Drag-and-Guess: Drag-and-Drop with Prediction

Takeshi Nishida[1] and Takeo Igarashi[1,2]

[1] Department of Computer Science, The University of Tokyo
tnishida@ui.is.s.u-tokyo.ac.jp
[2] JST Presto
takeo@acm.org

Abstract. Drag-and-guess is an extension of drag-and-drop that uses predictions which is based on application specific knowledge. As the user begins to drag an object, the system predicts the drop target and presents the result to the user. When the target is hidden in a closed folder or beneath other windows, the system makes it temporarily visible. This frees users from manual preparation such as expanding a folder tree or uncovering the target location. The user can accept the prediction by throwing the object, which then flies to the target. Or, if the prediction is unsatisfactory, the user can ignore it and perform the operation as usual. We built three prototype applications (email client, spreadsheet and overlapping windows) to show that DnG is useful in many applications. Results of the user study show that the proposed technique can improve task performance when the task is difficult to complete manually and reasonable prediction algorithm is available.

1 Introduction

Graphical user interfaces (GUI) have made much of the work using computers easy and intuitive. However, making an operation graphical does not necessarily improve performance. This is the situation with repetitive drag-and-drop operations (DnD). The user must make the source and target location visible by opening a folder or bringing the window to the foreground. Then the user must carefully move the mouse cursor back and forth for each object. This requires a substantial amount of labor especially when the user repeatedly performs DnD.

Some of these operations can be automated with user-defined macros or automatic systems. However, these approaches introduce several problems. First, defining macros or rules by hand is difficult and inflexible; it is nearly impossible for novice users. To make things worse, few macros or rules are reusable in other similar situations. Second, we cannot rely on completely automatic intelligent systems because they sometimes make serious mistakes (e.g. filing e-mail messages to incorrect folders). Because it is impossible to have a perfect prediction system, techniques that tolerate prediction errors are required.

In this paper, we propose an extension of traditional DnD, drag-and-guess (DnG), that uses predictions based on application specific knowledge. As the user starts dragging an object, the system predicts the drop target and responds

C. Baranauskas et al. (Eds.): INTERACT 2007, LNCS 4662, Part I, pp. 461–474, 2007.

by showing the result of the prediction to the user. When the target is not visible (e.g. hidden in nested hierarchical folders or outside of the visible area on the screen), the system automatically makes the target location visible. If the prediction is correct, the user can accept it by throwing the object (releasing the mouse button while dragging the object) and it will automatically fly to the target. If the prediction is not satisfactory, the user can ignore it and perform the operation as usual.

DnG can be helpful in various situations. For example, when the user is distributing e-mails in an inbox to nested folders, the system can predict appropriate target folder by its content and open it. When the user is editing a spreadsheet table, the system can predict the target cell based on the regularity observed in previous DnDs. When the user is opening a document with an application program using DnD, the system can find a possible drop target based on the document file type and the usage frequency of applications.

In the following sections, we review the related work, and present the DnG interaction design and its pros and cons. We then discuss the generality of the proposed technique in three prototype applications. We attempt to cover the issues that may arise when DnG is applied to various applications. We also describe the user study that we performed to examine the usability of DnG. Finally, we describe the details of the implementation.

2 Related Work

Many extensions to traditional DnD operations have been proposed. Pick-and-drop [1] and Hyperdragging [2] are designed for multiple display environments. Pick-and-drop allows the user to pick an object by tapping on a display and then to drop it on another display. Hyperdragging allows the user to DnD an object across physically separate but semantically connected displays. Drag-and-pop [3] is also designed for physically separate displays and very large displays. As the user starts dragging an object, proxies of possible drop targets appear around the object.

Many techniques have been proposed to improve object selection. Area cursor [4] and bubble cursor [5] increase the size of the cursor, while sticky icons [4] and semantic pointing [6] dynamically adjust the control display gain to make it easier to catch the target. Delphian Desktop [7] predicts the target using the linear relationship between the peak velocity and the target distance. The cursor jumps to the predicted location when the system detects the peak velocity. This greatly reduces the movement time in long-distance tasks, but adds extra time to short-distance tasks.

