. Others, like *the city district* varied between all four runs. Participants of the study *apartment house* searched for an apartment in the first two runs and then switched to searching for a house. The order was reverse for participants of the study *house apartment*. Due to the adaptive character of the real-estate app, the pre selection of house or apartment altered the position of the numerical criteria (e.g. the number of rooms) and also changed the path leading to lawn. This path differed for house and apartment from the second layer on.

Model

The data obtained with the studies *shopping 3-4* and *shopping 4-3* was utilized to develop the main model mechanisms and a first ACT-R model. The subsequent studies *house apartment* and *apartment house* were designed for two reasons: First, to test whether the model can predict data obtained with a different app and second, to ensure that the model mechanisms are held in a general matter. Thus, the model incorporates mechanisms for handling variations in depth within an app, changes in paths from varying layers on and variations in locations of targets and subtargets. The task of repeatedly selecting targets in multilayer applications is captured in the model.

Table 1: Examples of the chunk types of the model

meaning chunk NAME "SEARCH" OBJECT SEARCH	*association chunk* OBJECTS HOUSE CATEGORY SEARCH	*path chunk* FIRST SEARCH SECOND WHAT THIRD RENT FOURTH HOUSE TARGET-IM HOUSE COUNT FOUR
chunk with location SCREEN-POS VISUAL LOCATION35-0-0 VALUE "House" COLOR BLACK HEIGHT 10 WIDTH 28 TEXT T	*goal chunk* STATE PREPARECLICK SUBTARGET "SEARCH" FINALTARGET "HOUSE" TARGET-MEANING HOUSE MENUDEPTH FOUR IMMOLIST ("MOABIT" ....) MENUDEPTHLIST (THREE ...)	

Summary of Main Mechanisms[1]

Without prior experience with the specific target, visual attention is directed to the top of the page. For each visual processed word, a retrieval request for a *meaning chunk* containing the word as string and as meaning is made (see table 1 for examples of the chunk types used in the model). Navigation through the application is achieved via world knowledge, which consists of associations between two words (*association chunk*). For each read word the attempt to retrieve an *association chunk* with the target is made. If an *association chunk* containing the current word and the target is retrieved, a *path chunk is built*, holding the path leading to the target in the imaginal buffer and the word is selected. A *path chunk* consists of the slots *first, second, third* and *fourth* for the subtargets. The slot *target-im* holds the target word. The *count* slot of the *path chunk* holds information on the current menu depth and is changed if a different subtarget is required.

With experience with the specific target, navigating to this target is realized via the *path chunks* previously built. After a successful retrieval of a *path chunk* a chunk with the location of the relevant subtarget in the path is requested. The retrieved location is visually inspected and the subtarget is selected.

Model steering

Learning mechanisms are incorporated in the model. Furthermore, the model can handle a number of changes to the interface; such as version updates influencing all targets and smaller changes affecting only some targets *Model steering* is implemented with the goal in mind to reuse or extend the model for other applications. Thus, simplifications of some cognitive mechanisms and special chunk slots to account for interface variations are used. Model steering is realized via a *count* slot in the imaginal buffer, which holds the current depth and via different slots in the goal buffer. The *menudepth* slot holds information on changes in depth for the current target (e.g. number of layers leading to the target). The model can handle varying and constant depth values Currently, mechanism exist for a constant depth of three and two layers and for depth changing from three to four and vice versa, with the path leading to the target altered either from the second or from the third layer on. The *menudepth* slot is used to differentiate between strategies for the last versus the other layers in the path leading to the target. The *menudepth* slot is also required after an error in the path leading to the target is noted. From the affected layer on different *path chunks* are built and retrieved. Therefore, the *menudepth* slot holds the assumption that after an error in the path is noted, the erroneous (old) *path chunks* are used only for the layers that have not changed. The *menudepth* chunk furthermore holds knowledge about which layer is the final layer for each target. The *errorpath* slot in the goal buffer holds the knowledge about an occurred change in the path leading to any target; it is not reset between different targets. The *finaltarget* and the *subtarget* slot hold the target and the subtarget as a string. These two slots are used to determine if the target has been found and also required for a superficial visual search utilized on the last page and for researching a subtarget.

