mudibo: Multiple Dialog Boxes for Multiple Monitors

Dugald Ralph Hutchings, John Stasko
Georgia Institute of Technology/GVU Center
Atlanta, GA 30332 USA
{hutch, stasko}@cc.gatech.edu

ABSTRACT
A general problem identified in recent research on multiple monitor systems is the placement of small windows such as dialog boxes and toolbars. These small windows could be placed on top of the application window or on a monitor next to the application window; different situations call for different placements. We present *mudibo*, a component of the window manager that alleviates this problem by initially placing a window in multiple locations simultaneously and subsequently allowing the user to easily interact with the window in a desired location. Additional important contributions of mudibo are that as a general technique it can be applied to a number of situations and windows beyond simple dialog boxes, exploits the additional screen space that multiple monitors provide to solve a specific problem with dialog box interaction, and is among the first research prototype UIs that explicitly account for multiple-monitor users.

Author Keywords
multiple monitors, window management, dialog box

ACM Classification Keywords
H.5.2 Information interfaces and presentation: User interfaces – *GUI*, *Windowing systems*

INTRODUCTION AND MOTIVATION

Over the last few years we have witnessed the rise in popularity of multiple-monitor computer systems. Such systems present an inexpensive way for a user to double, triple, or further increase the amount of screen real estate available to complete tasks. Several usage strategies for multiple-monitor systems have emerged. For example, Grudin discovered that additional monitors are not "additional space" but "partition digital worlds": people rarely place windows across physical monitor boundaries [1]. However, some users do separate the windows of their applications, using one monitor for the main window and using another window for all of the tool windows [1].

Tool windows are not the only type of small windows that users typically encounter from applications in the standard window managers and operating systems. Among these small window types is the ubiquitous dialog box – a small window that allows a user to customize an application, alter or browse data in an application, view or change system settings, provide meta-information in an application, *etc*. In some cases, such as customizing an application, the ideal spot for a dialog box is on top of the main application window where interaction is already taking place. Interaction is likely to be brief and underlying data is not used, so occlusion of data is not a problem. In other cases, such as browsing or searching through data, the ideal spot for a dialog box is in a location other than on top of the main application window, which usually means on a separate monitor due to a combination of space constraints and users' general avoidance of placing windows across physical monitor boundaries.

It is difficult to predict *a priori* where any given dialog box should be placed. There are two system levels at which the assignments could be made: the window manager level and the application level. Since the window manager has no information about whether the dialog box is for changing settings, browsing data, or some other type of dialog box, determining the correct location could be difficult. If the decision is made at the application level, the burden on the application designer is heavy since the designer must not only acquire information of the monitor configuration of the user but also decide for every dialog box in the application what type of dialog box it is and where it should be placed. Even if the designer accomplished this task, there are chances that the designer made the wrong decision because different users may prefer different solutions. It is possible that an adaptive, intelligent approach could relieve the designer of this burden. However, even for a specific type of dialog box, it is possible that in some situations a user wants to place it in one location and in others another location. The adaptive algorithm would have to account for quite a large amount of the user's context.

mudibo is our proposed solution to the problem of correct placement of transient windows. The fundamental idea behind mudibo is breaking the assumption that a window can exist in at most one location; mudibo breaks the assumption by showing a window in multiple locations simultaneously. Although it is a prototype, mudibo emulates a component of the window manager level by monitoring system-wide window activity. When an application shows a transient window such as a dialog box, mudibo replicates the window on each of the monitors. After the user initiates interaction with one of the copies, mudibo automatically hides the other copies. This allows the user to select where the window should be placed each time it appears *without introducing any additional interaction time than what is currently needed* because the user would already navigate to the location where he or she desires to interact with it.

CHI 2005, April 2 – 7, 2005, Portland, Oregon, USA.
ACM 1-59593-002-7/05/0004.

Shortly we will give a number of examples for which this general technique could be and has been applied. Following those scenarios, we describe how the prototype has been implemented and then conclude with a discussion of related and future work.

USAGE SCENARIOS

In this section we will describe general scenarios, specific applications, and special situations for which mudibo can be useful. Although the examples come from our personal experiences, we hypothesize that the reader will have encountered one or more of the examples in his or her everyday interactions. Note that we will sometimes use the more general term *window* in place of *dialog box*. This is because a dialog box is just a one type of window, and mudibo is not necessarily restricted to dialog boxes.