One shortcoming of these techniques is that the benefits decrease when the potential targets are densely located. In addition, considerable effort is required to make the target visible in advance. DnG overcomes these problems by using information about the application and the grabbed object.

Fold-and-drop [8] allows the user to leaf through overlapping windows during the DnD operation. Windows are folded like a paper in response to the

crossing-based gestures. Although this reduces manual preparation, considerable effort is still required to find the desired window from the stack of windows.

Predictions are used in various systems. Eager [9] predicts the next editing operation on a hypercard and Dynamic Macro [10] predicts the next text editing operation on Emacs. Jul [11] used destination prediction to constrain the movement to simplify navigation in a three-dimensional environment. Dulberg et al. [12] used predictions to reduce the precision needed for selecting or activating a small control from a group of candidates.

MailCat [13] predicts a target folder for an incoming e-mail message and displays the prediction result as buttons. The system achieves excellent classification performance by providing the top three suggestions instead of filing messages automatically. SmallBrowse [14] facilitates Web-browsing in small-screen computers by predicting the hyperlinks that the user would follow and inserting those predicted hyperlinks at the top and bottom of the Web page. These systems resemble our approach in that they tolerate prediction errors. They show predicted results in an unobtrusive way instead of performing the tasks automatically. We have generalized and extended the ideas presented in these systems in our interaction technique.

3 Interaction Design

DnD is an interaction technique for transferring or copying objects using a pointing device. The user grabs an object by pressing a mouse button, drags the object to the target location, and then drops the object into that location by releasing the button. DnG enhances this standard DnD by predicting the target and automating the preparation and execution processes (e.g. making the target location visible and dragging the cursor to that location).

Figure 1 shows the basic behavior of DnG. As in DnD, DnG begins when the user grabs an object, and the system presents the result of its prediction. The system shows a line connecting the starting location of the operation and the predicted target. If the target is not visible, the system makes it visible to the user. For example, when the target is hidden in a contracted path within a tree, the system automatically expands the path to the target.

The user then decides whether to accept the prediction result on the screen. If the user wants to accept it, he or she simply throws the object (releases the mouse button while the cursor is moving) and the object flies to the predicted destination. This process is presented through an animation. If the prediction is not satisfactory and the user wants to drop the object somewhere else, he or she can continue the DnD operation ignoring the DnG system. Releasing the mouse button while the cursor is stationary will drop the object to the cursor location. The automatically expanded target location returns to its original status when the operation is completed.

If the prediction is not satisfactory, the user can also ask the system to show other candidates. To show the next likely candidate, the user performs a clockwise rotation gesture; to show the previous candidate, he or she performs a

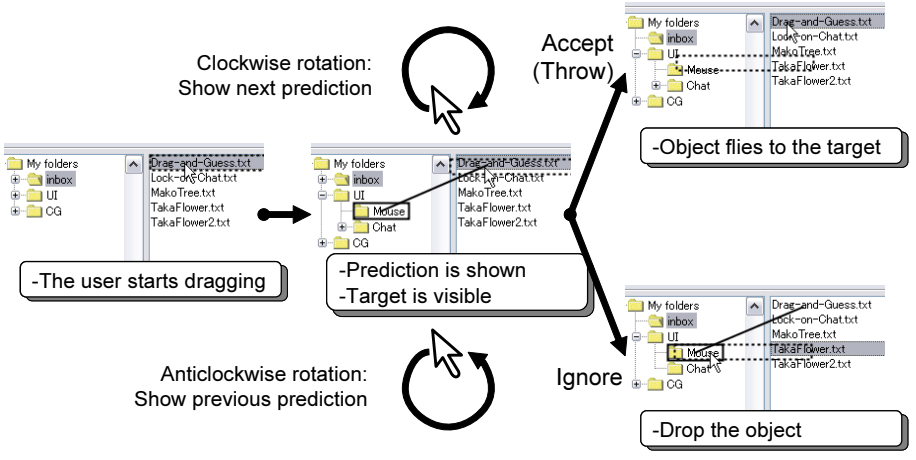

Fig. 1. Basic behavior of drag-and-guess

anticlockwise rotation. The system cancels the temporary expansion of the previous candidate and makes the next candidate visible. The user can leaf through the prediction results using these circular gestures until he or she finds a satisfactory candidate. The user is always free to ignore the list and perform the operation manually as we have described above.

