An Application-Independent System for Visualizing User Operation History

Toshio Nakamura
Department of Computer Science
The University of Tokyo
toshi@ui.is.s.u-tokyo.ac.jp

Takeo Igarashi
Department of Computer Science
The University of Tokyo / JST ERATO
takeo@acm.org

ABSTRACT
A history-of-user-operations function helps make applications easier to use. For example, users may have access to an operation history list in an application to undo or redo a past operation. To provide an overview of a long operation history and help users find target interactions or application states quickly, visual representations of operation history have been proposed. However, most previous systems are tightly integrated with target applications and difficult to apply to new applications. We propose an application-independent method that can visualize the operation history of arbitrary GUI applications by monitoring the input and output GUI events from outside of the target application. We implemented a prototype system that visualizes operation sequences of generic Java Awt/Swing applications using an annotated comic strip metaphor. We tested the system with various applications and present results from a user study.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces - Graphical user interfaces.

General terms: Design, Human Factors.

Keywords: storyboards, diagrams, program visualization, summarization, operation history.

INTRODUCTION
Many attempts have been made to make applications easier to use by incorporating a history-of-user-operations function. Examples include Undo/Redo, which can undo the last user operation or redo the last undone operation; the History feature, which can rerun a past operation; and Programming by Example, which generalizes user operations into a program. To make effective use of these features, it is important to be able to search a long user operation history and find target states quickly.

However, most conventional interactive systems provide the user operation history only as a list of text commands. The text format is very easy to manage, but it is difficult for users to comprehend detailed user interactions and application states quickly and precisely. Another approach is to visualize operations using animation to explain the user interaction and the corresponding change in the application's visual state. However, viewing an animation takes time. Of course, one could fast-forward through an animation, but as playback speed increases, it becomes more difficult to recognize important details.

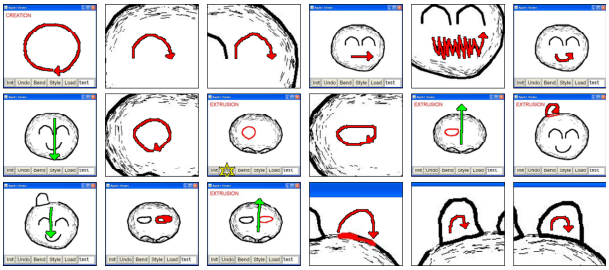

Figure 1: An annotated history generated from an operation history.

To address these problems, visual history representations have been proposed. For example, the Chimera system [17] visualizes an operation history as a sequence of small snapshots and Su's system [27] shows operation history as an annotated diagram. However, these systems are tightly integrated with their target applications and difficult to apply to new applications. Therefore, we propose an application-independent method that can visualize the operation history of arbitrary GUI applications without modifying the target application. We do this by monitoring the input GUI events and recording screen snapshots from outside of the target application.

This paper describes a prototype system that visualizes the operation history of Java Awt/Swing applications. For visualization, we combine a sequence of snapshots [17] with annotations on them showing detailed user operations [27]. We tested the system with several applications and observed that it could successfully visualize the operation history of these various applications. We also performed an informal user study to see whether the proposed system helps users find specific operations in a history.

The remainder of this paper is organized as follows. First, we provide a brief overview of previous work in related areas. Then we present the visual design of an annotated

UIST'08, October 19-22, 2008, Monterey, California, USA.

history and explain how user interactions are visualized. Then we describe its implementation in detail, introduce a prototype application using the technique, and present the results of a user study we performed to verify the effectiveness of the method. We conclude with a summary of the technique, its limitations, and future work.

RELATED WORK

Our work builds on the results of many research attempts in the past. We briefly summarize the most related systems that influenced the design of our current system.

Operation History

The history of operations performed by users has long been used for a variety of purposes, including undoing actions in a text or graphical editor and going back to previous pages in a web browser. Visualizing this history as a stack of text lines is the most popular method. For example, GINA [7] and Amulet [24] provide descriptive text lists for selective undo operations. A few applications, such as the electronic whiteboard [25] and desktop environment [26], retrieve past information using a time slider. The DocWizard [6] and Koala [19] systems learn how to perform a task from an operation history and present the result as instructions in a natural language. DocWizard also provides an annotated screenshot of the current step.

Other systems attempt to make the operation history easier to use and comprehend by using a graphical user interface. Meng et al. [22] provided interactive snapshots that can select objects and filter the operation history to show only those commands that affect the objects. However, each operation is shown as a text caption, not a graphical annotation as in our system. Chimera [17] uses a comic strip metaphor to depict the operation history; each operation is represented as a snapshot that focuses on the objects being manipulated. It also clusters related actions into a group for better comprehension. The Pursuit system [23] extends the comic strip metaphor and builds a more structured visual representation that represents a program with variables, loops, and conditionals. Mondrian [18] represents the behavior of user operations with a pair of screen snapshots just before and after the operations were executed. Su's system [27] visualizes the operation history of a drawing editor via a static image with a number of annotations on it, each of which represents an individual editing operation. ExperiScope [11] visualizes the log of low-level interactions to help in analyzing empirical evaluation data.

Visual Effect

In the fields of visualization and graphics, many types of visual effects have been proposed to illustrate dynamic phenomena using static depictions. Masuch et al. [21] applied speed lines to convey object motion. Kawagishi et al. [15] introduced several cartoon-blur techniques for 2D animations. Kim et al. [16] developed a semiautomatic system to add non-photorealistic and expressive illustrations of motion to video.

