Graphstract: Minimal Graphical Help for Computers

Jeff Huang
Yahoo
Sunnyvale, CA 94089
jeffhuangillinois@yahoo.com

Michael B. Twidale
Graduate School of Library and Information Science
University of Illinois, Urbana-Champaign
Champaign, IL 61820
twidale@uiuc.edu

ABSTRACT
We explore the use of abstracted screenshots as part of a new help interface. Graphstract, an implementation of a graphical help system, extends the ideas of textually oriented Minimal Manuals to the use of screenshots, allowing multiple small graphical elements to be shown in a limited space. This allows a user to get an overview of a complex sequential task as a whole. The ideas have been developed by three iterations of prototyping and evaluation. A user study shows that Graphstract helps users perform tasks faster on some but not all tasks. Due to their graphical nature, it is possible to construct Graphstracts automatically from pre-recorded interactions. A second study shows that automated capture and replay is a low-cost method for authoring Graphstracts, and the resultant help is as understandable as manually constructed help.

ACM Classification Keywords
H.5.2. Information interfaces and presentation

General Terms:
Design, Documentation, Human Factors, Experimentation

Author Keywords
software help, minimal manuals, graphical abstracts

INTRODUCTION
Current software help relies substantially on textual explanations to guide the user in performing correct operations. Various studies have shown that users are unlikely to read carefully, if they read help text at all [17]. Users are often looking for a quick way to make a slight change to their work, and may be unwilling to risk the investment of time and effort to read a lengthy explanation. Even when users are highly motivated, conventional help is mostly textual, which can be difficult to apply to the 2D graphical user interface environment of the application due to the dimensional mismatch between text and graphics.

UIST'07, October 7-10, 2007, Newport, Rhode Island, USA.

Very little work [19] has been done on graphical minimal manuals, a concept we believe has great potential. We have developed methods for creating abbreviated forms of graphical help using layered elements of the application window, inspired both by the metaphor of textual abstracts, and Carroll's Minimal Manuals [5,6]. The Minimal Manual approach involves making terse manuals that guide the user with specific tasks. The aim is to support individuals helping themselves as well as users helping their peers. We believe that this initial exploration of the design space of minimalist graphical help shows great promise, not just for better online help but as a lightweight method for supplementing informal peer support [21].

The ideas have been developed by three iterations of prototyping and user studies of Graphstract, a minimalist graphical help system. An initial study of the idea of graphical abstract help was reported earlier [10]. This paper describes our ongoing iterative prototyping and evaluation work. To clarify which version of Graphstract we are discussing, the prototype described in [10] will be referred to as *GS1*, and will be contrasted with two subsequent iterations, *GS2* and *GS3*.

CURRENT HELP FORMATS
In this section, we list formats used for software help, and discuss the problems with each.

Text-centric Help
Printed documentation and online help attempts to combine text and graphics but often relegates graphics to a secondary role. Thumbnail-sized pictures of a specific control or a relevant dialog box are used only to supplement textual descriptions. In addition, the images in text-centric manuals are presented as individual tokens of information, instead of a step-by-step solution to achieve a task.

However, users may not want to spend the time and effort reading through text [17] or may be unable to understand the terminology used in the text help. They have difficulty translating text from the help to widgets on the screen [12]. They can lose their place with the text instructions and miss crucial steps [11]. Creating text help documentation is also costly, magnified by translation costs if the application is targeted at international markets. We think there is a better way to create and present help information.

Minimal Manuals

Minimal Manuals were developed to counter the lengthy and ineffective help documentation provided with most computer software. By severely cutting down the size of the documentation, and providing task-oriented instructions, a more compact type of help is developed. Minimal Manuals have been shown to be more effective than standard manuals [3,4,5,6,22].

Traditional Screenshots

Screenshots are frequently used when creating both online help and paper manuals. A common example is the online help website of Internet Service Providers, where screenshots show how to configure an email client. They have many advantages such as clarifying what the end user should do, and what they can anticipate in a complex sequence of actions [8,9]. Nevertheless, a crucial disadvantage is that this method often involves numerous large and complex screenshots which yield far more information than necessary for most users. This approach adds to perceptions of complexity, discouraging the user from receiving help. In addition, the size and number of the screenshots can pose a problem in understanding a task as a whole. Because several pages are required to demonstrate a single task, the screenshots need to be spread out to form a coherent and holistic sense of the entire task. Spreading out a series of screenshots, either in a (large) manual, in the user's mind, or even on a desk is difficult. This difficulty is particularly acute when these help-giving screenshots are viewed on a computer, where users can only see part of the task on the screen at once. Even then, they must repeatedly switch between the screenshot and the application in order to apply the help. This approach makes it harder for users to skim the help if they are looking for just one piece of information. File size may also be an issue depending on the distribution method, since despite image compression, graphics typically take up much more storage space than text.

	Text	*MM*	*SS*	*AN*	*GS*
Support for skimming		X			X
No significant reading		X	X	X	X
Maps to the application interface			X	X	X
Ease of creation					X
Small spatial needs	X	X		X	X
Small file size	X	X			X

Table 1: Comparison of Graphstract with the different popular help formats. MM = Minimal Manuals, SS = screenshots, AN = animations, and GS = Graphstract.

Animation or Video

Animation can be used to provide effective, detailed and step-by-step help to complete a task [1,9,13,19]. Animation may be excellent as a means to introduce unfamiliar ways of working, but it may not always be the optimal way to learn the interface [13]. Animated help does not normally support the multiple different ways of in-depth reading, skimming, selection, and re-reading that are possible with conventional text and graphics. An animation makes it difficult for users to move at their own pace, confining each user to follow a single level of software proficiency - that for which the animation was written. It is frustrating for the advanced user, who may only be missing one key step of the interaction, to have to sit through an exhaustive description of what he or she already knows.