The system does not, however, always show the prediction. Predictions are shown only when the task is difficult to perform manually, and the system is confident in its prediction. We discuss this topic in detail later in the implementation section.

4 Benefits and Drawbacks

4.1 Benefits

DnG improves DnD in many ways. First, it saves the user from tedious pointing tasks. When the prediction is correct, the user only has to press the mouse button on the object, confirm the target, and throw the object. This is much faster than carefully moving the cursor to the target location, especially when the target is far away or very small.

Second, DnG is helpful for finding the desired target in densely packed targets. Such targets are likely to be difficult to distinguish from each other, because they are often visually similar. For example, all targets appear almost identical when filing documents or editing spreadsheets. Our technique accentuates the differences between these targets by using semantic information and presenting it to the user.

Third, DnG saves the user from the tedious tasks of manually making the target location visible before the start of dragging, such as to expand many folders in a folder tree, rearranging windows to make the source and target visible, and scrolling in a window.

DnG has several advantages compared to other possible interaction techniques using predictions. First, unlike other techniques, it does not introduce additional widgets or consume any additional screen space. Interactions such as showing the prediction results as buttons [13] require additional space, but DnG shows only temporary visual feedback.

Second, the target is shown in its original context, which makes it easier for the user to understand where the predicted target is located. For example, showing a prediction result by using the destination folder name on a button [13] cannot distinguish multiple folders that have the same name but are at different locations in a tree. In addition, spatial memory obtained through long-term use of the application would improve task performance as seen in [15].

Third, DnG supports a smooth transition to DnD when the prediction fails. This is very important because prediction can frequently fail.

Finally, DnG is designed to be a common front-end interface for predictions. This allows the user to enjoy the benefits of predictions without having to learn specialized interfaces for each application. A user familiar with the DnG interface in one application could easily use it in other applications.

4.2 Drawbacks

DnG can cause additional overhead compared to plain DnD because the visual feedback showing the prediction inevitably attracts the user's attention. The user needs to see the prediction result and decide whether to accept it. If the prediction is correct, this extra effort is rewarded and the total execution time becomes shorter than manual execution. However, when the prediction is incorrect, it becomes an overhead added to standard DnD.

Our experience show that the overhead is relatively small and overall performance is improved given reasonable prediction accuracy. We also observed that users tend to accept these additional overheads as a necessary cost. Nonetheless, this overhead is an issue and one must carefully consider the trade-off when introducing prediction into a system.

5 Prototype Applications

The proposed interaction design is applicable to a wide range of domains. At the same time, however, it raises new issues and challenges for each application related to the visualization techniques to reveal hidden targets and the algorithms to make predictions. In this section, we introduce prototype implementations of three application systems that use DnG and discuss specific issues related to individual applications. We also briefly discuss lessons learned from informal testing using the prototypes.

5.1 Document Filing

Managing personal information is one of the most popular uses of DnD. For example, most e-mail reader applications allow users to organize their messages

into user-defined folders, and for many people, distributing new messages in an inbox to appropriate folders using repetitive DnD is a daily routine [16]. Struggling with other personal information, such as bookmarks and photographs, presents a similar situation.

DnG can make this tedious task much more efficient. It reduces the time for making decision on where to file by opening the path to the folder in a folder tree, and carefully moving the object from the source to the target.

We implemented a prototype system mimicking an e-mail reader (Fig. 1). The system fills the "inbox" folder with a set of text files and provides a DnG interface when moving a text file to a user-defined folder. Predictions are made by considering the semantic distance between the grabbed text and the texts that are already filed to desired folders. Such predictions are unreliable until a sufficient number of files has been placed. However, DnG is still acceptable in this case because the user can simply ignore the initial error-prone predictions until they become reliable. It is also possible to suppress predictions until system becomes confident.

Hidden folders can be revealed by expanding ancestor folders. In addition, the space-filling strategy and animated transitions used in Expand-Ahead [17] would be beneficial to the user.