Some approaches improve user comprehension of user interfaces using visual effects. Mac OS X complements animation with motion blur when iconifying windows. Baudisch et al. [4] proposed a high-density cursor to help users keep track of fast-moving mouse cursors by adding a strobe effect to the mouse trajectory. Kaptelinin et al. [14] showed that transient visual cues, such as temporarily dimming old text immediately after scrolling, can improve the reading performance of scrolled pages. Bezerianos et al. [8] introduced the idea of mnemonic rendering, using persistence or flashback to present visual changes that may otherwise be missed by users because of changes that occurred in the background. Baudisch et al. [5] presented Phosphor, a technique for explaining transitions in the user interface using afterglow effects.

Visual Summarization

Our work was motivated by several recent approaches using visual abstraction and summarization. LineDrive [2] is a real-time system for automatically generating customized route maps emphasizing the most essential information for following a route. Agrawala et al. [1] also presented design principles for automatically creating effective assembly instructions that are easy to understand and follow. Assa et al. [3] attempted to carefully select the key poses or frames for composing a stroboscopic image (Action Synopsis) that illustrates motion capture data.

There are also techniques for summarizing video in a single image such as [28, 13, 29, 20, 9]. Goldman et al. [10] introduced a method for visualizing short video clips in a single static image, using the visual language of storyboards. Their schematic storyboards annotate subject and camera motion with 3D arrows, outlines, and text.

VISUAL DESIGN

Building upon previous visualization methods for GUI operation histories, we visualize the entire operation history with a comic strip metaphor [17] and augment each history entry with annotations [27] such as word balloons and arrows. This section describes the visualization details of the annotated history approach.

Figure 2: Visualizing a series of user interactions, namely two files dragged and dropped into a folder.

Figure 2 shows an example of an annotated history. User operations are visualized as annotations on each screen snapshot. The white arrows indicate mouse movements and the red arrows indicate mouse drags.

However, as can be seen in Figure 2, this method of visualizing all interactions in isolation appears complicated and redundant. For example, mouse movements need not be visualized because they are usually not associated with any operation in general applications. Another potential complication using this method is when users wish to integrate two consecutive clicking operations (double-click) into a single image because there are numerous way to depict such an operation, depending on the target application or the purpose of visualization. We refined this type of anno-

tated history to make it more effective and versatile by allowing customizable filtering and cluster processing as the needs arise (see implementation section).

Mouse Operation

In general, the behavior of mouse operations depends on the location of the mouse pointer. In the standard desktop environment, the clicking operation selects the object directly under the pointer. Therefore, mouse operations in the annotated history are annotated on the mouse trajectory. Figure 3 describes the annotation list for mouse operations. Operations with motion are denoted by an arrow; those without motion are denoted by an icon.

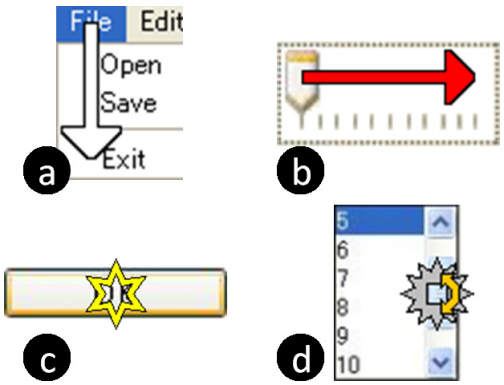

Figure 3: Annotation list for mouse operations: (a) moving the mouse, (b) mouse drag, (c) mouse click, and (d) mouse wheel.

Mouse operations also vary according to the type of mouse buttons in use, such as object selection via left click and menu open via right click. To distinguish these operations, the annotations for the same interactions are color-coded by button as shown in Figure 4. (Color-coding may not be the best choice. We would like to test more intuitive representations in the future.)

Figure 4: Color-coded annotations: (a) left click, (b) right click, and (c) click of other buttons.

Keyboard Operation

Unlike mouse operations, the objects targeted by key input operations cannot be identified by tracking the mouse pointer. To show the target object of keyboard operations, the annotated history first highlights the region of the keyboard in a green frame. Then a word balloon labeled with the key input sequence appears near the highlighted region (Figure 5). Some keys are represented by specific characters (e.g., "<Enter>" for the enter key and "<BS>" for the backspace key).

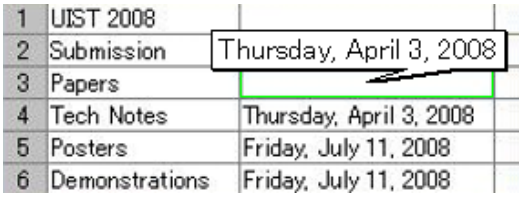

Figure 5: Annotation for key input operations.

Exception

Applications with failures behave in unexpected ways, and unless the result of failure is clearly shown on the screen, it is very difficult to identify the point of failure from the operation sequences. Therefore, it is important to recognize where failures occur by visualizing them in the operation history. In the annotated history, execution failures are denoted by a sticky note labeled with the type of failure (Figure 6).

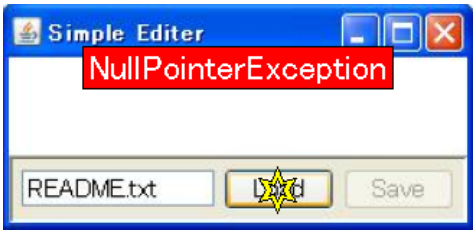

Figure 6: Annotation for execution failure.

The annotations for execution failures are shown together with the annotations for interactive operations. This makes it easier during debugging to investigate dependencies between execution failures and the responsible interactions.

Enhanced Snapshots

One of the simplest ways to illustrate application states at the time of operation execution is to use a single snapshot image. Although this can represent the application state at a specific point in time, it is not enough to represent the visual transitions. To address this weakness, we provide the following two visualizations as options: strobe style (Figure 7a) and inset style (Figure 7b).