Mechanisms en-detail

Initiation In the beginning of each run, the production *start* requests a *meaning chunk*. The building of a *path chunk* is initiated. The production *meaning-in-goal* then copies the retrieved meaning of the target into the slot *target-meaning* of the chunk in the goal buffer and into the *target-im* slot of the chunk in the imaginal buffer. Then, a retrieval request for a *path chunk* leading to the target is initiated with variations[2] of the production *look-for-path.*

Association approach Without prior experience with the specific target, a *path chunk* leading to the target is not retrievable and the production *change-strategy* fires, followed by *find-word* and *reading-word*. Visual attention is directed to the top of the page (to the highest location bellow the current visual attended location) and this location is visually processed. Variations[3] of the production *process-word* then visually encode the current word. For all layers, except the last layer, a request for a *meaning chunk* holding the meaning of the current word is initiated. If such a chunk is found the production *searching association* then initiates the search for an *association chunk* containing the current word and the target. If such an *association chunk* cannot be found, the production *no-association-found* clears the visual buffer and the search continues with the production *find-word*. If an *association chunk* is retrieved, variations[4] of the

[1] The model can be downloaded at https://depositonce.tu-berlin.de/handle/11303/5548.

[2] The variations of *look-for-path* consider two aspects: First, whether or not there was an error in the path and second, the differences in menu depths. This ensures that for a detected change in menu depth the old (misleading) path is not retrieved.

[3] There are variations of *process-word* for the last layer and for the other layers except the last. The variations consider the value of errorpath slot in the goal buffer and the value of the count in the imaginal buffer.

[3] Variations of *association- found* depend on the value of the count slot of the imaginal buffer.

production *association-found* update the *path chunk* in the imaginal buffer. If, for example the first subtarget in path is found, then the value of the *count slot* in the *imaginal buffer* is changed from *one* to *two*. Furthermore, the *first slot* of the *path chunk* is filled with the current *subtarget*, which is also copied into the *subtarget slot* of the goal buffer. A cursor move is initiated and the productions *prepare-click* and *click* initiate the motor movements to press the button. The productions *waiting-click* or *waiting-last-click* (for the final click, in order to initiate the backing procedure) let the model wait until the *manual buffer* is free. After the manual buffer is free a variation of the production *look-for-path* fires again. For the last layer, the elaborate procedure of reading from top to bottom and searching for *association chunks* is replaced by a superficial visual search procedure. If the word in the visual buffer and the word in the slot *finaltarget* of the chunk in the goal buffer are different, a variation of the production *process-word-last-page-wrong* will fire. Via the production *find-word* the next word is searched. If they are the same a variation[5] of *process-word-last-page-correct* will copy the last slot of the path into the *path chunk* in the imaginal buffer, raise the *count slot* and change the value of the *subarget slot* in the chunk in the goal buffer. A cursor move is initiated and the productions *prepare-click* and *click* press the button with the target.

Path Navigation If a *path chunk* leading to the target is retrieved a variation of the production *found-that-path*, depending on the value of the *count slot* in the imaginal buffer, fires. This production copies the value of the relevant slot (e.g. the *subtarget*) from the *path chunk* in the retrieval buffer into the *path chunk* in the imaginal buffer and changes the *count.* The production *find-location* requests for a *meaning chunk* of the relevant subtarget. The production *found-location* indicates that a location was retrieved and the visual attention is moved to the retrieved location. Then the visual buffer and the *subtarget* slot of the chunk in the goal buffer are compared. If they are the same, then the retrieved location is correct and the production *checking-match* fires, followed by *click-location* and *waiting-click*.

Modified Interfaces In the following subsection an overview on mechanism dealing with the modified interfaces is given, for a detailed description see box 1.