Additionally, although most animated help allows skipping back and forward, by its very nature, it enforces a temporal view of a task. Animation fails to give an explicit synoptic representation of the task as a whole. The steps must be watched in sequence and subsequently remembered if they are to be perceived as a set. This hinders a user's ability to get the bigger picture of the task. This limitation is not a problem for someone who already understands the overall task, but can be unnecessarily confusing for a novice user.

Animation files also take up a lot of disk space. This can be particularly problematic in online access to the help, where the user has to wait to download the help, potentially deterring the user from using the help at all.

GRAPHSTRACT

Applying the Minimal Manuals approach to graphical screenshots of the interface is the core idea of our design. Combining the ease-of-recognition of graphical tokens with the need to convey steps in an interaction sequence, we developed Graphstract (short for "Graphical Abstracts"). Graphstract aims to create a single page of image tokens for a multi-step task, so users can retain their sense of the entire task throughout the process. Images are focused screen captures that center on the actual control where action is required, creating what we call *graphical tokens*. Individual graphical tokens are then joined to form complex task representations. This approach reduces information clutter and naturally saves space. During user studies, some users commented that the graphical tokens were analogous to bolded text in text help. Many users said that they just skim text help for actionable bold words. We take that concept and transfer it to a graphical token for even easier scanning. The graphical representations also reinforce a user's sense that they are making progress by seeing in the help what they are seeing in the application.

In an extreme way, this approach addresses the problem of conventionally verbose and unread descriptions. The design assumes that users do not go beyond one or two help screens; they mostly skim the screen for useful information [7]. The aim is to give the user information for which actions to perform in a graphical environment using a combination of clues that Graphstract presents onscreen. It is possible to envisage versions of the Graphstract concept that combine minimalist graphics with minimalist text. For the purposes of this early exploration of the design space, we chose to restrict ourselves to an extreme version of the Graphstract concept – one that relies entirely on images with no accompanying text at all.

Graphstract employs a metaphor of ripping out parts of screenshots and pasting them into a scrapbook. Each snippet of help represents a step required to perform a task. This metaphor is further extended using features such as cut-out edges and different levels of help. Graphstract may also be thought of as a static instance of follow-me documentation wizards, which embed scripted help into the application (e.g. [1]). However, Graphstract is loosely coupled, not integrated into the target application. Its focus is to help the user learn the task, rather than providing an automated resource to perform the task for the user.

Design Principles

The core idea of constructing minimal graphical help is in capturing the snippet of screen surrounding the interaction in the application. This area is usually where the mouse interacts with controls in the application or keyboard input occurs. Luckily, these elements are often located in close proximity. Controls that are good candidates for capturing in Graphstract include menus, buttons, textboxes, and tabs (Figure 1). The *graphical tokens* are arranged in the help file using the method described below.

Figure 1 shows Graphstract demonstrating how to toggle the auto capitalization feature in Microsoft Word; a complex task because it includes interactions with multiple nested dialog boxes. Graphstract is not intended to be used on its own, but always in conjunction with the application it is supporting, as illustrated in Figure 2. As such, it can exploit the advantages of minimalism by providing pointers to elements of the interface of the main application without having to replicate them entirely.

Layout of Graphical Tokens

Controls are placed relative to their actual location in the application windows. This arrangement gives the user a clue to the control's location in the application. We take the traditional 'lower means later' convention for sequence information and add to it indentation, which serves to signal the opening of a new sub-window (often a dialog box). This provides a condensed overview of an entire task which can be several levels deep in the interface. Many test users' first comments were that they understood the top-to-bottom representation used by Graphstract.

When title bars are present, snippets of menus are positioned directly below them to replicate their placement in the actual interaction space. In early pilot studies, we found that systematic token placement (i.e. fixed distance between each token regardless of token location on the screen) was not as successful as roughly replicating the interaction space. Thus, controls are placed relative to their location in the menu or dialog box. For example, the graphical token of an OK button is placed closer to the right edge in the help, where it can be found on the dialog box. Users in the two user studies stated that the relative positioning of graphical tokens was both obvious and helpful.

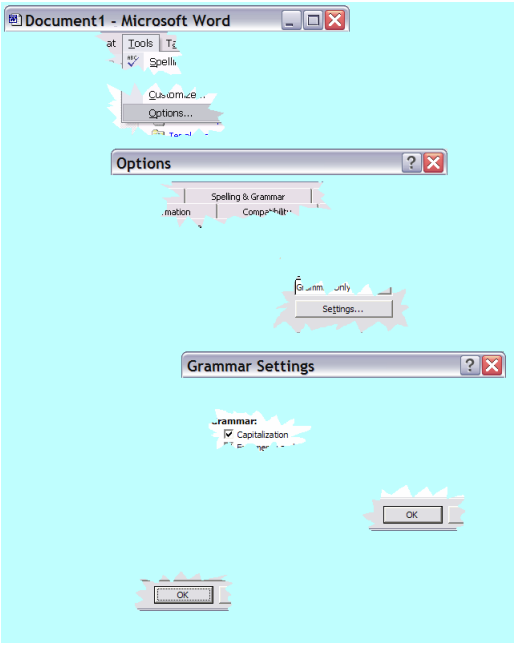

Figure 1: The Graphstract prototype (*GS1*), demonstrating how to toggle auto-capitalization in Microsoft Word.

Bounding Boxes

Graphstract's Bounding Boxes identify which parent object a control belongs to by stacking the layers of dialog boxes together. Examples of parent objects are dialog boxes, applications, or windows, since controls essentially belong to one of these. To create bounding boxes, two designs are possible: title bars and outline bounding boxes.

