Parameter Selection in Keyboard-Based Dialog Boxes

First Author Name (Blank if Blind Review) **Second Author Name (Blank if Blind Review)**

ABSTRACT

Recent keyboard-based alternatives to WIMP interfaces do not have good support for commands that require multiple parameters. We remedy this by extending a previous design by mimicking dialog boxes to provide good visual feedback while still keeping the advantages of keyboard input. A laboratory study showed the new technique to be competitive with dialog boxes on speed and error rate, but strongly preferred to dialog boxes by experienced command line users. This is a marked improvement over the previous design, which was also preferred by the target user group but did not compete performance-wise with dialog boxes.

Author Keywords

Command-line interfaces, dialog boxes, WIMP

ACM Classification Keywords

Blah blah blah

INTRODUCTION

Many systems have been introduced to allow keyboard input to augment or replace standard WIMP interfaces, which have well-documented benefits and drawbacks [2,7,8,9,10]. The goal of these keyboard-based systems is to increase satisfaction and performance for experienced command line users. Examples include Quicksilver [1], Enso [4], Inky [5], GEKA [3], and Ubiquity [6]. They provide well thought out ways to specify commands, but not as much work has gone into how parameters are specified, especially for commands with multiple parameters. In WIMP interfaces, parameters are often specified through dialog boxes, especially multiple parameters. Evidence suggests that experienced command line users strongly dislike dialog boxes [3]. Quicksilver and Enso support only one parameter. Ubiquity and Inky support multiple parameters, but use imprecise syntaxes that can make parameter entry confusing, and they are designed specifically for simple web-based commands, which have smaller and simpler sets of parameters. GEKA has a keyword-based parameter system in which any number of parameters can be precisely input in any order, but it showed lower performance than existing WIMP dialog boxes [3].

We describe a new keyboard-based method, DBOX++, for specifying parameters that allows the input of any number of parameters in any order. Our goal is to increase performance, improve experience, and minimize cognitive load by providing graphical feedback that looks and behaves very much like the dialog boxes users are familiar with, but which has the speed of a command line interface. Our laboratory experiment with experienced command line users showed that our new method is preferred to and performs competitively with WIMP dialog boxes.

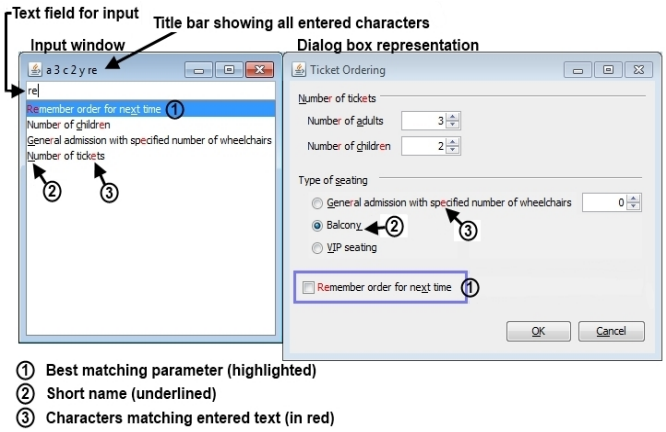

Figure 1: DBOX++ graphical feedback. The characters "a 3 c 2 y re" have been entered. This sets the values of Number of adults (to 3), Number of children (2), and Type of seating (Balcony). The final bit of input, "re", is selecting which parameter will be set next.

DBOX++ DESIGN

DBOX++ is based on GEKA. It makes minor changes to the GEKA command language and replaces its graphical feedback with a modified version of the command's existing dialog box. These changes should give DBOX++ a strong performance advantage over GEKA and make it competitive with WIMP dialog boxes.