DnG in this application can be seen as an attempt to address several problems associated with automatic classification. Automatic systems require a major amount of prior labor to set explicit rules for automatic distribution. It is also difficult to recover when fully automatic classifiers have failed; every day, for example, innocent messages are labeled as SPAM and it is a laborious work to rescue them. DnG address these issues by taking semiautomatic approach, where the user confirms the process interactively. This semiautomatic approach is especially effective for mid-size problems where only imperfect classification algorithms are available.

One can apply existing algorithms used in automatic classifiers for prediction. They examine the contents of the incoming message and find a folder that contains the most similar messages [13].

We observed that DnG benefits the user not only when the prediction exactly pinpoints the target, but also when the desired target is in a nearby location by revealing the target. This is likely to be a common situation if the messages are filed to semantic hierarchies.

5.2 Spreadsheets

DnD is repeatedly used to rearrange or reformat existing data. When editing a spreadsheet, for example, a user will make summary tables collecting values from various locations, sometimes in multiple worksheets. The user also rearranges the cells for partitioning purposes [18]. We often face similar spatial arrangement problems while working with presentation slides, interior design, and so on.

We implemented a prototype spreadsheet application using DnG (Fig. 2(a)). In the current implementation, the system predicts a target cell when moving a value by observing regularity in the operation sequence, for example, moving to

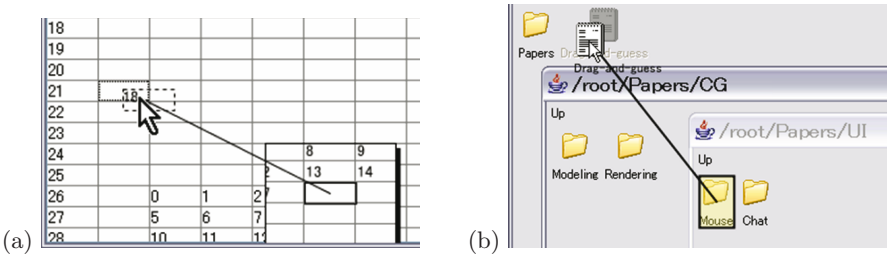

Fig. 2. Drag-and-guess applied to (a) spreadsheets and (b) overlapping windows

the cell next to the previous target cell, moving to cells with regular intervals, and so on. We employed similar rules as those used in [10].

The predicted targets are often out of view because spreadsheets tend to be very large. In such cases, the system shows a thumbnail of the target at the nearest edge of the window. Thumbnails include the context around the target so the user can easily recognize the situation. Thumbnails are also droppable; the user can manually drop onto the target cell or onto a cell around the target. This would be helpful when the prediction is not precisely correct but is near the intended target.

DnG here can be seen as a simplified version of programming by example or demonstration (PBE, PBD) techniques [9,19]. These systems observe the user's repetitive operations and try to construct a complete program by extracting rules. One problem with these traditional programming approaches is that it requires certain overhead to create and execute a program. The user typically needs to explicitly start a programming session and an execution session. A more serious problem is that these approaches fail when irregular instance appears during execution. DnG addresses these issues by presenting one prediction at a time and requires the user's confirmation instead of letting the system perform the task automatically.

DnG can coexist with PBE/PBD systems, serving as a general interface that interacts with the program being developed. It can support the user at an early stage, before he or she has performed sufficient number of examples for the PBE/PBD system to construct a complete program. Showing predictions to the user offers a guide to the inner state of the PBE/PBD system and acceptance of a prediction (throwing) works as a confirmation. After confirming that the system appropriately learned the desired rule with a sequence of successful DnG operations, the user can let the system do the remaining task automatically [9]. It is our future work to implement such PBE / PBD features and test its usability.

We observed that the users perform the tasks in comfort, mainly because it was quite clear for them when the predictions are likely to be correct. For example, when the user is constructing a two-dimensional table, the user can assume that the system makes a correct prediction while staying in the same row. In contrast, the user can anticipate that the prediction might be wrong when moving to the next row.

5.3 Overlapping Windows

We have implemented a prototype system mimicking a desktop in window systems (Fig. 2(b)). When the user starts dragging a file or a folder, the system automatically reveals the predicted target. There are many ways to reveal a target window hidden by other windows, for example, bringing the target window to the front, minimizing all overlapping windows, digging a hole to the target window [20], folding the overlapped window as in fold-and-drop [8]. In the current implementation, the system brings the target window to the front if it is hidden and restores the target window if it is minimized.