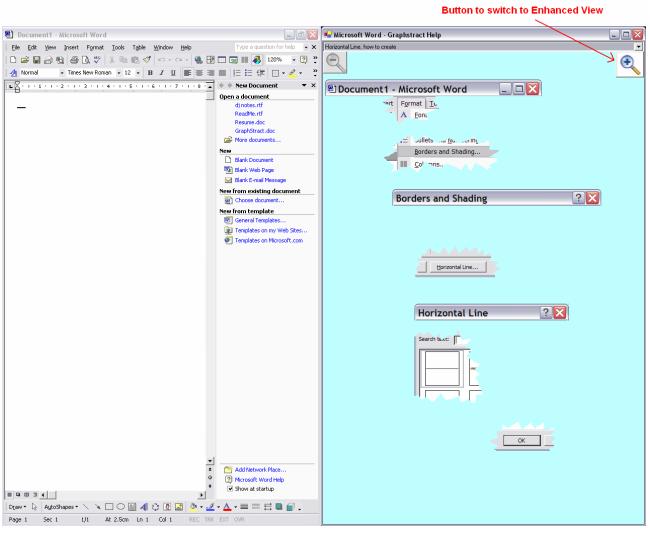

Figure 2: The Graphstract prototype (*GS1*), demonstrating how to insert a horizontal line into a document.

Title bars are graphical cut-outs of an application window's title bar. Figure 1 shows the main Microsoft Word title bar at the top. The approach assumes that the user will be able to tell that the indented Options dialog box title bar is a

child object of the main Word window, and the further indented Grammar Settings dialog box is a child object of the Options dialog box. User testing showed that people understood this convention, but had suggestions for its improvement.

An alternative to title bars is a bounding box outline around all controls belonging to the same parent object. They can be used with title bars, or as replacements. Bounding box outlines were proposed by users commenting on Graphstract after completing the *GS1* user study. An outline of a bounding box clarifies which controls belong to their parent objects, something that may be unclear with title bars (Figure 3). In the user study of *GS3*, users correctly identified bounding box outlines were separate windows.

Edges for Graphical Tokens

We experimented with different types of edges for the graphical tokens: jagged edges, straight edges, and smooth edges. Straight edges were used when a token rested near the border of the interaction space; an example is the OK button on many dialog boxes. In the pilot tests, we found that preserving straight edges where they exist in the UI serves the dual purpose of meshing more closely with what the user sees on the screen (thus reassuring) and hinting about the location of the control. This preservation of edges is especially valuable for an OK button, where the snippet often includes two straight edges that form a corner of the parent dialog box.

Our initial idea was to have jagged edges around the remaining graphical tokens to illustrate that the graphical abstracts were a representation of the interaction rather than a part of the interaction space. The user is given the impression that the tokens are parts of the user interface that have been ripped off the screen and placed together like pieces of a puzzle. From the first user study, we found that users did not quite understand this metaphor, so we decided to use smooth elliptical edges instead, which are also simpler to construct. We believe that the smooth edges will still give the impression that the graphical tokens are cut from the interface and make them appear more natural looking. However, users generally didn't notice the edges during the user study. When asked, they seemed indifferent about the type of edges used, so we believe that the choice of edges on non-bordering graphical tokens is immaterial.

Detailed View

We explored the idea of multiple levels of abstraction in Graphstract, enabling the system to help users with varying levels of experience. *GS1* originally had 3 levels of help: a concise view that presented only the title bar of the dialog boxes, the default view that is the standard placement of graphical tokens on the screen, and a more detailed view consisting of complete screenshots of the application to guide the user all the way. However, pilot tests using the 3 levels of help indicated that the concise view lacked sufficient information to be useful and it was subsequently removed. The default level of help (Figure 1) was designed to allow advanced users to skim through and near-novice users to step themselves through an interaction. Alternately, the detailed view allowed more novice users to see the entire interaction screen step-by-step, using thumbnails to confirm and reassure users. Users navigated between the two views with the + and - magnification icons near the top of the window (visible in Figure 2).

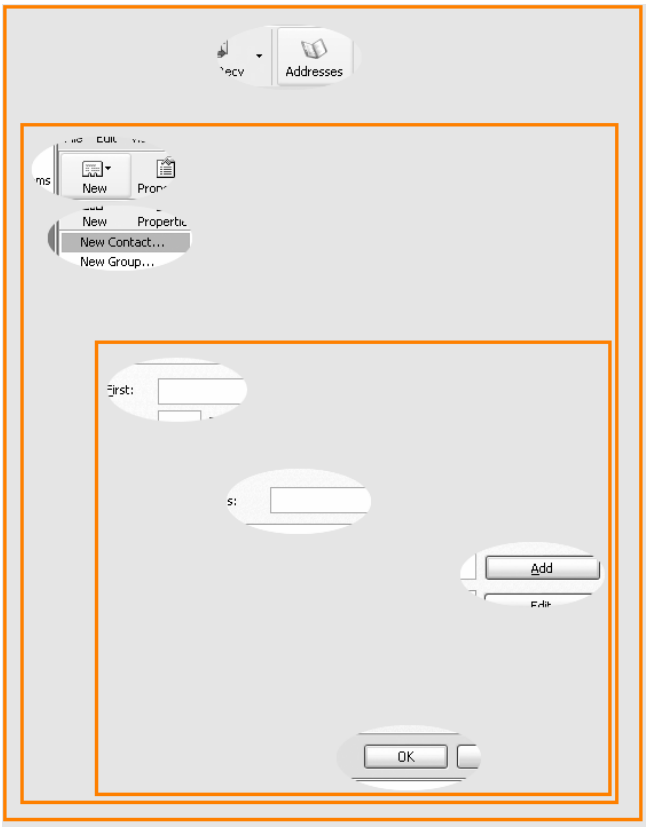

Figure 3: Graphstract (GS3) for adding a contact in Outlook Express, showing bounding box outlines.