The retrieved location is visual inspected and the subtarget is not found at the retrieved location, either because there is a different word, or no word at the retrieved location. This is indicated by the productions *checking-no-match* or *checking-empty*. A visual search for the subtarget is then initiated with the production r*ead-top-to-bottum-again-2*. Visual attention is directed to the top of the page and the production *scan-1* encodes the visual-location. If the word in visual buffer is the subtarget then *scan-correct* fires otherwise *scan-incorrect* moves the visual attention to the next highest word. If the subtarget is found it is selected via *prepare-click* and *click*. If the visual search via *scan-1* and *scan-incorrect* does not lead to the subtarget and the bottom

1. *Depth changes from three to four layers; the path is different from the third layer on.* An update adds a new layer to all targets of the shopping app, e.g. the old path for *alcohol free beer* is *1.stores 2.drinks 3.alcohol free beer* the new path is *1.stores 2.drinks 3.beer 4.alcohol free beer.* In the third run (after an update), *alcohol free beer is searched* on the third layer. But the retrieved location does not contain *alcohol free beer*. The third layer is rescanned from top to bottom in search for *alcohol free beer,* without success. The value of *errorpath* slot in the goal buffer is changed to true. The third layer is then read from top to bottom and for each word an attempt to retrieve *an association chunk* is made. The model visually encodes *beer* and an *association chunk* is retrieved The third slot of the *path chunk* in the imaginal buffer is assigned the value *beer* and beer is selected. On the fourth layer, the attempt to retrieve a *path chunk* that leads to *alcohol free beer* with the slot four having a value is made. Such a chunk cannot be retrieved. The superficial search approach for the last page will be used and directly search for *alcohol free beer is initiated*. For the next product, navigation is realized with the old *path chunk* for layer one and two. For the third and fourth layer the attempt to retrieve a *path chunk* with four layers will fail. The association approach or for the last page the superficial search will be used. In the fourth run the correct *path chunks* for all layers can be retrieved.

2. *Depth changes from three to four layers; the path is different from the second layer on, mixed list*. The path for lawn in the real-estate application changes depending on preselection. If house is preselected, the path is 1.search, 2.garden, 3.lawn and changes to 1.search 2.more 3.garden 4.lawn if apartment is preselected. On the second layer the subtarget garden is not found at the retrieved location. A rescanning of the page is unsuccessful. As in 1) an errorpath in the goal buffer is noted and the approach is changed to reading from top to bottom and searching for association chunks. A new *path chunk* is then built. For the third and fourth layer the attempt will be made to retrieve a *path chunk* with four layers leading to lawn and then utilize the reading and association approach or the scan approach for the last page. For the next target, the value of the menudepth slot in the goal buffer indicates constant depth; therefore it is not influenced by the error in the path. In the fourth run, a *path chunk* for lawn can be found.

3. *Depth changes from four to three layers; the path is different from the third layer on.* An update removes a layer for all targets from the shopping list app. The path for *alcohol free beer* changes from *1.stores, 2 drinks, 3.beer 4.alcohol free beer* to *1.stores, 2.drinks 3.alcohol free beer.* On the third layer the subtarget *beer* is not found at the retrieved location. The third layer is rescanned from top to bottom in search for *beer* and the target *alcohol free beer is found*. The errorpath slot in the goal buffer is changed to true. The third slot of the *path chunk* in the imaginal buffer gets the value *alcohol free beer* assigned and the target is selected,. For the next product navigation is realized with the old *path chunk* for layer one and two. For the third layer the attempt to retrieve a *path chunk* with three layers will fail. The superficial search will be used for the last page and the building of *path chunks will be completed*. In the fourth run the correct path chunks can be retrieved.