Command language

Multiple parameter entry utilizes auto-completion to quickly specify parameter name and value pairs. Example parameter specifications for the **`print`** command include:

> **`pages 2`** – *set the value of* **`pages`** *to* **`2`**
> **`printer downstairs`** – *set the value of* **`printer`** *to* **`downstairs`**
> **`downstairs`** – *because* **`downstairs`** *can only be a value for* **`printer`**, *this sets* **`printer`** *to* **`downstairs`**

For parameter name selection, a ranked list of possible matches is generated after each character is typed. Typing more characters refines the list. Pressing **SPACE** accepts the

CHI 2011, May 7–12, 2011, Vancouver, BC, Canada.

top-ranked match and moves on to value entry. If a parameter has a discrete set of possible values, the value is selected using the same auto-complete mechanism. Otherwise, the value is typed in its entirety. Pressing **SPACE** again accepts the value and allows the user to specify another parameter. If the top-ranked parameter name has a Boolean value, pressing **SPACE** toggles the value and immediately moves on to another parameter.

Additionally, each parameter has a short name, which is a short sequence of characters that unambiguously identifies the desired parameter name.

Auto-completion uses the same four categories described by Hendy et al. for GEKA to order possible matches: *exact match*, *prefix match, substring match, subsequence match*. Matches within each category are sorted alphabetically. The up and down arrow keys scroll through the ranked list.

Graphical feedback

There are two components to DBOX++ graphical feedback, the input window, which closely resembles the graphical feedback in GEKA, and a new dialog box representation, which provides additional visual feedback. Figure 1 shows these two components. The input window has a text box where input is typed and a list of possible matches to the input is shown. The best match is highlighted and appears first. For all entries, the matching characters are in red. The short name is underlined. The dialog box representation provides graphical feedback identical to a command's WIMP dialog box. Because users are already familiar with this dialog box, we expect that displaying it will allow users to quickly identify parameter names they want to use.

The best matching parameter highlighted in the input window is also shown with a blue circle in the dialog box. The dialog box also has the short name for each parameter underlined and characters that match the entered text are in red. This gives users the same information whether they look for feedback at the parameter list or the dialog box. Highlighting also allows users to quickly verify that the parameter they want is selected. If the best matching parameter is located in a sub-dialog box or a different tab within the dialog box, the dialog box representation is automatically updated to show the location with that parameter.

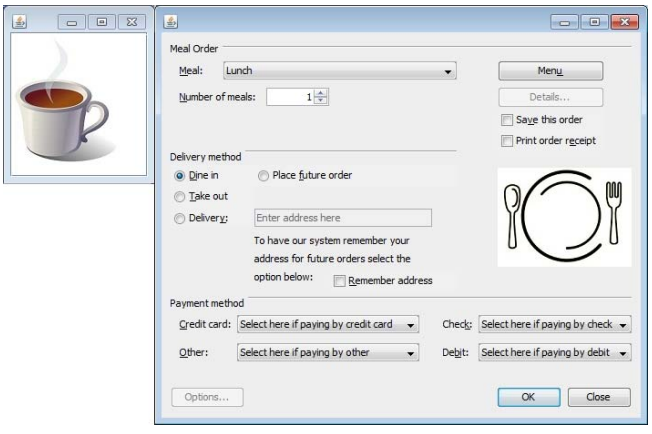

Figure 2: Experimental environment in the dialog box condition. This is the short task for the order food command. The left window shows that the parameter "beverage" must be set to "coffee." The beverage options are located in a sub-dialog accessed by the "Menu" button.

LABORATORY EXPERIMENT

We conducted a laboratory experiment to evaluate DBOX++ against the goals of high performance and preference by experienced command line users when compared to dialog boxes. We included a GEKA condition to compare DBOX++ to existing keyboard-based interfaces.

Experimental tasks

The tasks were to specify parameters for one of four commands. This was done in three conditions: DBOX++, GEKA as implemented by Hendy et al. [3], and traditional dialog boxes. With traditional dialog boxes participants could use any combination of mouse clicks, **TAB** key navigation, or keyboard mnemonics.

The experiment tested both familiar commands from a standard application and unfamiliar commands that we invented. It also tested simple commands, specified in a single dialog box, and complex commands, which involved multiple tabs and sub-dialog boxes. The four commands are listed in Table 1.