Visual Cues

We used small colored dots in the graphical abstracts to draw the user's attention to specific parts of the image. In *GS1*, we used red dots with a blotchy, rather than solid appearance in the graphical tokens to indicate areas of importance. Users from the first user study positively commented on the cues in the *Detailed View*, which guided the user to the location of a control by showing the entire interaction space and highlighting the location of pertinent controls. In addition, the red dots serve as a reassurance in cases where two similarly named controls exist. However, some users were confused because they expected the visual cues to be visible in the actual software as well. In another implementation, we used orange circles and arrows which is more standard practice in existing screenshot help.

Icons were also found to be helpful in providing the user with clues for performing the task when the snippets themselves were unclear or insufficient. An example is the

linked text task from *GS1* (Figure 4), where the user needs to use the right mouse button, rather than the left, to access certain functions, and to drag and drop with the mouse. Icons were placed near the location of the action; in this case, the mouse icon was placed next to the location the user should right-click.

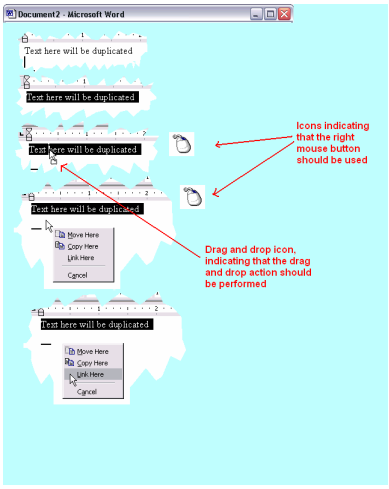

Figure 4: Hint icons that suggest using the right mouse button and drag and drop in GS1.

Color

One of the most visible changes from our first Graphstract implementation to the latest one (GS3) was the removal of color from the graphical tokens. A problem we discovered during the first user studies was that users would often confuse Graphstract with the actual application. Although anyone can tell the difference between a map of Boston and the city of Boston, the application and the 'map' of the application look exactly the same, because the latter is a screen capture of the former. This problem has arisen in past user studies of graphical help where users tried to interact with the images [11]. We found that removing the color from the graphical tokens resolved this issue; a grayscale image no longer looks like the application. Grayscale is used in many interfaces to represent a currently non-interactive element, such as disabled buttons, creating an analogy for our maps of the interface. The user study of *GS3* showed that users were no longer confused about Graphstract as an application; there were no additional comments about the grayscale nature of the help.

Semi-Automatically Constructing Graphstracts

Writing help text is laborious. Having an application create the help is appealing, but is typically complex and error-prone. Previous work done on generating help from text [14,15,16,18] suffers from the problem of "What are the inputs?" as well as the difficulty of actual implementation. It is obviously difficult to automatically create readable instructional text help from an application. However, with graphical help, it is possible to capture the area where a help designer interacts with the application interface. The help designer can simply perform a task on an application, and let a recording application take care of making the correct screen captures and generating the Graphstracts from them. The concept of recording the actions and replaying them for the user as instructional material is the basic idea behind many software-based instructional videos such as Video Professor [23]. DocWizards [1] is similar to Graphstract and allows capture and replay of applications, but generates textual instructions from the scripts. Such a feature can also be used to support informal help-giving between peers in an organization, and the reuse of that help [21]. In such circumstances, the Graphstract may be constructed during a peer help-giving episode, used as a rhetorical device to talk through and clarify certain critical steps of the interaction or concepts necessary to understand the process, and then used as a take-home crib for subsequent re-use by the help-seeker.

In *GS3*, we implemented a system that creates Graphstracts by capturing the correct action sequences as performed by the help designer. Captures are made on each mouse click (down and up button presses) and just before keyboard input. The captured area at the mouse or cursor location is an elliptical token shaped so the width is twice the size of its height. Given the way that text and interface controls are generally structured on a user interface, this method captures a reasonable amount of pertinent information with a fixed shape. A dynamic shape has been suggested but this would have been problematic when the interaction is not with a small control. The generator filters out duplicate screen captures and creates the Graphstracts from the remaining ones. One disadvantage of automatically generated Graphstracts is the difficulty of piecing together graphical tokens that are related, such as the menu bar, menu items, and submenus. The process of selecting a menu item under a submenu is one action, but due to the number of potential clicks, appears as separate graphical tokens. Although there may be more sophisticated methods to decide when to perform an image capture, this naïve method works in most cases to generate understandable minimal graphical help. The automatic construction of the help alleviates much of the tedious tasks of a help designer, but problems do occur. The recorder is unaware of the degree of importance of the surrounding text, so it cannot adjust the size of the captured area. For example, in forms where the field name (e.g. *address*) is followed by a textbox, a snippet of just the textbox looks too similar to the other textboxes in the form. Still, such a tool can also be useful just as a way to suggest what might be recorded, even if this has to be subsequently edited manually.

Generating Multi-Contextual and Multi-Lingual Help

A system that generates help automatically can record the action the help designer performs and then generate the help later. These recorded actions are replayed on another machine and the help is generated in that environment. The advantage is that potentially, for certain interfaces, the

resultant help can be created in the computational context of the help-seeker's home machine, rather than simply being a graphical representation of what was seen on the help-giver's screen. Thus if the help seeker is using a particular interface skin, color scheme, or theme, the provided help will look like the actual applications running on their machine. Furthermore, if they are using an interface with a different language, the help generated will use images derived from that setting too (Figure 5). Because the task recording and replaying tool is blind to the text in the application, it will generate Graphstracts with whatever text is displayed in the environment (Figure 6). This has also been done in text help in [15,18].