General Scenarios and Examples

There are some times when a dialog box should be close-at-hand for interaction, and should be placed on top of the main application window. Take for example a dialog box in a web browser that alters privacy settings. Interaction in this dialog box likely has no relationship to any web pages being shown in the main browser window and was summoned because a user desired to check and alter the level of privacy in the browser. If this dialog box appears on a faraway monitor, then the user has to navigate quite a distance to change settings. mudibo alleviates this problem of navigation since if the dialog box appears elsewhere, then a copy is placed on the monitor where the main browser window resides. If the dialog box does appear in the correct location, mudibo places no additional burden on the user since the other copies automatically disappear once interaction begins on top of the main window.

An example of an application for which mudibo can be useful is the Mozilla/Firefox web browser. Mozilla stores the most recent absolute coordinates of each dialog box in the application. If the main browser window is initially located on one monitor and subsequently moved to another monitor, or if multiple browser windows are located on multiple monitors simultaneously, mudibo ensures that each dialog box always appears on the monitor of interest to the user, whether directly on top of the browser or elsewhere.

Indeed, there are times when a dialog box should ideally appear elsewhere. For example, consider any "find"-type of dialog box, such as a dialog box that allows the user to search through a document to find a specific word. Locating this dialog box on an alternate monitor allows the user to see the entire context of the found word without forcing the dialog box to move elsewhere.

Microsoft Word is an example of an application that has a "find" type of dialog box. Word's designers realized the importance of seeing the context of found words. The find box moves itself elsewhere if it occludes a found word in the document editor window. This action forces the user to constantly navigate to a new location to interact with the box. When located on an alternate monitor, the find dialog box never moves and thus allows the user to complete the searching or replacing task more quickly. Normally if the dialog box appears on the application window, the user must navigate to the dialog box and drag it to an alternate monitor. With mudibo, the user can navigate directly to the alternate monitor, immediately begin interaction, and avoid any additional navigation.

Specific Applications

There are two particular applications for which we have found mudibo to be quite useful. One such application is instant messaging. In multiple-monitor systems, there is a danger that a user will miss the arrival of a new instant message if it appears on a monitor other than the one on which the user is currently interacting. Furthermore, knowing which monitor the user is viewing can be very difficult without an eye-tracker: even though the focus window might be located on one monitor, the user might be looking at another monitor. Even with an eye-tracker, the user might have risen for a moment to stretch, answer the telephone, *etc*. mudibo helps to ensure that the user sees the new instant message by presenting it on all of the monitors simultaneously (Figure 1). mudibo also allows the user to easily select which monitor is appropriate for the window. If the message will be dealt with in short order, the user might elect to interact with it on the same monitor as the window with which they are working. If the message will lead to a longer conversation, the user might elect to

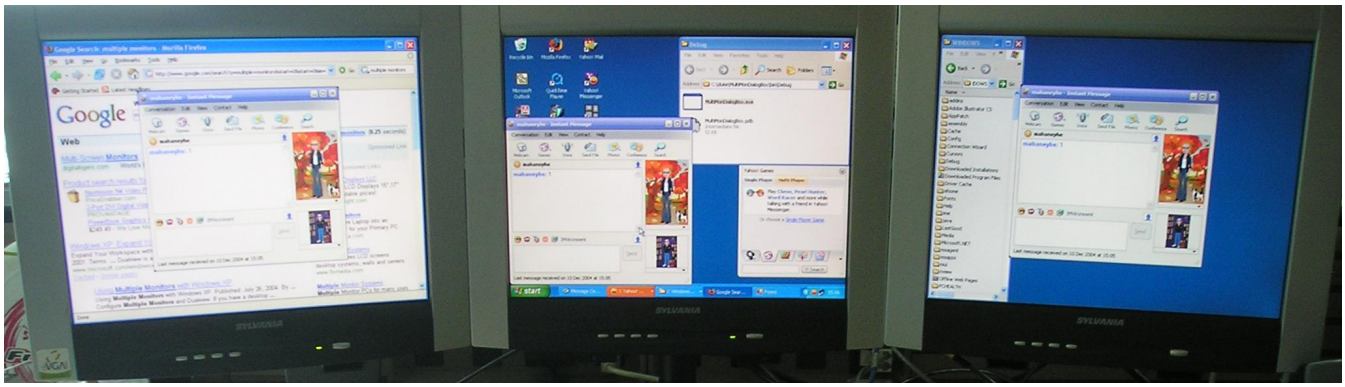

Figure 1. mudibo facilitates the appearance of a new incoming instant message on all three monitors of a multiple-monitor system. Monitor resolution has been decreased for visual effect. Typically, IM windows are much smaller and thus easier to miss.

interact with it on an alternate monitor so as not to occlude the user's main task. A disadvantage of mudibo could be that a user does not want to deal with instant messages when the user is heavily focused on a task. We discuss how this preference can be addressed in the future work section. Many IM clients also show a notification when a user comes online or goes offline and mudibo replicates those notification windows as well. If a multiple-monitor user is interested in seeing notifications, mudibo can decrease the chances that the user will miss them.