	Familiar	Unfamiliar
Simple	`Insert table`	`Order tickets`
Complex	`Print`	`Order food`

Table 1: Four commands used in the experiment

For the two familiar commands we implemented Word 2003 replica dialog boxes (including parameter names). All of our participants reported that they had used these dialog boxes in Word 2003. O`rder tickets` was created for this experiment; it has the same number of parameters and a similar dialog box layout to `insert table`. O`rder food` was similarly analogous to `print`. The two respective dialog boxes are shown in Figures 1 and 2.

For each command in the experiment, we created a short task, in which only one parameter value was to be specified, and a long task, in which four values were to be specified. For the complex commands, both the short and long tasks required the use of at least one sub dialog. Thus, there were eight different task combinations.

As in the GEKA study [3], all tasks were prompted using image-based descriptions to avoid biases introduced by using text descriptions. Figure 2 shows the prompts for the long `order food` trial. For each task, the command was pre-selected in the interface. The participant needed only to select the specified parameter values. This was done to isolate parameter selection times.

Participants

There were 12 participants (3 females). In a pre-screening questionnaire all reported having command line experience and correctly answered at least two of three command line

knowledge questions. Participants received $20. A $5 bonus was offered to the top third fastest participants to motivate quick and accurate performance.

Procedure

Each participant completed a single two-hour session. A session began with an introduction to the experiment. Participants then completed all eight tasks in a particular condition before moving on to the next condition. The presentation order of the three conditions was counterbalanced. Each condition began with an introduction to the interaction method used in that condition and a practice block in which each task was completed once. During a practice block, participants could ask questions and refer to printouts of which parameter each of the task images referred to. No aids were permitted during experimental blocks.

Presentation order of the four commands was randomized but remained constant across the three conditions for each participant. The order of the two task lengths was similarly randomized across participants. During each condition, participants completed one trial for each of the two task lengths for a command and then repeated this pair four more times before moving on to the next command. Completion time was recorded for each trial from when the participant dismissed a begin-task prompt (by clicking the mouse or pressing a keyboard button) to when the task was successfully completed.

An error occurred if a participant selected **OK** in the dialog box condition or pressed **ENTER** in the GEKA or DBOX++ condition with an incorrect set of parameter values selected. If a participant made an error during a trial, a dialog notified the participant that an error was made and the task had to be repeated until it was successfully completed. After all trials for a one command, participants took a 30 second break. Between conditions, participants took a 2 minute break.

To end, a questionnaire and interview completed the study.

Design

The experiment used a mixed factor design: 3 (interface) x 2 (command complexity) x 2 (command familiarity) x 2 (task length) x 5 (repetition) x 6 (presentation order). All factors were within-participant except for presentation order, which was between-participant control variable. Bonferroni corrections were used for all pairwise comparisons. When sphericity was an issue, Greenhouse-Geisser corrections were used, which can be identified by non-integer df.

Note on experimental design and participants

The focus of the DBOX++ design and the evaluation was on experienced command line users, but we were also interested in exploring how DBOX++ would be received by users without command line experience. Our initial design thus included experience as a factor; we ran 12 participants with no command line experience. We saw potential trends due to experience level, but there was very high variance in the data from inexperienced users. We would have needed to run many more participants to expose any significant differences that existed. We thus only discuss the data from experienced users.

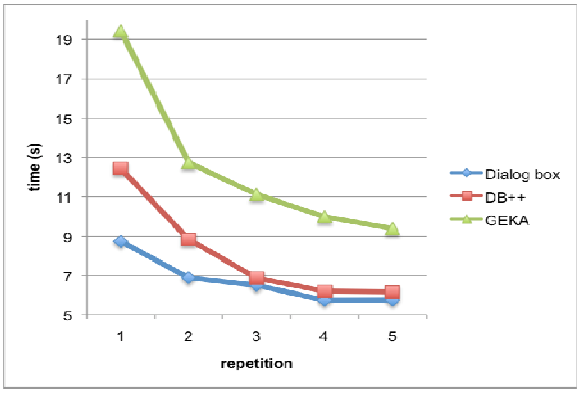

Figure 3: Mean completion times for each interface in each repetition (N=12)