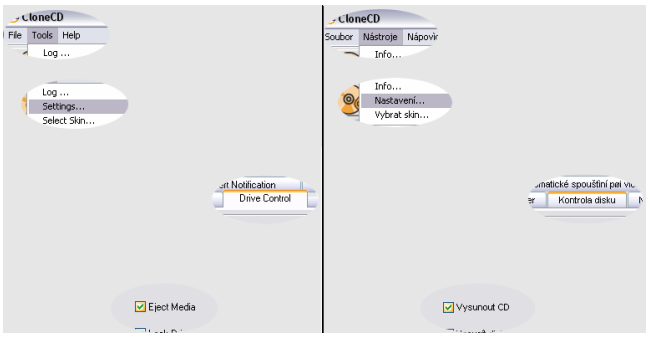

Figure 5: Graphstracts for Disabling CD ejection in CloneCD in English and Czech.

This feature, although compelling is very brittle. Recorded logs are typically rendered useless after a version change in an application. Removing a single control, or changing the order that dialog boxes are displayed will confuse the replay tool and generate incorrect screen captures. We acknowledge there are technical difficulties in implementing a perfect or even consistent record and replay system, but the potential is intriguing.

One design decision to make is 'analog vs digital'. When recording the help designer's actions, are we recording the low-level input (mouse moves 3 pixels left and up 8 pixels) or messages sent to the user controls (Save was clicked)? Recording the low-level input is an analog-like approach, and can cause problems if the positions of the controls are even minutely different in the environment. For example, Microsoft Windows allows users to change the default font size on controls. This may cause a button to be slightly larger and change the position of the interface's controls. A recorded click at a specific location may now just click empty space. Different languages will also have different lengths of the text used in the controls. The menu item labeled "File" in English is "Fichier" in French so takes up more space in French. An analog recording of the help designer clicking on the "Edit" menu in an English version of the software may instead click on the "Fichier" menu if replayed in the French version of the software.

On the other hand, a "digital" recording of interaction may be better, but much harder to implement. Capturing the interaction of a specific control will avoid errors caused by control position changes in the user environment. For example, if "CLICK sent to TEXTBOX3" is recorded, the replay system can find the location of TEXTBOX3 at runtime, move the mouse over to that control, and perform a CLICK action. However, the API (Application Programming Interface) offered by the operating system has to allow for recording of these specific actions. Implementation difficulties include the lack of a notification message sent out in Windows when a menu item is selected, custom controls not sending notification messages when interacted with, and a different action that must be performed for every notification message.

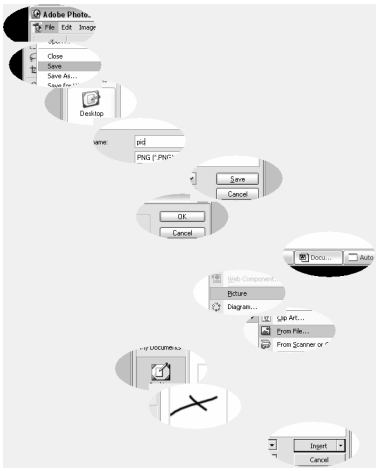

Figure 6: A Graphstract generated automatically using *GS3* showing how to move an image from Photoshop to Word.

Implementation

We implemented three prototypes of Graphstract. Each iteration took the successful design principles of the previous prototype and incorporated new design principles we came up with while studying the previous prototype. Table 2 summarizes the different versions of Graphstract and their respective features.

Graphstract Design Feature	*GS1*	*GS2*	*GS3*
Structured Layout of Graphical Tokens	X	X	X
Visual Cues	X	X	X
Detailed View	X		
Title bars	X		X
Outline Bounding Boxes			X
Grayscale			X
Automated Graphstract Construction		X	X
Static Help Construction	X		X
Dynamic Help Construction		X	X

Table 2: Design Features incorporated in the different Graphstract implementations.

GS1 – The First Generation Graphstracts

The first Graphstract prototype, *GS1* (described in more detail in [10]) was written in C++ using the .NET libraries.

Besides graphical tokens, *GS1* used different levels of help, visual cues, and jagged edges. Graphstract help was created for Microsoft Word with user studies in mind, and the tasks had a wide range of interactions with the application.

GS2 – Constructing Dynamic Graphstracts Automatically

Encouraged by the success of *GS1*, we developed a new version *GS2* that addressed some of the biggest issues identified in the evaluation work. By having a recording application that stored the help designer's interactions, and then creating the Graphstracts during replay, we were able to construct Graphstracts automatically. The time and effort required to create the Graphstracts is now less than that to create the equivalent text help. We also solved the problem of Graphstract's vulnerability to changes in the user environment – a Graphstract created by help designers would look awkward and different on a user's computer if the user had a different theme. Since there are even several different popular themes for Microsoft Windows (Windows Classic, Windows XP, and Windows Vista look), it is likely that the application images in the Graphstracts will be in a different theme from the user's own environment.

We added a module to Jacareto, a Java capture and replay program, which created the Graphstracts when the replaying occurred. The idea is that the help designer would perform the task while Jacareto was running and these captured logs would be the help files packaged with the software application. We created Graphstracts using our modified Jacareto program to record and replay tasks in Greenstone, a Java digital library application. When installed on the user's computer, the Graphstract help would be generated while replaying the actions on the user's computer. An added benefit with this method is that Graphstract can be localized to the user's environment, with matching language and regional settings. However GS2 can only work with certain Java-based applications.