We have not yet implemented realistic prediction algorithm in this application partly because there are many possibilities in this case and it is difficult to design a general method. This is even more difficult because each target window can run different application. We discuss this inter-application DnD in the implementation section. The simplest method of using most recently used drop target as prediction result might work best for most cases (it would be better to show predictions after the user drop objects in the same window for a couple of times).

We observed that revealing the target attracts much larger attention of the user compared to other applications. This is because the system redraws a larger area to reveal the target. We suggest using a conservative way for revealing the target. Moreover, if the worst happens, the correct target might be hidden by the system. Interactions to cancel the effect would be beneficial in these cases.

6 User Study

We conducted a user study to examine the usability of DnG. In this study, we adopted two tasks, message filing and data rearrangement, which are typically done by DnD. All of the participants carried out both tasks with DnG and without DnG (= DnD).

Our hypothesis was that participants would perform faster with DnG if the system shows a prediction only when the task is difficult to do manually and the system is confident in its prediction.

Note that we evaluated the interaction techniques and not the prediction algorithms. It is our future work to evaluate the performance of our approach using real predictions. In addition, in this study, we did not address the issues that arise from showing multiple predictions to make the control of test conditions simple; subjects did not leaf through multiple candidates.

6.1 Design

Task 1. The first task was to file objects into their appropriate folders (Fig. 3). The folders were hierarchically organized three deep. Each folder contained two folders named "a" and "b," and the eight leaf folders were used as the target folders. The files used in the study were named randomly by the system, using

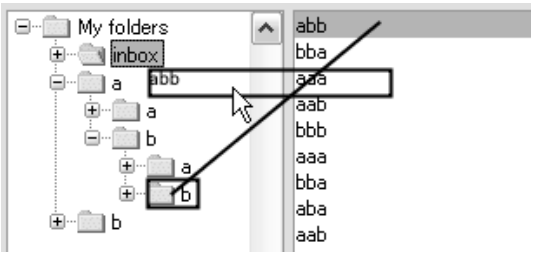

Fig. 3. Snapshot of the application used in Task 1

two letters "a" and "b." These file names served as instruction on where to file the object; a file name "aba" had to be filed to folder "a-b-a."

We used random numbers to simulate a situation in which some imperfect prediction algorithms are available and that the system only shows prediction when it is rather confident of that prediction (the system does not show prediction when it is likely to be wrong). In this experiment, the system shows predictions at the rate of 50%. If the prediction was shown, its accuracy was 80%. These numbers are chosen as a plausible setting considering available filtering algorithms [13]. All operations were treated as difficult to perform manually.

In this study, subjects were not allowed to expand the folder before performing DnD or DnG. Instead, they had to hold still on the folder for 500 ms while dragging to expand it. In addition, the folders that expanded during the operation were collapsed after each operation.

Subjects were instructed to perform the task as quickly and as accurately as possible, but were not required to redo the operation when objects were filed to incorrect folders. The subjects performed 20 filings under each condition. The order of the condition was balanced, with half of the subjects performing the tasks with DnG first, and half with DnD first.

The system recorded the time interval from when the participant started dragging an object to when the object was filed in the appropriate folder. The system also recorded the drag trajectories and the errors that occurred during the task.

Task 2. The second task was to rearrange a one-dimensional table with 32 cells into a transposed two-dimensional (8 4) table (Fig. 4 (left)). The window size was fixed so that the whole table did not fit in the view (Fig. 4 (right)). The participants had to scroll the view, either by using the scrollbar, or by dragging the object near the edge of the window. Prediction was done by observing the regularity in the operation sequence as described in the prototype applications section.

Under both conditions, subjects could undo the operations using a keyboard shortcut (Ctrl-Z). This was especially important under the "with DnG" condition because the algorithm used in the study learned from mistakes.