RESULTS

Completion Time

An RM ANOVA indicated no main effect or interactions of presentation order; thus it was dropped as a factor. Figure 3 shows that participants got faster over time: a main effect of repetition ($F(1.51, 16.58) = 77.812$, $p < .000$, $\eta^2 = .876$) with significant differences ($p < .05$) between all pairs of repetitions except repetitions 4 and 5 ($p = .20$) indicates that performance was plateauing. The ANOVA we report eliminates repetition as a factor by using the mean times of just repetitions 4 and 5 to eliminate the learning affect.

Overall, DBOX++ and dialog boxes are not significantly different. There was a main effect of interface ($F(2, 22) = 20.476$, $p < .001$, $\eta^2 = .654$) with mean times for dialog boxes, DBOX++, and GEKA respectively being 5.76s, 6.19s, and 9.69s. Pairwise comparisons showed no significant difference between DBOX++ and dialog boxes ($p = 1.0$). All other pairs had significant differences ($p < .05$).

For simple commands, dialog boxes are fastest. The interfaces were impacted differently by the different command complexities (an interaction effect between interface and complexity, $F(2,22) = 5.969$, $p = .008$, $\eta^2 = .352$). For simple commands, a trend ($p = .072$) suggested that dialog boxes (4.09s) were faster than DBOX++ (5.73s), but no different for complex commands ($p = 1.0$). GEKA was slower than dialog boxes for both complexities ($p < .003$).

For short tasks, DBOX++ is fastest. An interaction between interface and task length ($F(2,22) = 30.88$, $p = .000$, $\eta^2 = .737$) indicated that for short tasks, DBOX++ (2.64s) was faster ($p = .007$) than dialog boxes (3.74s). There was no difference between the two for long tasks ($p = .355$). GEKA was slower than DBOX++ for both lengths ($p < .007$).

There was no interaction between interface and command familiarity ($F(2,2)=.867$, $p=.434$, $\eta^2=.073$)

Errors

Figure 4 shows that participants made fewer errors as the study progressed. While there was a borderline effect of interface ($F(2,22) = 3.01$, $p=.07$, $\eta^2 =.215$) across all repetitions, by repetition 5, errors had nearly disappeared for all three interfaces. In repetition 5 dialog boxes, DBOX++, and GEKA had 3, 3, and 4 errors respectively out of the 96 total trials for each interface.

Questionnaire and interview

Overall, participants rated DBOX++ the highest, with a mean of 17.00 on a scale of 0 ("I really dislike it") to 20 ("I really like it") vs. 10.50 for dialog boxes and 12.25 for GEKA. There was no significant difference between dialog boxes and GEKA ($p=.392$) all other pairs were significant ($p<.004$). When asked to explain why they preferred DBOX++, the most common reasons cited were being able to set parameter values without first having to specify the parameter name (7/12), liking the graphical feedback from the dialog box representation (6/12) (especially for verifying that the correct parameters were set (4/12)), and being able to only use the keyboard (5/12).

Participants said they were faster with DBOX++, with a mean of 18.17 on a scale of 0 ("very slow") to 20 ("very fast") vs. 11.75 for dialog boxes and 13.25 for GEKA. There was no significant difference between dialog boxes and GEKA ($p=1.0$) all other pairs were significant ($p<.03$). Explanations offered were that they sometimes were able to skip specifying the parameter name (4/12) or they believed that typing was faster than using a mouse (3/12).

Participants said they made fewer errors with dialog boxes, with a mean of 2.17 on a scale of 0 ("very few errors") to 20 ("many errors") vs. 6.58 for DBOX++ and 9.08 for GEKA. The only significant difference was between dialog boxes and GEKA ($p=.002$). While they did not necessarily like using the mouse, they felt that using one made it harder to make an error (5/12).

Ten of the 12 participants said that DBOX++ was easier to learn than GEKA and gave reasons similar to why they preferred DBOX++ overall. However, dialog boxes were felt to be the easiest to learn because participants were already familiar with them.