GS3 – A Refined System-Wide Graphstract

We decided to combine automation and grayscale to implement our third Graphstract prototype, *GS3* (Figure 7). *GS3* was written in C++ and used the native Windows API libraries. It incorporated all of the features of the previous versions except for detailed view (Table 2) since in use this was not found to be a popular feature. One key feature of *GS3* was the ability to work across multiple applications. Many computer tasks, such as "instant message a web link to somebody", require more than one application. This is of particular value for contextualized help-giving, in the workplace, for example. Traditional help systems are application specific, but users regularly use multiple applications to get their work done. The implementation of *GS3* was "analog" across the operating system, and so recorded and replayed low-level input blind to the application it was manipulating. One drawback is that the replay will not work if windows are moved in the operating system, since the recording system depends on specific absolute locations of GUI elements.

With *GS3*, we also implemented a version that constructed the Graphstracts immediately after recording the actions by the help designer, rather than constructing the Graphstracts during a replay of the recorded actions. The advantage is that it avoids the problems that arise with delayed replay, such as moved windows or changed settings. However, Graphstracts generated no longer blend into the user's environment, and the translation facility is lost. This is the trade-off consequence of supporting multi-application help.

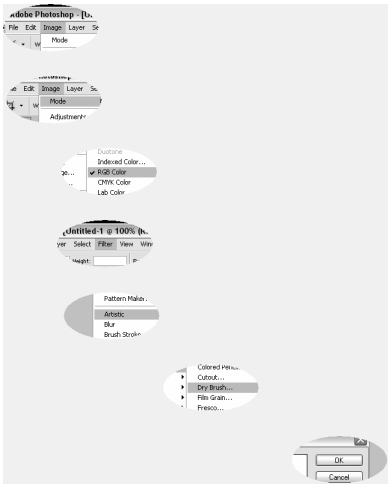

Figure 7: *GS3* automatically generated help for changing the color mode of an image and then applying the "Dry Brush" filter in Adobe Photoshop. The grayscale color shows that the Graphstracts are not actual interactive controls.

USER STUDIES

Our approach for determining the effectiveness of Graphstract was similar to those used in other instructional help user studies [9,22]. Users were observed as they completed tasks using both Graphstract and the built-in text help. The help screen and application were presented side-by-side to avoid the confusion of switching between applications [11]. There were two formative user studies – one for *GS1* to determine the general effectiveness of graphical abstract help, and one for *GS3* to examine the usability of automatically generated Graphstracts.

GS1 User Study

Method

Each user was told that Graphstract was a graphical help system and they could use either Microsoft Word Help or Graphstract (whichever one was presented to them) to help them complete various tasks. We did not explain how to use Graphstract, since we wanted to find out whether they could learn Graphstract for themselves in order to emulate a real-world experience. For each task, we briefed the user on the task, and opened up the corresponding Graphstract or Word Help application. Users were encouraged to talk out loud during the interactions and a questionnaire was administered at the end of the study. Each task was timed to see how long a user took to complete it.

The study was intended to see whether users could understand the Graphstract concept, learn how to use it without training, and how well users performed a task given Graphstract help versus Microsoft Word Help. We wanted to explore users' interactions with the help, rather than the process of locating the right help page which is a separate problem altogether. This sets a high bar for success of the Graphstract concept: it was competing with a very familiar product (the standard Microsoft Help interface), no prior training was provided, and an extremely minimalist form of the Graphstract concept was being tested – one with absolutely no accompanying text. After roughly a dozen rounds of iterative informal pilot tests of Graphstract conducted in small batches, we conducted a more formal study with 20 students in various disciplines.

Each user completed four tasks with Microsoft Word Help, and four tasks with *GS1*. The eight tasks used in the study were broken down into four pairs of similar difficulty. Half of the user group used Graphstract on the first task and Word Help on the second task for each pair of tasks, and vice versa for the other half of users. Microsoft Word was displayed on the left of the screen taking up about 2/3 of the screen, and the help system was displayed on its right. The users performed the task with Microsoft Word and were able to simultaneously use the help system.

The eight tasks used in the formal studies are shown in Table 3, arranged in pairs in order of increasing difficulty (based on the likelihood an average user would have experience with a particular task). The users performed the tasks in order of difficulty starting with the easiest. The tasks were chosen through a brainstorming session by the research group. The goal was to find representative tasks that were often performed in Microsoft Word, which included a diverse set of actions.

Results

Results of the user study on *GS1* are shown in Table 3. For more details about the tasks, see [10]. A t-test of the data from [10] is presented here. For 6 of the 8 tasks there was no significant difference in the results between those using Graphstract or the built-in text help. However, there were 2 tasks where users of Graphstract significantly outperformed those using the built-in text help. Users of Graphstract performed these 2 tasks in about half the time. Using Graphstract or the built-in text help did not affect the success of completing the task; both types of help had the same number of failed attempts.

We hypothesized that learning to use Graphstract with no prior instruction would not impose too much of a performance burden, and that using graphical abstracts would be faster and less confusing for the average user to learn a task. This was confirmed in the study; on average over all tasks, users were able to solve their problems 15% faster when using Graphstract than with the textual Microsoft Word help. However, Graphstract did not help users perform faster in every task.

Graphstract users were slower in tasks 4 and 6 but not significantly; the two help systems had similar performance times in tasks 2, 3 and 8, and users using Graphstract were substantially faster in tasks 1, 5 and 7.

