Dodging Window Interference to Freely Share Any Off-the-Shelf Application among Multiple Users in Co-located Collaboration

Shinichiro Sakamoto, Makoto Nakashima, and Tetsuro Ito

Department of Computer Science and Intelligent Systems, Oita University
700 Dannoharu, Oita-shi, Oita-ken, 870-1192, Japan
{v0753035,nakasima,ito}@oita-u.ac.jp

Abstract. A method of dodging window interference is described for allowing multiple users to freely share any off-the-shelf single-user application in co-located collaboration utilizing a shared device. This method is indispensable for transparently realizing application sharing in light effort with a centralized architecture by using a surrogate window which is a mimic of the original application's window. Although the original application should process any event on the surrogate window, window interference could be caused by overlapping the location of an event with the surrogate window and then the event cannot be processed. To avoid window interference we formulate the method based on quadrant-based window positioning, in which the original application's window is dynamically repositioned for displaying only one quadrant of this window in one corner of the screen area. The availability of the proposed method was certified and the usability was clarified in co-located collaboration in a university laboratory.

Keywords: Dodging window interference, window positioning, collaboration, application sharing, CSCW, centralized architecture, screen-sharing system.

1 Introduction

Many ways of sharing any off-the-shelf single-user application among multiple users are well documented for computer-supported cooperative work (CSCW). A centralized architecture is employed in most available screen-sharing systems (e.g., [5] and [7]). The application sharing is achieved by centralizing an original off-the-shelf single-user application (an 'original application' in short) and event occurrences onto one PC, and by copying the window image of the original application on each user's PC. This architecture can transparently realize application sharing in distributed collaboration with no specific effort, i.e., without changing the source code of the original application for replicating it. However, the utilization of this architecture in co-located collaboration, where multiple users gather around a shared device, e.g., a tabletop display, has not been studied in depth.

A centralized architecture has the benefit of supporting co-located collaboration by allowing each user to utilize the original application via its surrogate window in his/her preferred location and orientation [1]. Here, each of the surrogate windows works as a mimic of the original application's window (an 'original window' in short) while displaying its copied window image. Each user initiates any event, such as clicking a mouse button, dragging a mouse, pressing a key, etc., on his/her own surrogate window, not on the original window. The original window should, however, be on the top of other windows in order to receive every event on its surrogate window. The events on the surrogate window are then interfered with by the original top window. In order to freely share the original application among multiple users without causing such interference, it is crucial that the original window is repositioned to an appropriate place on the screen area according to the location of the event.

We propose a novel method of *dodging window interference* to freely share any off-the-shelf single-user application among multiple users in co-located collaboration. This method dynamically repositions the original window so that this window dodges the interference with its surrogate windows, allowing the original application to receive the events on the surrogate windows at any given time. To achieve this, the method realizes *quadrant-based window repositioning* which draws only one quadrant of the original window among four quadrants including the corresponding location of the event on the original window as their common origin.

Since the original window has a square shape, it enables us to prove that at least one quadrant exists thereby avoiding window interference even if the event occurs at any location on any surrogate window when the size of the original window is smaller than that of the screen area. It is also possible to minimize the effort required to reposition the original window caused by the later events when the displayed quadrant is far away from the event location on the screen area to avoid window interference.

The rest of this paper is organized as follows: The problems of previous application-sharing systems in co-located collaboration are discussed in Section 2. The requirements and the quadrant-based window repositioning of the proposed method are described in Section 3. We estimated the availability of our method in Section 4 and clarified the usability of the method in co-located collaborative work in Section 5, where we implemented our method into an application-sharing system, CollaboTray [1] employing a centralized architecture.

2 Application Sharing in Collaboration

This section describes previous screen-sharing systems and an advanced application-sharing system based on a centralized architecture. We also discuss the problems they have in co-located collaboration.

2.1 Application-Sharing Systems

For application sharing, screen-sharing systems have been used in practice for almost twenty years, e.g., PCAnyWhere[7], NX[4], and VNC[5]. Among them the open-source screen-sharing system VNC is utilized in many systems for application sharing (e.g. [2], [8], and [9]). Those systems can allow the users to share any off-the-shelf application via its original window and surrogate windows without sharing the whole screen on one PC. A VNC-based toolkit for window managing on X-window system, Ametista [6], allows a user to rotate the window image as the users need in co-located collaboration [3]. Those VNC systems, however, have a problem that the users of the surrogate windows are not able to initiate any event as the user of the original window is operating the original application.