To generate a strobe style image, the system first synthesizes a single background image by examining all images during the execution of the target operation such as dragging. Specifically, the system examine the color of a specific pixel across all target frames and choosing the dominant color as the background color. The system then identifies the moving object by comparing each frame and the base background image. Finally, the system adds the moving object on the background with increasing opacity (Figure 7a). This technique is useful for translational movement, but it is not good for in-place movement such as rotation and scaling.

To generate an inset style image, the system first computes the difference of pixel color between the scene's first and last snapshots. If the difference between two pixels in the same location is higher than a given threshold, the location is identified as the active region. The system creates a smaller sized version of the last snapshot, adds bounding boxes indicating the location of active regions, and pastes it

onto the annotated snapshot avoiding the mouse trajectory (Figure 7b).

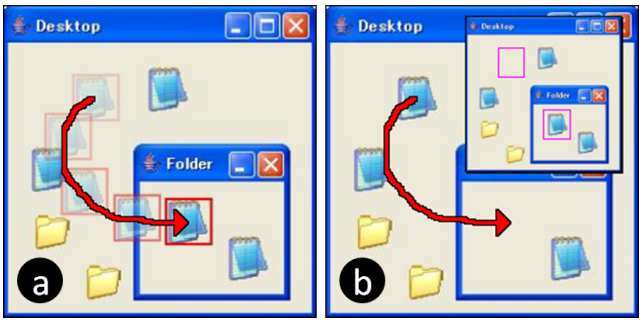

Figure 7: Two extended styles of snapshots used in the annotated history: (a) strobe style and (b) inset style.

IMPLEMENTATION

Our system is broken into the following four stages, performed in sequence: data recording, operation analysis, filtering/clustering, and scene composition. This section describes the implementation details of each of these stages.

Our current system is implemented as a Java application and is designed to visualize user operation history for Java applications using Awt/Swing. However, our basic approach can be applied to general interactive applications with GUIs.

Recording

There are several ways to monitor user interactions. One is to modify the platform where the target application is running, as in DocWizard [6] modifying the Eclipse platform and Koala [19] modifying FireFox. However, modifying the virtual machine seems to be overkill for our purpose. Another is to modify the target application directly as in many PBD systems [17, 27], but this is labor-intensive and lacks generality. In general, GUI events are managed in an Event Queue object and sent to the target application. Therefore, we insert a proxy between them to add a record function without modifying the target application (Figure 8).

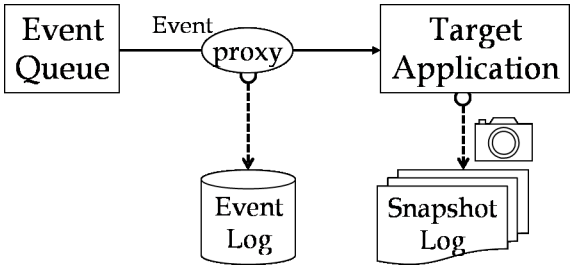

Figure 8: Overview of the record module.

When recording, the system inserts the proxy and automatically records the interaction sequences as operations are executed. In addition to mouse and keyboard operations, however, there are various GUI events sent from the Event Queue to the target applications, such as paint screen, focus change, and notification events for windows or components. The system records all of these GUI events in order to analyze what is happening in detail.

The system also automatically captures a snapshot of windows in parallel with recording events as needed. Each snapshot includes not only the image data on the screen but also some additional information such as a timestamp corresponding to the recorded events and hierarchical structures of GUI components.

Operation Analysis

This process takes the recorded raw event sequence as input and derives semantically coherent operations. The raw event sequence recorded by the recording module includes low-level events representing user interactions (e.g., mouse and key events) and a variety of other events such as change focus and paint components. From these, it derives high-level semantic event sequences by applying three conversion processes, as shown in Figure 9.

Paint events and windows events are notification events generated for the internal processing of applications (i.e., not in response to user interactions). Therefore, we consider these events a result of the last user interaction (trigger event) and cluster them with the trigger event into a single "basic operation event" (Figure 9a).

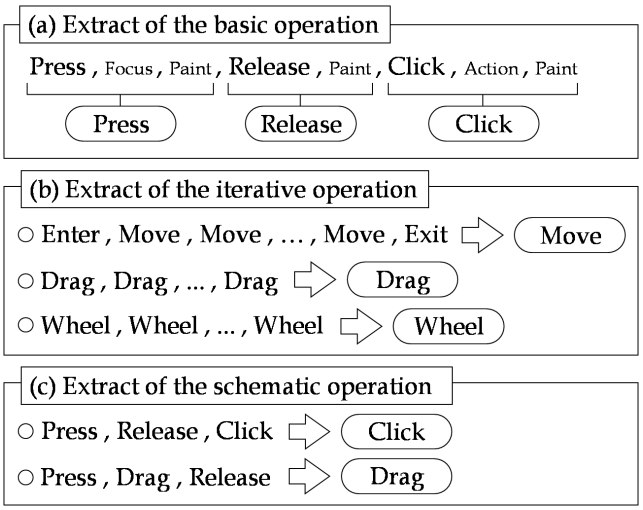

Figure 9: Extraction process of schematic events.

The next process is to identify iterative events. Many components of event sequences are successive basic operation events of the same type. Therefore, we convert such sequences into an "iterative operation event" (Figure 9b).

Finally, we derive semantic operations from event sequences that are converted by the two processes described above (Figure 9c). For example, a semantic operation of a mouse click is the result of sequential operations where the user first presses the button and then releases it. Another example is the semantic operation of a mouse drag expressed as follows: the user presses the button, moves the mouse pointer without releasing the button, and then releases the button. We convert a sequence of such meaningful operations into a "semantic operation event".

Our system assumes no interactions overlap. That is, the system cannot handle a right mouse click with the left mouse button pressed. The appropriate treatment of multiple overlapping user interactions requires further work.