The subjects performed the task four times under each condition (with and without DnG), alternating between the two conditions. The order of the interface

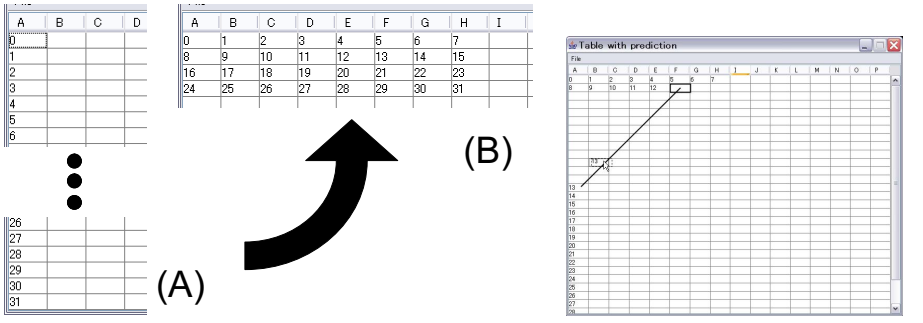

Fig. 4. Task 2. (left) Rearranging a one-dimensional table (A) to a two-dimensional table (B). (right) Snapshot of the application.

was balanced: half of the participants first performed the tasks with DnG, and vice versa.

The system recorded the interval from the time when the participant pressed the start button located outside the window, to the time when the desired table was completed. The system also recorded the drag trajectories during the task.

6.2 Procedure

Ten male subjects were recruited for this experiment. They were all right-handed and frequent mouse users ranging in age between 23 and 28 years old. The study took place in our laboratory, one participant at a time. Before the experiments, the participants were briefed on the purpose of the study and the method of operating the applications. The participants were allowed to practice the technique until they felt confident in operating the applications. The tasks followed the training. Finally, the subjects filled out a questionnaire. The study was run using a standard PC with MS-Windows XP and a 17h monitor (resolution 1280×1024) equipped with an optical mouse.

6.3 Results

Task 1. Figure 5 shows the average operation time and the average drag distance required to file one object. The error bar indicates standard deviation. According to paired t-test, subjects could file the objects significantly faster ($p < 0.05$) with significantly shorter drag distance ($p < 0.05$) using DnG. Error rate during the study was considerably low for the both conditions (2% with DnG vs. 3% with DnD).

Task 2. Figure 6 shows the average time and the average drag distance required to complete the rearrangement task. The error bar indicates standard deviation. According to paired t-test, subjects could finish the task significantly faster ($p < 0.05$) with significantly shorter drag distance ($p < 0.05$) using DnG.

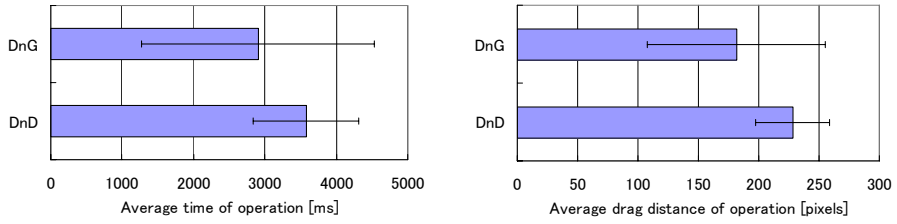

Fig. 5. Average operation time (left) and drag distance (right) of the first task

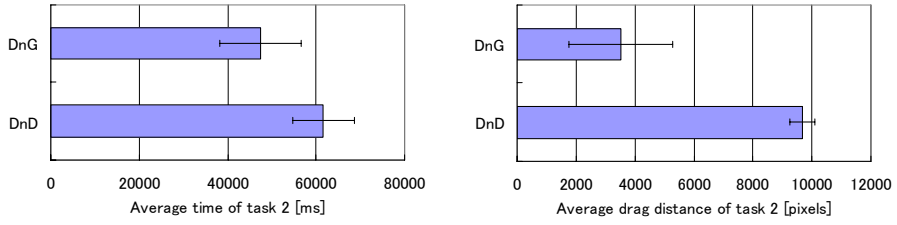

Fig. 6. Average operation time (left) and drag distance (right) of the second task

Questionnaire and Feedback from Participants. At the end of the study, participants answered a short questionnaire. Eight participants answered that they file messages manually (like task 1) with their regular e-mail readers, and six of them wanted to use DnG if the prediction accuracy is comparable to the study. Five participants answered that they perform rearrangements in spreadsheets (like task 2) with their regular applications, and all of them wanted to use DnG in those applications. Four participants felt difficulty in throwing the objects and suggested improvements. Three suggested combinations with mouse or keyboard buttons, and one suggested visual feedbacks showing what is going to happen before releasing the mouse button.