Another application for which we have found mudibo to be useful is keyboard window switching. Since mudibo is implemented in MS-Windows, we will discuss the method of pressing <alt>+<tab> to switch windows. When a user presses <alt>+<tab>, a small window showing icons and titles appears and allows the user to switch to another window. As it is currently presented, this window appears on only one monitor. mudibo replicates this window on all of the monitors, which relieves the user of the task of first finding the <alt>+<tab> window before finishing the switch. We also mention here that some video cards hide the fact that a user has multiple monitors and present the screen space as one large monitor. This often has the effect of placing windows (such as the <alt>+<tab> window) across physical monitor boundaries, which can be difficult to read or otherwise undesirable. In our specific implementation of mudibo, we allow the user to define the monitor boundaries. mudibo assesses when an application places a window across monitor boundaries and replicates the window on each monitor (though note that methods such as wideband displays can also address this [4]).

Before discussing two additional specific scenarios in which mudibo has been useful, we pause to remind the reader of mudibo's primary objective: to relieve the application designer of trying to guess the "correct" location for each small subwindow in the application. In each of the scenarios described above, the application could have made a decision to place a dialog box in an ideal location. Our simple technique is powerful in that it (1) relieves application designers from the task of guessing the correct location, (2) automatically hides the incorrect options, and (3) can be used in many, if not most, situations of dialog box placement.

Special Situations

We have found two special situations where this technique has been useful. One situation is when users (attempt to) close a window (call it *X*) and switch focus to a window in another monitor. Later on, the user finds that *X* is still open and discovers that the application has requested that the user take further action before allowing *X* to close (usually to save or discard changes to the data in *X*). At that time, the user might not remember the desired response to the question posed by the dialog box. With mudibo, the application's request to save or discard changes immediately appears on all of the monitors and allows the user to answer the question while the context of the application is still fresh in his or her memory.

The other situation where mudibo has been useful is assisting with "trapped" dialog boxes. Sometimes modal dialog boxes (*i.e.*, those that require interaction and dismissal before the main application window can retain focus) accidentally appear *behind* the main application window. This causes a trap because the modal dialog box refuses to let the main application window have focus, but that main window must be moved in order to dismiss the dialog box underneath. mudibo provides a "hook" or "pointer" to the dialog box by replicating it on different monitors, thus allowing the user to easily relocate the dialog box, interact with it, and finally dismiss it.

SYSTEM IMPLEMENTATION

The mudibo prototype is implemented as an application in Visual C#.NET and runs on any modern Microsoft Windows operating system that has the .NET Framework installed. It uses the P/Invoke platform to access a variety of system resources such as synthetic input generation, window management functions and information, multiple-monitor information, and graphics functions including *PrintWindow*, which gives a full-copy bitmap of a window.

At startup, mudibo collects multiple monitor information or uses user-defined coordinates in cases where video cards hide the details of multiple monitors from Windows. It then monitors the window message queue for the events of window creation and showing. When a window is shown, mudibo acquires information about the window including its *style bits*, which generally define whether a window is a dialog box, tool window, regular window, *etc*. When the style bits match the definition of a dialog box or other window of interest, mudibo uses *PrintWindow* to capture a full bitmap of the window (call it *W*). For each monitor *i*, mudibo creates a topmost "placeholder" window P_i of the exact same size of *W* and places the captured image of *W* as its background. To the user, it appears as if each monitor has a copy of *W*. When a user clicks on any P_i, mudibo moves *W* underneath P_i, hides all of the P_i's, and "sends" the mouse click to *W* by generating synthetic input.

One specific piece of code worth mentioning is the implementation of mudibo with respect to the <alt>+<tab> switching window. Windows generates a special window management event each time the <alt>+<tab> window is *created* and *shown* but not on subsequent key presses. Upon seeing this event, mudibo sets a keyboard hook that allows it to be notified on each additional key press. Then each time the user presses <alt>+<tab>, mudibo re-calls the *PrintWindow* function and updates the image of each P_i.