4. *Depth changes from four to three layers; the path is different from the second layer on (special case).* First apartment is preselected. Thus, the path for lawn in the real-estate application is *1.search 2.other 3.garden 4.lawn*. Then house is preselected and the path is *1.search, 2.garden, 3.lawn*.On the second layer, the subtarget other is not at the retrieved location. The page is rescanned and *other* is found and selected. On the third page garden is searched for unsuccessfully. The value of the errorpath slot in the goal buffer is then set to true and the model goes back to the second page. A new *path chunk* is built from the second page on via association chunks and reading top to bottom. For the next target, the value of the menudepth slot in the goal buffer indicates constant depth; therefore, it is not influenced by the error in the path. In the fourth run, a *path chunk* for lawn can be found.

Box 2 : A description on how different changes in the interfaces are processed by the model.

of the page is reached, an *errorpath* will be noted in the goal buffer. An *errorpath* in the goal buffer will also be noted, if the target is found while scanning for a subtarget (via *scanning-path-is-wrong*). After the *errorpath* slot in the goal buffer is set to true and the subtarget has not been found, the production *error-in-path-2* resumes the visual attention to the top of the page and via *find-word, reading-word* and *association-found* a *path chunk* is build in the imaginal buffer. The changed value of the *errrorpath* slot will stay in the chunk in the goal buffer for all further model runs. Therefore, if the *errorpath* value in the goal buffer is set to *true*, a wrong *path chunk will not be retrieved*. Either no *path chunk* is retrieved and navigation is implemented via *association chunks*, or *path chunk* are utilized only as long as they are correct. For example, if the *path chunk* is helpful for layer one and two – navigation with this chunk for these layers is implemented, but different *path chunk* are sought for on the third layer.

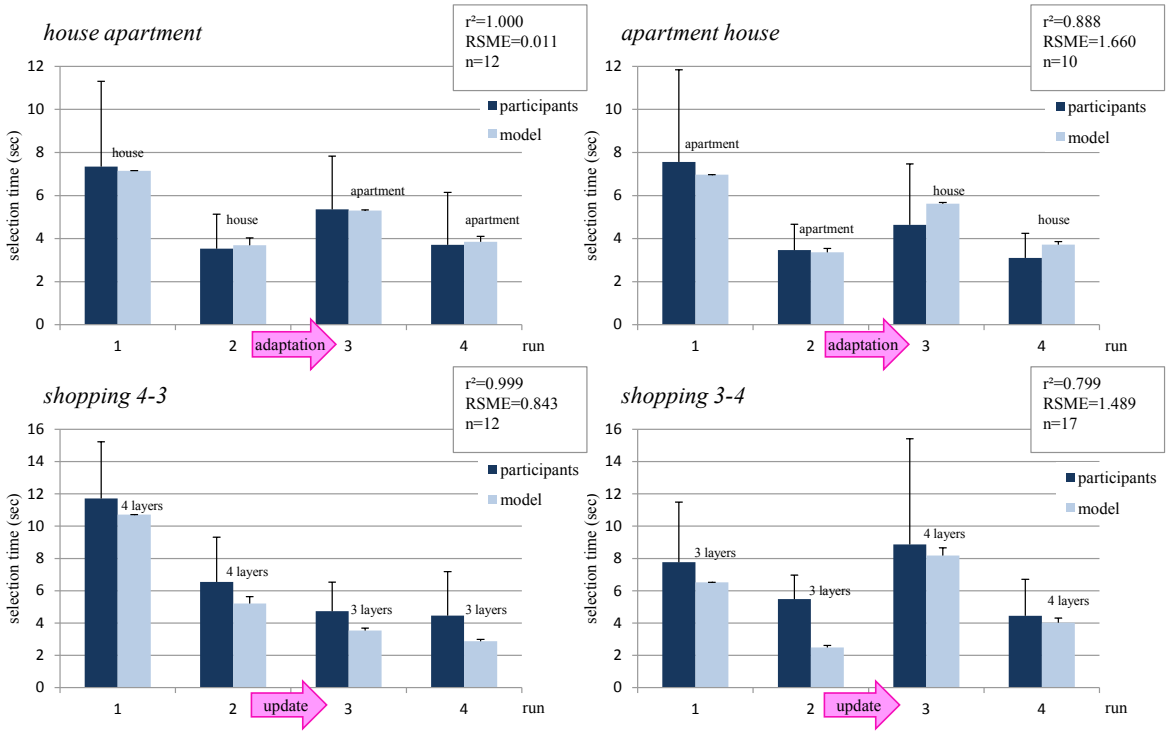

Figure 2: Mean target selection time for the four different studies for the modeled and empirical data.