Filtering and Clustering

We further refine semantic event sequences by filtering and clustering. Filtering removes unnecessary operations for visualization and clustering groups operations into a single static image.

In general, these processes depend on the application or the intentions of the user, so it is impossible to prepare predefined rules. For example, in nearly all applications, it is unnecessary to visualize mouse movements; exceptions are applications that invoke processes when the mouse pointer crosses the boundary of GUI components. Another example is the double-click operation. For files or folders in a standard desktop environment, the double-click operation is different from plain consecutive clicks so double-clicks should be clustered. However, if double-click is not associated with specific operation, individual click should be treated separately.

Therefore, in our system, the filtering and clustering modules can be set per application. In the current implementation, the settings of these modules are specified manually. The user makes the modules and inserts them into the code. It is not too difficult to write clustering rules: simply specify what kinds of interactions should be clustered together. For example, in Figure 12, the given rules are to cluster a left-button click after a right-button click (pop-up menu), consecutive clicks (double click), and multiple drags in the same window. It took 10-30 minutes for us to write rules for each application. In the future, we would like to incorporate support for specifying or changing these modules dynamically via Programming by Demonstration.

Clustering does fail when the rule is not appropriate, e.g., when semantically un-related operations are clustered, the snapshot fails to represent the application state appropriately. However, such failures only slightly degrade the quality and are not fatal.

Scene Composition

This process takes a semantic operation and snapshot sequence as input and returns annotated scene images. Initially, the system generates a background image of the scene, by combining multiple snapshot sequences captured during interactions. Our current implementation provides the following three visualization styles:

- Normal Style: The simplest style, it represents each background image with a single thumbnail, typically using the scene's first snapshot.
- Strobe Style: This style represents the visual transition during an interaction. We use a simple background subtraction method with a posterior probability to extract moving objects and paste them onto the background with increasing opacity.
- Inset Style: This style represents the final result of a visual transition. We use a temporal differencing method and a nearest-neighbor method to detect changing regions and highlight them in the scene's last snapshot. The result is provided as a sub-image inserted in the scene's first snapshot.

After generating the background image for each scene, the system adds annotations corresponding to user operations. Newer annotations are stacked on top of older ones. If annotations become covered by the sub-image, the system repositions the latter to avoid overlap.

EXAMPLES

The previous sections provided a few preliminary results of our annotated history approach. This section presents several more examples applied to actual applications.

Figure 10 shows images selected from an annotated history for a sketch-based modeling system used in the user study. The left image illustrates a cutting operation, and the middle image illustrates an extrusion operation. (Both are executed by drawing a free-form stroke via a left drag.) The right image illustrates a rotation operation (via right drag).

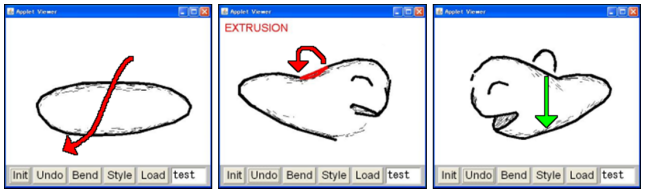

Figure 10: Three operation scenes from an annotated history for a sketch-based modeling system.

Figure 11 shows an annotated history (in inset and strobe styles) for a control panel composed of standard GUI widgets. The left image represents changing a combo box by clicking, and the right image represents adjusting a slider by dragging. This way the user can quickly identify which widget was modified by the operation, as previously demonstrated in Phosphor [5].

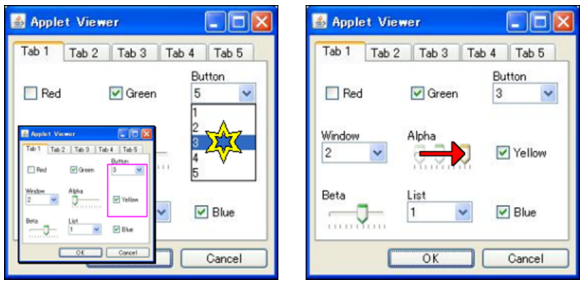

Figure 11: Two operation scenes from an annotated history for standard GUI widgets.

Additional examples are shown in Figure 12. They are successive operation scenes for a desktop environment, and describe how the user creates a new folder to group files, then moves green files into it, and finally changes its name to "Green." In this example, we used a customized cluster module to handle double-click operations as a single operation and integrate multiple related operations into a single image. The annotation for double clicking is denoted by a click mark and click count. In addition, these annotations

are numbered to provide the operation order in each scene that includes multiple operations.

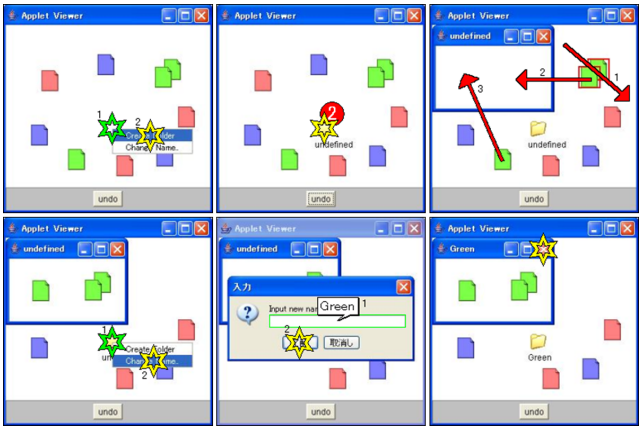

Figure 12: Part of an annotated history for a desktop environment. It includes eleven sequential user interactions. A background image for each scene uses a carefully selected normal style image.