7 Implementation

7.1 Architecture

Our current implementation of DnG is written in Java. The common part is implemented as an application independent component to minimize the programming costs for the application developers. When adding the DnG feature to an existing application, they can focus on the application specific prediction algorithms and visualizations.

Filtering the Prediction List. Predictions are filtered from the prediction list if the user can manually drop the object on the target with ease. The system judges that the task is difficult when the expected operation time is longer than a predefined threshold. The expected operation time can be given as a summation of the time required for two-dimensional pointing and that required for

application-specific subtasks. We used the index of difficulty of a two-dimensional pointing task [21] to estimate the effort required for the pointing task. We also have to consider the difficulty of the subtasks associated with a specific application, such as opening a folder, rearranging windows, scrolling the view, and so on. For example, a model of user performance in traveling down a tree to the desired path is given by McGuffin et al [17]. We do not model these subtasks here because they are different for each application. Instead, we chose to provide a way for software developers to encode the additional difficulty and pass it on to the system.

Detecting Gestures. A throwing gesture is used to distinguish between a normal drop and prediction acceptance when the mouse button is released. A mouse release would be treated as "throw" if the drag velocity before the release is faster than a threshold value. Rotation gestures are used to switch the prediction shown by the system. We used the exterior angle of the cursor trajectory so that the motion is not constrained to a specific circle. This is similar to [22]. The system detects a clockwise rotation when the summation of angle history reaches 2π, and vice versa. Summation will be reset to 0 when a gesture is detected.

7.2 Implementation Issues

Implementation Cost. The prediction algorithms need not be perfect; developers may choose the second-best algorithms considering the implementation cost and the computation time. This is much easier than adding completely automatic features to the application.

Computation Time. It is not necessary that prediction algorithms run at the moment when the DnG controller makes a request. If the prediction takes time, developers should choose to run the prediction in a background thread, and then return the precomputed result when requested. The current implementation does not include such multi-threading features. It has to be implemented by the software developers.

Inter-Application Predictions. Prediction algorithms for inter-application DnDs would be combinations of the approaches discussed in the application section: content-based classification, learning from operation sequence, and other heuristics. For example, we often classify documents using hierarchical folders and arrange document layout on the desktop.

In addition, many heuristics can be incorporated into the system. For example, when the user is dragging a text fragment, the most likely targets are the search boxes and the text-component which just lost the input focus. Similarly, the system should put a higher priority on folders that have just been opened, or folders in a plugged-in USB thumb drive. A simple rule, like "the user tends to repeat DnD to the same folder" might work also.

Inter-application predictions can be treated with inner-application predictions, by putting the prediction list together and presenting it to the user as

a whole. Alternatively, predictions can be treated separately by extending the system to present inner-application candidates when the cursor stays inside the window and inter-application candidates after the cursor exits the window. It is our future work to implement and test such inter-application DnG methods.

8 Conclusion

We have proposed an extension of traditional drag-and-drop using predictions, called drag-and-guess (DnG). DnG enables the user to interact with various application specific prediction algorithms through a common gestural interface. We introduced three prototype applications to show the generality of the proposed method and discussed various issues specific to individual applications. We also described the user study that we performed to examine the usability of DnG. Finally, we described the details of the implementation. We believe that DnG can be a common front-end for predictions and encourage the use of predictions in many applications.

To make DnG more practical, we need to address several issues. First, we have to observe the daily usage of DnD and find more application fields that are suitable for DnG. Next, we need to develop algorithms and interactions specific to each application. Evaluating DnG under such algorithms and interactions is also another important issue; algorithms have to be precisely controlled if they are to be used in formal evaluations.