Results

Data processing

The empirical data comprises of four studies, with a varying number of student participants ($10 < n < 17$). The model was run 10 times per study. Target selection time (for both model and empirical data) is defined as the time between the selection of the target and the selection of the first subtarget. Extreme values were excluded from analysis. Mean target selection time for each target was calculated and averaged over the runs (six to nine targets per run). Mean target selection time for each run was calculated and averaged over the participants. Model and empirical data were compared via goodness of fit indices and qualitative analysis of graphs.

Model parameter

The latency factor (lf) and the retrieval threshold (rt) parameter were fit to match the data of the study house apartment. They are set to the values: lf = 0.1 and rt = -1.5.

Comparison of modeled and empirical data

The model provides a good to very good fit to the data, see figure 2 for the comparison of the mean target selection times of the empirical and modeled data. The main trends of the four studies are mapped in the modeled data, as are most of the absolute values.

1. The *house apartment* study reveals a decrease of mean target selection time followed by an increase and again a decrease. This is exactly represented in the modeled data; $r^2 = 1.0$. The absolute values of the modeled and empirical data are also very close; RSME = 0.011.

2. The *apartment house* study reveals the same pattern (decrease, increase and decrease); $r^2 = 0.888$. The mean target selection time predicted by the model is slightly lower in the first two runs and a bit higher in the last two runs; RSME = 1.660.

3. The empirical and modeled mean target selection time of the *shopping 4-3* study indicate a decrease from run 1 to run 4, with the decrease leveling out towards the last run. The magnitude of the decrease is very similar for both

datasets; r^2= 0.999. In all runs the mean target selection times predicted by the model are lower than those of the participants; RSME = 0.843.

4. For the *shopping 3-4* study, both datasets show a pattern of a decrease, followed by an increase and a final decrease. The modeled increase in mean target selection time between run 2 and 3 is greater than the increase found in the empirical data. The magnitude of the other variations are similar for the empirical and modeled data; r^2 = 0.799. The mean target selection time predicted by the model is lower than that of the participants, especially in the third run; RSME = 1.489.

Summary

In summary, all trends found in the empirical data are predicted by the model, with a high r^2 = [0.799; 1.0] for all four studies. Decreases and increases in target search time are predicted by the model. Therefore, the model provides information on learning and relearning effects for different applications. Moreover, the absolute values are met by the model for the majority of data points, with RSME = [0.011; 1.660], values which are lower than the average STD of the empirical data. Hence, the model can appropriate depict mean user behavior at different time points during a repeated target selection task.

Discussion

The modeling approach provides accurate predictions of the data from four studies with two different apps. Efficiency, learnabilty and the impact of updates or of adaptivty are predicted by the model.

Only a few steps are necessary to alter the model for a different app; world knowledge needs to be provided in form of *association chunks* and reading ability as *meaning chunks*. Furthermore, a compilation of the *targetlist*, containing the targets and of the *menudepthlist*, containing the menu depth of the targets before and after an update, is required.

The model can easily be extended to other list-style hierarchical apps with different menu depths than 3 and 4 layers and to apps with different switches in the number of layers. To do so, variations of the productions *process-word*, *process-word-last-page* and *look-for-path* are required.

Plausibility of Modeling Decision

A general modeling approach facilitates the adaptability of the model. To achieve this, partially simplified assumptions have been made. This applies especially to the *menudepth* slot, which offers a technical solution for the handling of varying menudepth. This slot contains meta-knowledge of the model, in other words knowledge about what kind of update occurred. The *menudepth* slot holds information if the update relates to the entire menu structure or if it relates to individual paths. Furthermore, the *menudepth* slot is useful to identify the last page. It is plausible to assume that users know if the current page is the last page. However, users are likely to obtain this knowledge with the help of visual features. Since these in turn significantly differ between apps, it is useful for a general approach to detect the last page in a simplified manner.