We also applied our system to publicly available programs that were not developed by the authors. Figure 5 shows the result of applying our method to a publicly available spreadsheet program. It is a little bit difficult to directly apply our current system to arbitrary real world applications, because the operation history of such applications may be dependent on various non-GUI factors, such as multi-threading, networking, database and file access. We believe that our method can work for text editing by using an appropriate representation (e.g. the balloon in Figure 5). We have not yet worked on algorithms to segment and cluster long typing sequences; this remains as future work.

PROTOTYPE SYSTEM

To demonstrate and evaluate the usability of this annotation approach, we developed a prototype browser system using the annotated history. This section describes the features of the system.

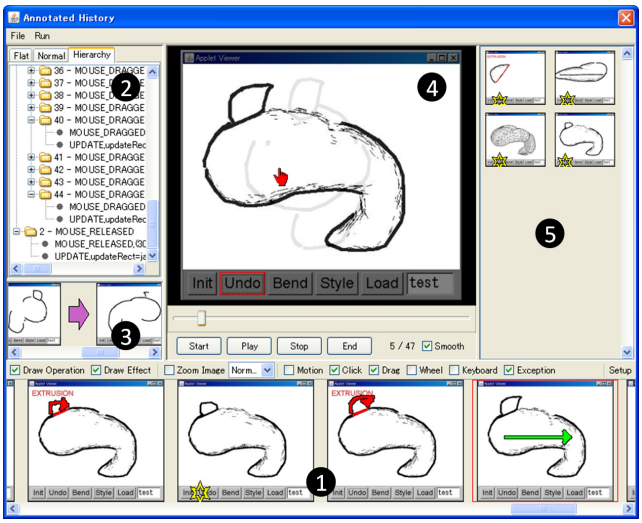

Figure 13: Prototype system for browsing the annotated history: (1) storyboard, (2) event list, (3) thumbnail list, (4) preview, and (5) search results.

Figure 13 shows a screenshot of the annotated history browser. The system consists mainly of two working spaces. The lower part is a storyboard area for visualizing the annotated user operation history, and the upper part has areas for surveying details and focusing on a scene in the annotated history. To the left is an event list and a thumbnail list that display recorded events and snapshots. In the middle is a preview area that shows a scene using animation. To the right is an area that displays search results, described later.

Reviewing an Annotated History

In the storyboard, elements of an annotated history are arranged chronologically to make it easier to understand the location of the scene in the history. Users can right-click on a scene to switch between two display formats: one displaying the entire targeted window and the other displaying only the region of interaction (Figure 14).

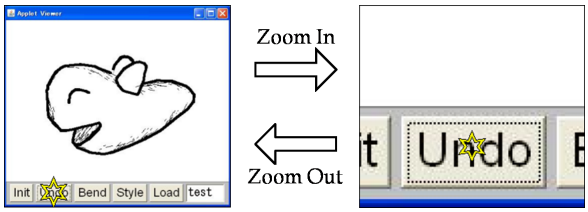

Figure 14: Two display formats of an annotated history. A view of the entire targeted window (left) or only an operation-related region (right).

In the preview area, users can preview a scene by manipulating a slider and several buttons. Interactions executed in the scene are simulated by a pseudo mouse cursor, word balloon, and other features. During previewing, the operation-targeted GUI component is highlighted.

Searching an Annotated History

The preview area also works as a query area for searching. To search for specific interactions, users select the target component regions by directly clicking or rubber-banding the preview image. The system displays only the scenes that include user interactions related to the selected components in the result area. The focus of the storyboard area moves to the corresponding location when users click a scene in the search results.

This feature is designed to improve the efficiency of reviewing enormous numbers of user interactions in an annotated history. It is particularly useful when users know where the events occur but do not explicitly remember the details of user interactions or visual states at that point in time.

Restoring Previous Execution States

As with previous systems that record and visualize operation histories [17], our system also provides a function that can actually restore the execution states of each scene. To use this function, users click the menu command and select the scene to restore. Then the system automatically restores the execution state of the scene. After restoration, users can continue to interact with the target application and restore other execution states.

This is particularly useful for debugging tasks. Even if the target application does not provide undo/redo functionality, it makes rollback of application execution possible by re-playing all recorded events from the beginning [17]. This helps users investigate application execution previous to the point of failure.

USER STUDY

We conducted a user study to evaluate the performance of our visualization technique applied to several different applications. Our aims were to examine whether users could successfully understand the annotated history, and whether visualizing user interactions as annotations would improve user performance. Participants had to locate specific user interactions or application states in the operation history. Our main hypothesis was that the annotated history would lead to a better understanding of user activities, and outperform conventional visualization techniques in visual search. We did not apply manual clustering in this study and each snapshot corresponded to an individual low-level event.

Apparatus

The experiment was run on a laptop computer with a 1.20 GHz Pentium M processor and 1024 MB of memory. The resolution of the screen was 1024×768 pixels. The interface used in this experiment was implemented using Java™ 6.0 running on Windows XP SP2 Professional. Participants interacted with the system using a standard mouse and keyboard.

Participants

Twelve university students ranging in age from 18 to 27 years old participated. Six of them were graduate students in computer science and user interface researchers (expert users). The other six were undergraduate students non in computer science who rarely used computers (inexperienced users). None of them had previous experience with visualization using an annotated history. Each participant worked on the task individually.

Task

The task was to find a figure that represented user interactions or visual states in the target application, based on certain questions. Each task proceeded as follows. (1) Users viewed a video showing how the target application operates, and visual states were changed accordingly. The length of the video was about 150 seconds. Users were allowed to control the playback of the video freely and watch it as many times as they wished. (2) The system provided an operation history explaining an operation sequence (Figure 15). Each operation was displayed as a single still image and the history consisted of approximately 50 operations. (3) Users clicked a Start button, and a dialog appeared. The dialog contained text and an image describing an operation (Figure 16). The descriptive images were drawn manually to illustrate the target operations and were not system-generated snapshots. This may have added a bias in favor of our system, but we chose this because images seem to be the best lightweight way to specify an operation. Users clicked an OK button to close the dialog and began to search for the corresponding image in the operation history. Users clicked a Find button after selecting the image, and then a sound was played and another question appeared if the selected image was correct. If a mistake was made, an error message and warning beep were displayed, and the same question was repeated. (4) After answering ten questions, the search task was complete.