RELATED WORK

The concept of mudibo is based on previous work regarding multiple monitors. As mentioned, Grudin's qualitative work found that users tend to place windows entirely within monitor boundaries [1], justifying the placement of windows on each monitor and assessing whether dialog boxes were placed across monitor boundaries. Hutchings *et al*. also observed this trend in their quantitative study comparing the window management practices of single-monitor and multiple-monitor users [2]. Mackinlay and

Heer developed a technique to make it easier to interact with windows across physical boundaries [4], but they were more concerned with reading, interpreting, and interacting with images and imaged-based visual interfaces than with interacting with generally text-heavy dialog boxes. Another important aspect of Grudin's work is that he demonstrated that different usage strategies for multiple monitors have emerged. In particular, he found that some users chose to maximize the amount of space that some applications had to display data by moving tool windows to other monitors [1]. That finding directly supports the idea that some subwindows belong near the main application window whereas others belong farther away; mudibo addresses this homogeneity in preferences by providing the initial option to place windows anywhere.

mudibo is closely related to WinCuts [6]. That system allows users to manually select an arbitrary region of a window and replicate the region on another display on the user's system or across a network on a friend's display. mudibo departs from WinCuts in that it is (1) *automatically* replicates contents and (2) captures only full windows. Another closely related system is Ametista, which uses window image capture to transform a 2D window manager into a 3D window manager [5]. Ametista is implemented on top of the X-Windows-like Mac OS X system, showing that the technique of using window images for actual interaction is not limited to just MS-Windows window systems. In fact mudibo should be relatively simple to implement in the Ametista system, further demonstrating its power as a general windowing technique.

FUTURE WORK

To this point we have mostly discussed the advantages of the current mudibo prototype, so we now discuss some disadvantages and opportunities for improvement.

With respect to assisting in the interaction with instant messages, users might prefer to leave instant messages where they appear to minimize distraction to a current task. There might also be other specific types of dialog boxes or windows that users would prefer not to appear on all of the monitors such as material that is private or sensitive in nature (see Hutchings and Stasko for more discussion on this important aspect of window management [3]). We are considering options such as smaller indicators on the other monitors, using change-blindness for notification-style windows like IMs, allowing users to naturally specify types of windows that they prefer not be replicated, and possibly combinations of such techniques.

We have found one particular type of dialog box to be problematic: progress indicators such as those that inform the user how much of a file has been downloaded or how much of an application has been installed. Since there is no interaction with these dialog boxes, replicating their contents across monitors can be annoying because it forces the user to take additional steps to interact when such steps are usually unnecessary. Furthermore a synchronization problem exists because each placeholder window is only an image of the dialog box and is thus likely to convey an inaccurate picture of the progress. This "synch issue" could be assuaged with an actual window manager implementation rather than the use of a separate application, where "live windows" could more easily be maintained. In the current prototype, options include refreshing the copies every few seconds or monitoring for the source's specific window repaint events and refreshing each time the source window is redrawn.

We also need to address various accessibility concerns. For example, there is currently no supported way to use mudibo with only a keyboard. One straightforward solution is to assign keys to monitors to allow initial placement by pressing a monitor's corresponding key.

Formal evaluations of the mudibo prototype will clearly be beneficial. We feel that mudibo can help users' interactions to be more efficient by reducing necessary navigation, but it is possible that the reduction is quite small. Perhaps a stronger benefit of mudibo is the potential to decrease user frustration and reduce the amount of *perceived* navigation necessary for interaction with multiple-monitor systems (as identified by Grudin [1]). As such, the first set of evaluations will probably involve deploying the tool to end users. We will log their actions to try to understand navigation patterns, but focus more on interviews and other researcher-participant interactions to assess how mudibo has altered the way that users interact with multiple monitor systems. That information could be quite useful in building on the small amount of research to date that assesses how people employ multiple monitors.

REFERENCES

1. Grudin, J. Partitioning digital worlds: focal and peripheral awareness in multiple monitor use. *Proc. CHI 2001*, ACM Press, 458 – 465.
2. Hutchings, D. R., Smith, G., Meyers, B., Czerwinski, M., and Robertson, G. Display space usage and window management operation comparisons between single monitor and multiple monitor users. *Proc. Advanced Visual Interfaces 2004*, ACM Press, 32 – 39.
3. Hutchings, D. R. and Stasko, J. Revisiting display space management: understanding current practice to inform next-generation design. *Proc. Graphics Interface 2004*, Canadian Human-Computer Communications Society, 127 – 134.
4. Mackinlay, J. D. and Heer, J. Wideband displays: mitigating multiple monitor seams. *CHI 2004 Extended Abstracts*, ACM Press, 1521 – 1524.
5. Roussel, N. Ametista: a mini-toolkit for exploring new window management techniques. *Proc. Latin American Conf. on HCI 2003*, ACM Press, 117 – 124.
6. Tan, D. S., Meyers, B., and Czerwinski, M. WinCuts: manipulating arbitrary window regions for more effective use of screen space. *CHI 2004 Extended Abstracts*, ACM Press, 1525 – 1528.