The two tasks where Graphstract was less effective involved changing the text to upper-case and linking text respectively. In task 4, the textual help was very concise and clear about how to change the case of the text. In task 6, users had trouble figuring out the right click drag and drop because Graphstract was unable to show this easily. Another explanation is because these tasks were mainly text focused, and Graphstract lacked the ability to demonstrate features that were text-based because of its graphical nature. The results also indicated that using Graphstract versus textual help did not change the success rate, i.e. the number of times the user gave up.

Task	*Text*		*Graphstract*		*p*
	Avg	SD	Avg	SD	
Highlight a selection of text	68	55	**23****	16	0.012
Switch to outline view	51	83	**40**	50	0.369
Insert a horizontal line	47	32	**44**	29	0.408
Convert text to uppercase	**21**	19	34	40	0.174
Set a keyboard shortcut	244	77	**178**	121	0.098
Create linked text	**81**	103	116	101	0.264
Disable capitalization check	110	56	**45****	25	0.003
Overlap two layers of text	177	84	**167**	84	0.402

Table 3: Task completion times in seconds for the *GS1* usability study

GS3 User Study

Method

A short survey was given to 4 students with computer backgrounds. The objective of the survey was to find out which application would be a good target for comparing help systems. The survey asked for 3 example tasks the participant would want the application's built-in help system to explain. We identified Outlook Express, the built-in Windows email client, and Adobe Photoshop, an image editing application as possibilities for the user study because they are popular Windows applications, with fairly complex tasks for the user to perform. From the survey, we concluded that Outlook Express would be a better candidate for user studies because most participants had less experience with it. The tasks the participants most wanted help for were: message grouping, adding contacts to the address book, adding multiple contacts as a group, and creating message filters. Therefore, user studies were conducted using Graphstract help and the built-in Outlook Express help for those tasks.

The second user study was a simple within subjects evaluation with 17 users. Users who participated in the *GS1* user study were contacted, while the remaining students

were found through message boards or from a Computer Science course and received course credit for participation.

This study aimed to determine if automatically created Graphstracts were a viable alternative to manually created Graphstracts. The Graphstract help was generated by the authors performing the task with the recorder on, and then replaying it to construct the Graphstracts. The users were not told that the Graphstracts in the second study were automatically generated.

The same user study format was used as before. Users were asked to complete 4 tasks (listed in Table 4), 2 using each help system. They were encouraged to talk aloud during the pre-study instructional session and completed a post-survey questionnaire. Half the users completed Group 1 tasks using Graphstract and Group 2 tasks using text help. The remaining users completed Group 1 tasks using text help and Group 2 tasks using Graphstract.

Task	Group
Group messages by conversation	1
Add contact to Address Book	2
Add multiple contacts as a group	1
Create a rule to delete messages with 'spam' in the subject	2

Table 4: Tasks for the user study using GS3

Results

There were 4 users who took part in both user studies – *GS1* and *GS3*. All 4 students said that *GS3* was an improvement over *GS1* in the post-study questionnaire. One user mentioned that it was easier to follow and another commented that it was clearer. Of the 17 users who took the user study, 12 said they liked Graphstract better, 2 said they liked the text help better, and 3 did not mention any preference. The results from this study were similar to those in the first study of *GS1*, and a direct comparison by the 4 users who participated in both studies supported the hypothesis that automatically generated Graphstracts were as effective as the manually constructed ones.

Qualitative results were similar to those in the first study. Users commented that they were better able to skim Graphstract, and that using Graphstract was like following the bolded action words in text help. We also observed that users would get stuck on a step in text help if they did not understand some of its terminology. This doesn't occur in graphical help because the graphical tokens are from the application itself.

In one particular instance of a Graphstract, users were confused by the Graphstract forms; the automatically generated Graphstract cut off the label for the textboxes because it was located to the left of the textbox. The users were unable to distinguish between the different textboxes.

DISCUSSION AND FUTURE WORK

The user studies gave an overall impression that users were able to use Graphstract to solve their problem in less time than using the built-in Microsoft Word text help. On average, Graphstract users were able to speed up the time it took to complete a task in 6 of the 8 tasks (2 were significant). In both post-study surveys, users said that they enjoyed using Graphstract more and thought using Graphstract helped them complete a task quicker; they also stated that they would prefer it to textual help. Since all the users but 4 were first-time users, and they were able to understand Graphstract, we can claim that no substantial learning effort is required. Note that the studies were developed as formative evaluations to inform subsequent exploration of the design space. They cannot and should not be taken as claiming definitive proof that the particular versions of the applications tested are effective.

When implementing Graphstract, we hypothesized that several key features would enhance the user experience. We found that these varied between users. Most understood that the jagged edges meant that each pictorial piece was a snippet; however, there was a minority who were confused by them. Likewise, of the few users that explored the *Detailed View*, most thought the red dots were useful guides, but the rest said that the red dots got in the way. The mouse hint icon used in the linked text task was helpful to some users, but failed for others in indicating that a right-click drag and drop action should be performed. Even those who understood it took a long time to figure it out. We believe that a possible solution is minimal animation, showing a mouse in the drag and drop action, combined with a down-pressed right mouse button.

The post-study surveys also showed that the biggest problem users had with Graphstract was its lack of any text. Most commented that some text to explain Graphstract or guide the user in using Graphstract would be beneficial. In the second study, several users suggested numbering the Graphstracts, even though they understood that Graphstract flowed from top to bottom. We agree with this and believe that help systems should include some guiding text when necessary, based on the complexity of the task. The Graphstract prototype used in the user studies was purely graphical to examine the potential of such systems. A topic for future work is to develop a help system that combines Graphstract and minimal text help.