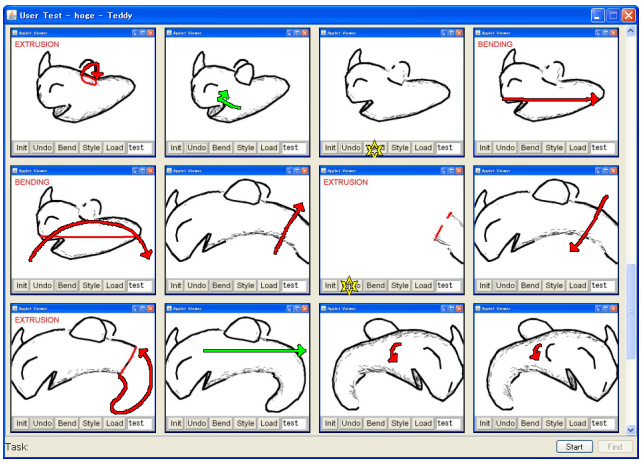

Figure 15: Screen snapshot of the searching task using the annotated history.

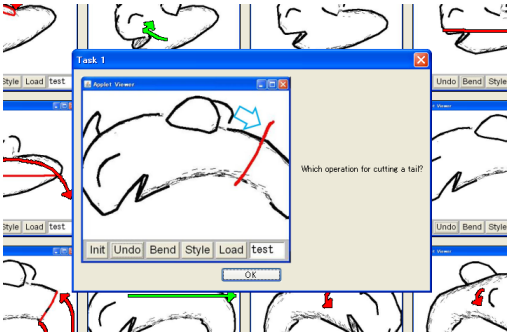

Figure 16: The question dialog for each task.

Interface

Two interfaces were presented, *annotated history* and *snapshot history*. They were identical, except that each user interaction in the former was depicted as an annotation on a static thumbnail image. Users could zoom in on regions of user interactions by right clicking.

Design

Each participant worked within both interfaces. Participants were divided into two groups of six (each with three expert and three inexperienced users). One group used the annotated history first and the other group used the snapshot history first. In addition, each participant worked with three different applications with each interface, for a total of six trials (3 applications × 2 interfaces). The first target application was a sketch-based three-dimensional (3D) modeling system named Teddy [12] (Figure 17a). The second was an icon manipulation system that mimicked a standard desktop workspace in a modern window system (Figure 17b). The last was a window containing a mix of standard GUI widgets: check boxes, combo boxes, and sliders organized in a regular 4×4 grid (Figure 17c). All participants worked on the tasks in this order.

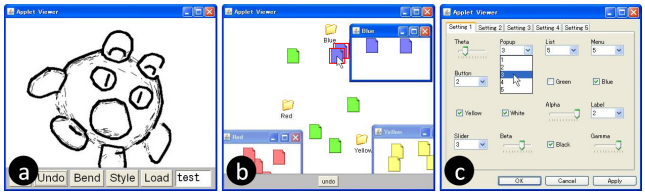

Figure 17: Three target applications in the user study: (a) 3D modeling system, (b) icon manipulation system, and (c) standard GUI widgets system.

First, we gave a tutorial explaining the task and how each operation was denoted as an annotation. Then participants were given time to learn how to use each target application. Then the trials began, and participants were asked to complete the trials as quickly and accurately as possible. For each trial, we recorded the completion time and number of errors. Completion time was measured from the moment the user pressed the dialog's OK button until the moment the user hit the Find button. The error was the number of times the user clicked the Find button with the wrong answer. At the end of the experiment, participants completed a questionnaire assessing their performance. The experiment lasted approximately 90 minutes per participant.

Results

We summarized and analyzed the data, taking the means of completion time and error over the ten questions for each trial. We used a 2 (Interface) × 3 (Application) × 2 (User Type) analysis of variance (ANOVA) on each of the dependent variables, completion time and error rate.

There was a significant main effect of interface type on completion time ($F_{1, 648} = 13.41$, $p < .001$), with participants completing the task significantly faster with the annotation interface than with the snapshot interface. The mean completion times for all applications and user types were 15.73 seconds for the annotation interface and 24.70 seconds for the snapshot interface. There was also a significant effect of user type on completion time ($F_{1, 648} = 4.94$, $p < .028$), as expected. The overall completion times were 17.49 seconds for expert users and 22.94 seconds for inexperienced users. In addition, there was an interaction between interface type and application type. There were no significant differences among applications.

Figure 18 shows the average completion time and standard deviation of the two interfaces for each application. The annotated history generally outperformed the snapshot history. Especially in the 3D modeling system, task completion time was influenced more by annotation (approximately 35% completion time) regardless of the level of computer skill. These results show that an annotated history can be very useful for applications that require operations such as sketches and gestures. In contrast, the GUI widgets system, which mainly requires click operations and has local visual transitions for operations, was less affected by annotation. To determine the detailed characteristics of annotated histories would require further larger-scale investigations (e.g., GUI widgets in a 10×10 grid).