An application-sharing system, CollaboTray [1], can deal with the above problem by making each of the users utilize the surrogate window while its original window is made invisible to disallow use of it by any user. A CollaboTray centralizes an original application on only one PC and decouples the drawing of the surrogate window of an orignal window from the processing of any event on the surrogate window. Any original CollaboTray, which is loaded with an original application, can yield its clone CollaboTrays each of which manages inherently the same surrogate window as the original CollaboTray. The CollaboTray uses the original window in a different way from the previous screen-sharing systems and has advantage of allowing the users to share the original application in any orientation and time.

Figure 1 illustrates the basic approach of realizing application sharing in co-located collaboration with a centralized architecture, where an original window and its two surrogate windows A and B exist. In utilization of VNC, if ownership of the original window is disallowed by any user like CollaboTrays, the users can initiate any event by taking turns among themselves when utilizing the original application via their surrogate windows. However, a common problem arises in application sharing by VNC and CollaboTrays when the original window overlaps with its surrogate windows as shown in the figure. Even if the original window is invisible, the original window needs to be on top of the surrogate windows to receive any event on them. The problem is that the surrogate window owned by a user should also be on top of the original window to allow him/her to operate the original application. This contradiction causes window interference between the original window and its surrogate window. Note that each of the surrogate windows do not interfere with each other since neither is required to be on top of the other surrogate window when its user initiates any event.

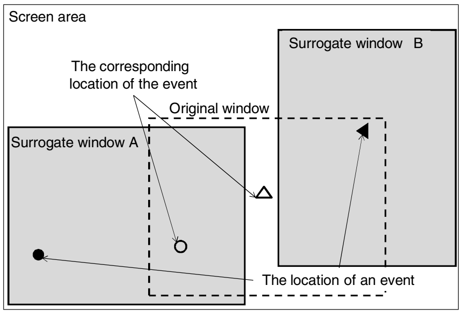


Fig. 1. Application sharing in co-located collaboration

2.2 Window Interference in Co-located Collaboration

The two cases of window interference are illustrated in Fig. 1. In the figure, the location of each event on the surrogate windows is represented by a filled circular or triangular shaped mark. The unfilled ones correspond to the location of the event on the original window. For surrogate window A, the area including the corresponding location of the event on the original window is overlapped with this surrogate window. When the user initiates the event on the surrogate window A, the original window is interfered with by surrogate window A as the user is initiating an event on the surrogate, the original window is unable to get on the top, and thus the event cannot be sent to the original application. Conversely, for the surrogate window B, the area that includes the location of the event on this surrogate window is overlapped with the original window. When the event is sent to the original application, the original window gets on top of surrogate window B and thus the user of surrogate window B cannot initiate his/her next event.

In addition to the above, there is another concern about the feature of the mouse moving. If a user uses a standard USB mouse, the location of the mouse cursor on the screen area is updated every 8 msec. For any event, the location of the next event may jump to the place on the original window causing the kind of interference seen in surrogate window B in Fig. 1. This can occur even if the original window is placed where it can avoid window interference.

3 Dodging Window Interference

This section describes the method of dodging window interference by using quadrant-based window repositioning. We first specify the requirements to avoid window interference and then formulate the method to meet these requirements.

3.1 Requirements

There are two requirements to avoid the window interference mentioned in Section 2.2: (a) to avoid the physical overlapping between the original window and its surrogate window to allow each other to get on top if needed, and (b) to avoid window interference by any fast movement of the mouse cursor. The former leads the following conditions to be satisfied:

C_{a1}: The corresponding location of the event on the original window is outside its surrogate window on which the event occurs.

C_{a2}: The location of the event on a surrogate window is outside its original window.

The latter requirement is possibly avoided by satisfying the following condition:

C_b: The original window stays as far away from the location of an event on its surrogate window as possible.

Conditions C_{a1} and C_{a2} lead us to understand that only the smallest possible area of the original window has to be displayed, which includes the corresponding location of the event. For condition C_b, since the screen area has a square shape, one of the four corners of the screen area is the furthest from the location of any event. Given these facts, we devise a way of quadrant-based window repositioning, which selects a quadrant of the original window with the corresponding location of the event as its origin, and display this quadrant on the corner of the screen area, thus satisfying the above conditions.

3.2 Quadrant-Based Window Positioning

Fig. 2 shows an example of dodging window interference for the case of surrogate window A in Figure 1, in which the original window is repositioned to the top right corner of the screen area. Only the third quadrant, i.e., Q_3, of the original window is selected to be displayed with low opacity on the screen area, where four quadrants including the corresponding location of the event as their common origin exist. When each of the two quadrants, Q_2 and Q_4, is selected, the original window can be repositioned to the bottom right and the top left corners, respectively, as shown in the dashed square in the figure. The original window is, however, the furthest from the location of the event when Q_3 is selected. Selecting Q_3 satisfies C_{a1}, C_{a2}, and C_b. If Q_1 is selected, the original window is repositioned to the bottom left corner of the screen area but condition C_{a2} is not satisfied.