Eleven of the 12 participants said they would like to continue to use DBOX++, whereas only one would chose to use dialog boxes and only two said they would use GEKA.

DISCUSSION AND CONCLUSIONS

DBOX++ showed speed and error rates in the final repetitions nearly identical to dialog boxes. Yet participants reported a very strong preference for DBOX++. This suggests that DBOX++ should be included as an option for parameter specification: it makes experienced users more satisfied and does not impede their performance.

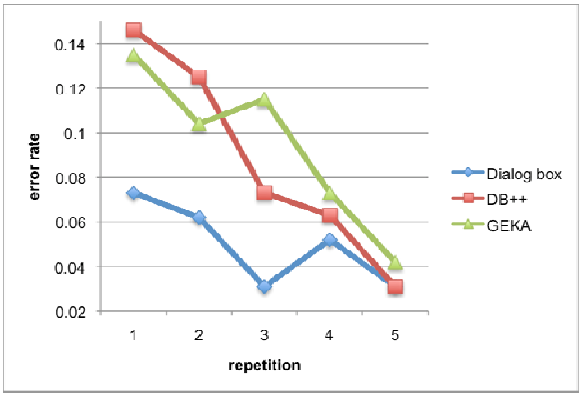

Figure 4: Error rates for each interface in each repetition (N=12)

We identified one case (simple commands) where DBOX++ is slower than dialog boxes and one case (short tasks) where it is faster. We expect that with practice users will quickly learn which tasks benefit from using DBOX++ and which do not, which will increase overall performance. Performance could potentially be further improved using a mixture of DBOX++ text input and standard WIMP interaction for a single task.

Despite the very similar speeds between dialog boxes and DBOX++, participants felt that they were much faster with DBOX++. Hendy et al. found a similar contradiction while evaluating GEKA, an interesting perceptual finding that bears further investigation.

DBOX++ significantly outperforms GEKA in speed under almost all conditions. This suggests that the dialog box-like graphical feedback of DBOX++ is a major improvement over the feedback in GEKA. Another major improvement over GEKA is that DBOX++ can support a wide variety of parameter types. Hendy et al. [3] lists several types of commands that GEKA is not able to accommodate because of its restrictive graphical feedback. DBOX++ overcomes most of these limitations.

GEKA has previously been shown to be an effective general interaction method for specifying many types of commands, including those with multiple parameters; it only fell short compared to dialog boxes. We have shown that DBOX++ outperforms GEKA and is highly competitive with dialog boxes. Application designers looking to improve their interfaces for experienced command line users could incorporate DBOX++ style graphical feedback into a keyboard-based command selection similar to GEKA.

REFERENCES

[1] Blacktree. (2009). Retrieved March 31. http://www.blacktree.com

[2] Gentner, D., and Nielson, J. (1996). The anti-Mac interface. In Communications of the ACM 39(8):70-82.

[3] Hendy, J., Booth, K., McGrenere, J. (2010). Graphically Enhanced Keyboard Accelerators for GUIS. Graphics Interface 2010.

[4] Humanized. (2009). Retrieved March 31. http://humanized.com

[5] Miller, R. C., Chou, V. H., Bernstein, M., Little, G., Van Kleek, M., Karger, D., and schraefel, m. (2008). Inky: a sloppy command line for the web with rich visual feedback. In *Proc. UIST 2008*, 131-140.

[6] Mozilla Labs. (2009). Ubiquity. Retrieved December 12. https://mozillalabs.com/ubiquity/

[7] Norman, D. (2007). The next UI breakthrough: command lines. In Interactions, 14(3):44-45.

[8] Raskin, J. (2000). The humane interface: New directions for designing interactive systems. Addison-Wesley.

[9] Shneiderman, B. and Plaisant, C. 2004 Designing the User Interface: Strategies for Effective Human-Computer Interaction (4th Edition).

[10] Stone, D., Jarrett, C., Woodroffe, M., Minoca, S. (2005). User interface design and evaluation.