References

1. Rekimoto, J.: Pick-and-drop: a direct manipulation technique for multiple computer environments. In: Proc. UIST 97, pp. 31–39. ACM Press, New York (1997)
2. Rekimoto, J., Saitoh, M.: Augmented surfaces: a spatially continuous work space for hybrid computing environments. In: Proc. CHI 99, pp. 378–385. ACM Press, New York (1999)
3. Baudisch, P., Cutrell, E., Robbins, D., Czerwinski, M., Tandler, P., Bederson, B., Zierlinger, A.: Drag-and-pop and drag-and-pick: techniques for accessing remote screen content on touch- and pen-operated systems. In: Proc. INTERACT 2003, pp. 57–64 (2003)
4. Worden, A., Walker, N., Bharat, K., Hudson, S.: Making computers easier for older adults to use: area cursors and sticky icons. In: Proc. CHI 97, pp. 266–271. ACM Press, New York (1997)
5. Grossman, T., Balakrishnan, R.: The bubble cursor: enhancing target acquisition by dynamic resizing of the cursor's activation area. In: Proc. CHI 05, pp. 281–290. ACM Press, New York (2005)
6. Blanch, R., Guiard, Y., Beaudouin-Lafon, M.: Semantic pointing: improving target acquisition with control-display ratio adaptation. In: Proc. CHI 04, pp. 519–526. ACM Press, New York (2004)
7. Asano, T., Sharlin, E., Kitamura, Y., Takashima, K., Kishino, F.: Predictive interaction using the delphian desktop. In: Proc. UIST 05, pp. 133–141. ACM Press, New York (2005)

8. Dragicevic, P.: Combining crossing-based and paper-based interaction paradigms for dragging and dropping between overlapping windows. In: Proc. UIST 04, pp. 193–196. ACM Press, New York (2004)
9. Cypher, A.: Eager: programming repetitive tasks by example. In: Proc. CHI 91, pp. 33–39. ACM Press, New York (1991)
10. Masui, T., Nakayama, K.: Repeat and predict: two keys to efficient text editing. In: Proc. CHI 94, pp. 118–130. ACM Press, New York (1994)
11. Jul, S.: Predictive targeted movement in electronic spaces. In: CHI 02 Extended Abstracts, pp. 626–627. ACM Press, New York (2002)
12. Dulberg, M.S., Amant, R.S., Zettlemoyer, L.S.: An imprecise mouse gesture for the fast activation of controls. In: Proc. Interact 1999, pp. 375–382 (1999)
13. Segal, R., Kephart, J.: Mailcat: an intelligent assistant for organizing e-mail. In: Proc. AGENTS 99, pp. 276–282 (1999)
14. Sugiura, A.: A web browsing interface for small-screen computers. In: CHI 99 Extended Abstracts, pp. 216–217. ACM Press, New York (1999)
15. Robertson, G., Czerwinski, M., Larson, K., Robbins, D., Thiel, D., van Dantzich, M: Data mountain: using spatial memory for document management. In: Proc. UIST 98, pp. 153–162. ACM Press, New York (1998)
16. Whittaker, S., Sidner, C.: Email overload: exploring personal information management of email. In: Proc. CHI 96, pp. 276–283. ACM Press, New York (1996)
17. McGuffin, M., Davison, G., Balakrishnan, R.: Expand-ahead: a space-filling strategy for browsing trees. In: Proc. INFOVIS 04, pp. 119–126 (2004)
18. Hendry, D.: Display-based problems in spreadsheets: a critical incident and a design remedy. In: Proc. Visual Languages 95, pp. 284–290 (1995)
19. Maulsby, D., Witten, I., Kittlitz, K.: Metamouse: specifying graphical procedures by example. In: Proc. SIGGRAPH 89, pp. 127–136. ACM Press, New York (1989)
20. Ishak, E.W., Feiner, S.K.: Interacting with hidden content using content-aware free-space transparency. In: Proc. UIST 04, pp. 189–192. ACM Press, New York (2004)
21. Accot, J., Zhai, S.: Refining fitts' law models for bivariate pointing. In: Proc. CHI 03, pp. 193–200. ACM Press, New York (2003)
22. Moscovich, T., Hughes, J.F.: Navigating documents with the virtual scroll ring. In: Proc. UIST 04, pp. 57–60. ACM Press, New York (2004)