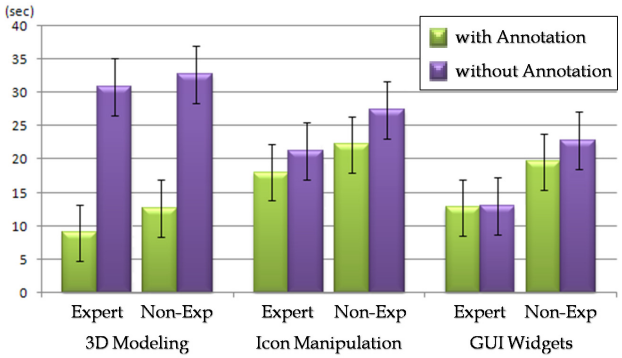

Figure 18: Average trial time under each interface condition (time in seconds, +/- standard error of the mean).

The error rate metric also revealed interesting effects of all factors, including interface ($F_{1, 648} = 4.14$, $p < .043$), user type ($F_{1, 648} = 5.96$, $p < .016$), and application ($F_{2, 648} = 4.17$, $p < .017$). The mean error rates for all applications and user types were 14% for the annotation interface and 20% for the snapshot interface. Moreover, the overall error rates for applications were 15% for the 3D modeling system, 23% for the icon manipulation system, and 14% for the GUI widgets system. For user types, the mean error rates were 14% for expert users and 20% for inexperienced users. There were also interactions between interface and application, and application and user type.

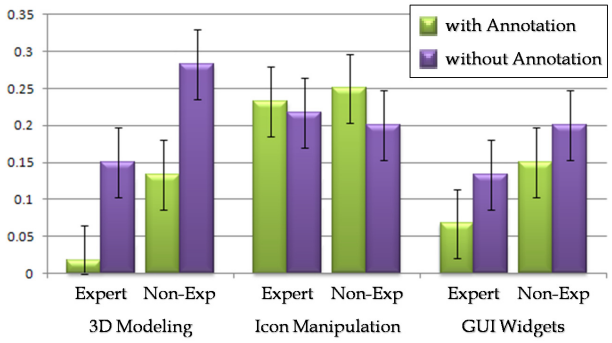

Figure 19: Mean error rate under each interface condition with standard error.

The error rate and standard deviation by interface and application are shown in Figure 19. The annotated history led to lower error rates for the 3D modeling system and the GUI widgets system. This indicates that annotation provides more user-friendly and precise depictions, which reduce user mistakes. Interestingly, the interface with annotation led to a higher error rate for the icon manipulation system, whereas the mean completion time for the icon manipulation system was reduced by using annotation. User behavior during the study and the post-test interview revealed that several users, mainly inexperienced users, unintentionally executed simple click operations such as menu open and icon select in the standard desktop environment without consciously knowing the correspondence between buttons and operations. Thus, they first treated the annotations for right and left click equally and made mistakes.

There was also a case where the annotations overlapped with the objects in the scene and the user did not notice these hidden objects and then made a mistake. It might be possible to reduce such mistakes by using a semitransparent depiction.

According to the questionnaires, all participants preferred the annotation interface over the snapshot interface. Furthermore, they all answered "yes" to the question, "Was the concept of annotated history easy to understand?" On a five-point Likert scale, participants agreed that annotations were more intuitive depictions for user interactions (mean value: 4.67) and that annotations helped users recognize interactions and application states (mean value: 4.75). In addition, some participants requested combinations with textual command lists.

Discussion

The user study indicated that annotation made it easier to track and review user interactions and visual transitions. In particular, the use of the annotated history in the gesture-based 3D modeling system resulted in higher performance for both trial time and error rate. On the other hand, in the traditional GUI widgets system, the annotated history was only as efficient as snapshot history for task completion time despite the superior error rate.

These differences among applications suggest that the performance of annotated history in visual comprehension largely depends on two factors, namely, the complexity of user operations and the scale of visual transition caused by the operations. The 3D modeling system used in this experiment required sketch-based expressive operations and changes in visual states for the operations. In contrast, the GUI widgets system used in this experiment required simple click or drag operations, and the visual transitions were very local. Therefore, we conclude that the annotated history method is well-suited for applications in which the operations are expressive and the visual transitions caused by the operations are large.

CONCLUSIONS AND FUTURE WORK

We presented a method to capture and visualize the operation history of an arbitrary target application by observing its event queue. Our visualization method combines a comic strip metaphor that shows the operation sequence with annotations that illustrate the details of individual operations. We implemented a prototype system for Java Awt/Swing applications and applied it to several example applications. Our user study demonstrated that using an annotated history can improve user performance in a searching task compared to standard static snapshots. Although our results do not guarantee the superiority of our technique in more complex cases or in other types of applications, the results show that an annotated history helps users recognize the order of operations and how visual transitions occur.

There are many possible directions for further exploration. One is to improve the visual quality of our results. Our current system eliminates unnecessary operations for visualization and groups multiple operations together. Other systems treat only low-level operation types, such as click and drag, and lack the quality most users seek. One interesting future direction is to consider the behavior specific to each GUI component. Almost all GUI components have specific actions for each operation. For example, pressing or clicking a button are significant operations while dragging is meaningless. In contrast, dragging the slider or scroll bar is significant. By considering these aspects, more meaningful operations and some filtering and clustering tasks performed by users may be easily automated.

Some test users said that they wanted to see more abstract summarized images. The level of abstraction in our prototype implementation is the same for each element of annotated history, and users must view the entire previous annotated history sequence step-by-step to obtain prior information. We would like to extend the annotated history toward adaptive entities that change their abstraction level dynamically according to focus, context, size, and so on. For instance, the image in focus depicts a short span of operation history and the image out of focus visualizes a long span of operation history. Extensions of this sort would allow users to recognize both details and an overview of the operation history without reviewing many images.

ACKNOWLEDGEMENTS

We would like to thank participants of the user study. We would also like to thank the UIST reviewers and committee members for detailed feedback which improved the paper.