Graphstract was especially strong in tasks where the user had to navigate dialog boxes and interact with certain controls. From the *GS1* study, tasks 3 (Insert a horizontal line) and 7 (Disable capitalization check), were examples of this. Users were able to complete each of these tasks much faster using Graphstract, and we observed that they performed the tasks more smoothly. Graphstract users completed the 2 tasks in less than half the time as text help users, better than the average improvement for all the tasks. We believe this is because users are able to recognize the dialog boxes and controls easily to discern the next step to complete a task. The visual recognition takes less time than reading text. It is less obvious what to do when presented with an unfamiliar image or one not in the application.

In our second user study of Graphstract, *GS3*, we found that users were able to understand the automatically generated help at least as well as the manually generated help. This is impressive considering some Graphstracts had obvious flaws that a human would correct if manually constructed.

CONCLUSION

The iterative development and testing confirms that minimalist graphical help offers great potential. Graphstracts usually use less space than their screenshot counterparts and are a better way to present information than text-centric help. Although Graphstract has some problems representing certain dynamic actions, it is effective in aiding a user to perform tasks involving dialog boxes and controls. Graphstract's advantage here is that users gain both overview and detail. They are able to skim the help graphics and match them with the actual interface, and so quickly execute the next action step.

The Graphstract prototypes presented in this paper were used to investigate the potential of the idea in a rather challenging comparison. Despite having to compete with conventional help, which users are very familiar with, and despite the additional burden of understanding the meaning and purpose of a novel interface, Graphstract was found to be effective overall, and dramatically better than conventional help for some problematic tasks. Although we do not dispute that some accompanying text is often useful, we wanted to explore graphical help in its purest form, and understand how users react to image-only help. Advantages of pure graphical help include being able to dynamically and automatically generate them from recorded actions, and the potential of creating help in multiple languages.

ACKNOWLEDGEMENT

This work was supported in part by the National Science Foundation under award No. ITR 0081112.

REFERENCES

1. Bergman, L., Castelli, V., Lau, T., and Oblinger, D. DocWizards: a system for authoring follow-me documentation wizards. In *Proc. of UIST '05*, ACM Press (2005), 191-200.
2. Bharat, K. and Sukaviriya, P. Animating User Interfaces Using Animation Servers. In *Proc. of UIST '93*, ACM Press (1993), 69-79.
3. Black, J. B., Carroll, J. M., and McGuigan, S. M. What kind of minimal instruction manual is the most effective. In *Proc. of CHI '87*, ACM Press (1987), 152-162.
4. Brockmann, J. R., The Why, Where and How of Minimalism. In *Proc. of SIGDOC '90*, ACM Press (1990), 111-119.
5. Carroll, J. M., *The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill*. The MIT Press (1990).
6. Carroll, J. M., Smith-Kerker, P. L., Ford, J. R., and Mazur-Rimetz, S. A. The Minimal Manual. *Human-Computer Interaction*, 3 (1987-88), 2, 123-153.
7. Farrand, A. B. and Wolfe, S. J. On-line help: Are we tossing the users a lifesaver or an anchor? *Proc. of CHI '92: Posters and Short Talks*, ACM Press (1992), 21.
8. Gellevij, M. and Van der Meij, H. Empirical Proof for Presenting Screen Captures in Software Documentation. *Technical Communication* (2004), 51(2), 224-238.
9. Harrison, S. A Comparison of Still, Animated, or Nonillustrated On-Line Help with Written or Spoken Instructions in a Graphical User Interface. In *Proc. of CHI '95*, ACM Press (1995), 82-89.
10. Huang, J., Lu, B., and Twidale, M. B. Graphical Abstract Help. In *Proc. of CHINZ '05*, ACM Press (2005), 83-89.
11. Knabe, K. Apple guide: a case study in user-aided design of online help. In *Proc. of CHI '95*, ACM Press (1995), 286-287.
12. Lau, T., Bergman, L., Castelli, V. and Oblinger, D. Sheepdog: Learning procedures for technical support. In *Proc. of IUI '04*. ACM Press (2004), 109-116.
13. Palmiter, S. and Elkerton, J. An Evaluations of Demonstrations for Learning Computer-based Tasks. In *Proc. of CHI '91*, ACM Press (1991), 257-263.
14. Paris, C., Linden, K. V. and Lu, S. Automated Knowledge Acquisition for Instructional Text Generation. In *Proc. of SIGDOC '02*, ACM Press (1992), 142-151.
15. Power, R and Scott, D. Multilingual authoring using feedback texts. In *Proc. of COLING '98* (1998), 1053-1059.
16. Reiter, E., Mellish, C., and Levine, J. Automatic Generation of Technical Documentation. *Applied Artificial Intelligence* (1995), 9, 259-287.
17. Rettig, M. Nobody Reads Documentation. In *Communications of the ACM* (1991), 34(7), 19-24.
18. Rösner, D. and Stede, M. Generating Multilingual Documents from a Knowledge Base: The TECHDOC Project. In *Proc. of COLING '94* (1994), 339-343.
19. Sukaviriya, P., Isaacs, E., and Bharat, K. Multimedia Help: A Prototype and an Experiment. In *Proc. of CHI '92*, ACM Press (1992), 433-434.
20. Sukaviriya, P. Dynamic Construction of Animated Help from Application Context. In *Proc. of UIST '88*, ACM Press (1988), 190-202.
21. Twidale, M. B. Over the shoulder learning: supporting brief informal learning. In *Proc. of CSCW '05,* ACM Press (2005), 14(6), 505-547.
22. Vanderlinder, G., Cocklin, T. G., and McKita, M. Testing and developing minimalist tutorials: A case history. *Proc. of ITCC '88*. Society of Technical Communication (1988), 196-199.
23. Video Professor. http://www.videoprofessor.com/