The reading direction of the model is always directed from top to bottom and each item is processed, this is a further simplification. It is possible to model visual processing of menus more precisely (Bailly, Oulasvirta, Brumby and Howes, 2014). Since, the goal of our work was to predict average mean search time for items, our chosen simplification of visual processing was sufficient.

Another simplified assumption is, that the imaginal holds the count of the current page. This is done only for practical reasons to make model steering easy and adjustable to different updates. Furthermore, it does not affect overall item search time.

Limitations and Further Steps

To use the model approach for usability testing, a remaining obstacle is the need of prototyping the interface. In order for the ACT-R model to interact with the application, a copy of the app in Lisp is required. To avoid this effortful step, we are working on a tool called ACT-Droid (Doerr, Russwinkel, & Prezenski, 2016).With ACT-Droid android apps can be connected directly with ACT-R models. This tool is will reduce the practical hurdle of testing usability of apps with cognitive models further. Further steps to improve the model are to replace the mouse movements in the model with touch movements, as provided by ACT-Touch (Greene & Tamborello, 2013). Besides, an in-depth validation of the model with eye tracking data is planned. A new study should also investigate if the model can predict average click times for each menu layer as well. The empirical data of the current studies, do not allow such predictions, due to the study design. Nevertheless, on an individual level the model and empirical data show, that after an update (shopping app) and after an unexpected adaption (real-estate app) search time increases.

We are also intending to look into how far mechanisms of the current model can be used to develop a model for hierarchical apps with icons instead of text.

Moreover, our general modeling approach for hierarchical list-style apps is not only useful for such apps. It is well suited to predict the average user search time for all kinds of list-based interfaces that spread semantically related subtargets across multiple layers.

References

Bailly, G., Oulasvirta, A., Brumby, D. P., & Howes, A. (2014). Model of visual search and selection time in linear menus. *Proc. of the 32nd Annual ACM Conference on Human Factors in Computing Systems - CHI '14, (pp.* 3865–3874)**.** Toronto, Canada: ACM.

Byrne, M. D. (2001). ACT-R/PM and menu selection: applying a cognitive architecture to HCI. *Int. J. Human-Computer Studies*, 55, 41-84.

Das, A., & Stuerzlinger, W. (2007). A cognitive simulation model for novice text entry on cell phone keypads. In *Proc. of the 14th European Conference on Cognitive Ergonomics*, (pp.141-147). London, UK: ACM.

Doerr, L.-M., Russwinkel, N., & Prezenski, S. (in submission). ACT-Droid: ACT-R Interacting with Android Applications. In D. Reitter & F. Ritter (Eds.), In *Proc. of the 14th Int. Conference on Cognitive Modeling.* Pennsylvania, USA.

Greene, K. K. & Tamborello, F. P. (2013). Initial ACT-R extensions for user modeling in themobile touchscreen domain In *Proc. of the 12th Int. Conference on Cognitive Modeling.*(pp.348-353)Ottawa, Canada.

John, B. E., Prevas, K., Salvucci, D. D., & Koedinger, K. (2004). Predictive human performance modeling made easy. In *Proc. of CHI´04*, (pp. 455–462). New York, USA: ACM.

Prezenski, S., Lindner, S., Moegele, H., & Russwinkel, N. (in preparation). Archetyping the User.

Prezenski, S., & Russwinkel, N. (2014). Combining Cognitive ACT-R Models with Usability Testing Reveals Users Mental Model while Shopping with a Smartphone Application. *Int. J. on Advances in Intelligent Systems*, 7(3), 700–715.

St.Amant, R., Horton, T.E., & Ritter, F.E. (2007). Model-based evaluation of expert cell phone menu interaction. *Transactions on Computer-Human Interaction*, 14(1), 1-14.