REFERENCES

1. Agrawala, M., Phan, D., Heiser, J., Haymaker, J., Klingner, J., Hanrahan, P., and Tversky, B. Designing Effective Step-By-Step Assembly Instructions. In *Proceedings of SIGGRAPH*, pp. 828-837, 2003.
2. Agrawala, M., and Stolte, C. Rendering Effective Route Maps: Improving Usability Through Generalization. In *Proceedings of SIGGRAPH*, pp. 241-249, 2001.
3. Assa, J., Caspi, Y., and Cohen-Or, D. Action Synopsis: Pose Selection and Illustration. In *Proceedings of SIGGRAPH*, pp. 667-676, 2005.
4. Baudisch, P., Cutrell, E., and Robertson, G. High-Density Cursor: A Visualization Technique that Helps Users Keep Track of Fast-Moving Mouse Cursors. In *Proceedings of INTERACT*, pp. 236-243, 2003.
5. Baudisch, P., Tan, D., Collomb, M., Robbins, D., Hinckley, K., Agrawala, M., Zhao, S., and Ramos, G. Phosphor: Explaining Transitions in the User Interface Using Afterglow Effects. In *Proceedings of UIST*, pp.169-178, 2006.
6. Bergman, L., Castelli, V., Lau, T., and Oblinger, D. DocWizards: a system for authoring follow-me documentation wizards. In Proceedings of UIST, pp. 191-200, 2005.
7. Berlage, T. A Selective Undo Mechanism for Graphical User Interfaces Based On Command Objects. In *Proceedings of CHI*, pp. 269-294, 1994.

8. Bezerianos, A., Dragicevic, P., and Balakrishnan, R. Mnemonic Rendering: An Image-Based Approach for Exposing Hidden Changes in Dynamic Displays. In *Proceedings of UIST*, pp. 159-168, 2006.

9. Freeman, W.T., and Zhang, H. Shape-Time Photography. In *Proceedings of CVPR*, pp. 151-157, 2003.

10. Goldman, D.B., Curless, B., Salesin, D., and Seitz, S.M. Schematic Storyboarding for Video Visualization and Editing. In *Proceedings of SIGGRAPH*, pp. 862-871, 2006.

11. Guimbretie're, F., Dixon, M., and Hinckley, K. ExperiScope: an analysis tool for interaction data. In *Proceedings of CHI,* pp.1333-1342, 2007.

12. Igarashi, T., Matsuoka, S., and Tanaka, T. Teddy: A Sketching Interface for 3D Freeform Design. In *Proceedings of SIGGRAPH*, pp. 409-416, 1999.

13. Irani, M., and Anandan, P. Video Indexing Based on Mosaic Representations. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, Vol. 86, No. 5, pp. 905-921, 1998.

14. Kaptelinin, V., Mantyla, T., and Astrom, J. Transient Visual Cues for Scrolling: An Empirical Study. In *CHI '02 Extended Abstracts*, pp. 620-621, 2002.

15. Kawagishi, Y., Hatsuyama, K., and Kondo, K. Cartoon Blur: Non-Photorealistic Motion Blur. In *Proceedings of CGI*, pp. 276-281, 2003.

16. Kim, B., and Essa, I. Video-based Nonphotorealistic and Expressive Illustration of Motion. In *Proceedings of the CGI*, pp. 32-35, 2005.

17. Kurlander, D. and Feiner, S. A history-based macro by example system. In *Proceedings of UIST*, pages 99-106, 1992.

18. Lieberman, H. Mondrian: A Teachable Graphical Editor. In *Watch What I Do: Programming by Demonstration*, pp. 341-358, 1993.

19. Little, G., Lau, T. A., Cypher, A., Lin, J., Haber, E. M., and Kandogan, E., Koala: capture, share, automate, personalize business processes on the web, In *Proceedings of CHI*, pp. 943-946, 2007.

20. Massey, M., and Bender, W. Salient Stills: Process and Practice. *IBM Systems Journal*, Vol. 35, No.3-4, pp. 557-573, 1996.

21. Masuch, M., Schlechtweg, S., and Schulz, R. Speedlines: Depicting Motion in Motionless Pictures. In *SIGGRAPH '99 Conference Abstracts and Applications*, pp. 277, 1999.

22. Meng, C., Yasue, M., Imamiya, A., and Mao, X. Visualizing Histories for Selective Undo and Redo. In *Proceedings of APCHI*, pp. 459, 1998.

23. Modugno, F. and Myers, B. A. Pursuit: graphically representing programs in a demonstrational visual shell. In *Conference Companion of CHI*, pp.455-456, 1994.

24. Myers, B.A., McDaniel, R.G., Miller, R.C., Ferrency, A.S., Faulring, A., Kyle, B.D., Mickish, A., Klimovitski, A., and Doane, P. The Amulet Environment: New Models for Effective User Interface Software Development. *IEEE Transaction on Software Engineering*, Vol. 23, No. 6, pp. 347-365, 1997.

25. Mynatt, E.D., Igarashi, T., Edwards, W.K., and LaMarca, A. Flatland: new dimensions in office whiteboards. In *Proceedings of CHI*, pp. 346-353, 1999.

26. Rekimoto, J. Time-Machine Computing: A Time-Centric Approach for the Information Environment. In *Proceedings of UIST*, pp. 45-54, 1999.

27. Su, S. Visualizing, Editing, and Inferring Structure in 2D Graphics, *UIST 2007 Doctoral Symposium*, 2007.

28. Taniguchi, Y., Akutsu, A., and Tonomura, Y. PanoramaExcerpts: Extracting and Packing Panoramas for Video Browsing. In *Proceedings of MULTIMEDIA*, pp. 427-436, 1997.

29. Teodosio, L., and Bender, W. Salient Video Stills: Content and Context Preserved. In *Proceedings of MULTIMEDIA*, pp. 39-46, 1993.