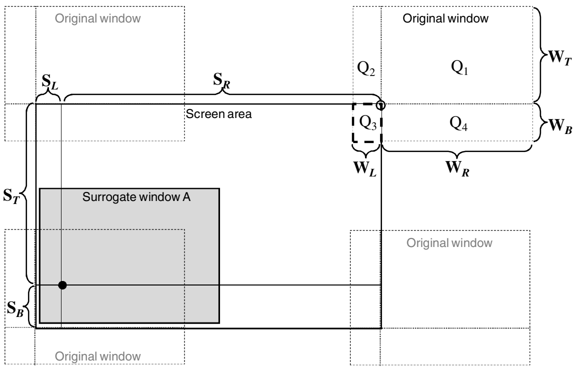


Fig. 2. Dodging window interference

We here call a quadrant Q_i (i= 1,2,3,4) of the original window an *available quadrant* if, Q_i can be displayed so as to satisfy conditions C_{a1} and C_{a2} by positioning its origin on the corner of the screen area in the opposite direction of Q_i. The overall process of dodging window interference between the original and its surrogate window is summarized as follows:

Step 1: Divide the original window into four quadrants with the corresponding location of the event as their common origin.
Step 2: Find all available quadrants from those four quadrants.
Step 3: Select one quadrant Q_i from among the above quadrants, which satisfies condition C_b.
Step 4: Reposition the original window so as to only display Q_i on the screen area.

If the above process can find an available quadrant, we can say that the original window can avoid window interference with its surrogate window. Although the available quadrant can be overlapped with the surrogate window in step 4, the utilization of the surrogate window dose not interfere with the quadrant by making it invisible. Let us prove the robustness of the above process in finding available quadrants. As shown in Fig. 2, let W_L, W_R, W_T and W_B denote the distance of the corresponding location of the event on the original window from the left, right, top and bottom edges of the window, respectively. Also let S_L, S_R, S_T and S_B denote the distance of the location of the event on the surrogate window from the left, right, top and bottom edges of the screen area, respectively. Suppose that both the size of the original and surrogate windows are smaller than the screen area in width and in height, i.e., $W_L + W_R < S_L + S_R$ and $W_T + W_B < S_T + S_B$. We now have two key lemmas in terms of an available quadrant in the situations in which the surrogate window is not rotated and is freely rotated.

Lemma 1: Suppose the size of the original window (and surrogate window) is smaller than that of the screen area. Then at least two available quadrants exist even if the event occurs at any location on the surrogate window.

Proof: Since the screen area has a square shape, the surrogate window can overlap with only one corner of the screen area at a maximal. This implies that there are three quadrants of the original window to satisfy condition C_{a1}. If two of these quadrants, Q_i, and Q_j, do not satisfy condition C_{a2}, then it can be said that the size of each of Q_i, and Q_j is greater than or equal to that of the quadrant of the screen area, which has the location of the event as its origin, in the opposite direction of Q_i and Q_j, respectively. Thus $W_L+W_R \geqq S_L+S_R$ or $W_T+W_B \geqq S_T + S_B$ should be satisfied. This contradicts the supposition about the size of the original window, so we can conclude that there are at least two available quadrants.

What is the influence of rotating the surrogate window to the number of the available quadrants? With regard to this question we have the following lemma.

Lemma 2: Suppose the surrogate window is rotated. Then at most two corners of the screen area are overlapped with this surrogate window.

Proof: Regardless of the supposition about the size of the surrogate window, the diagonal length of this window can be longer than the height and width of the screen area. The surrogate window can thus overlap with the two corners at the same time by rotating it. This rotation can only occur when one corner of the original window is positioned at one corner of the screen area and the window is rotated upon this corner.

We are now ready to formulate the following theorem for the number of available quadrants.

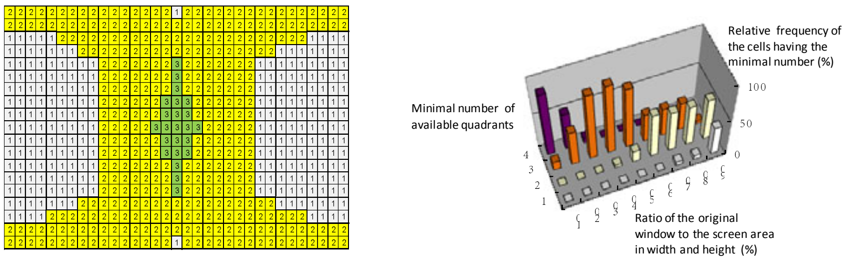
Theorem: Suppose the size of the original window (and surrogate window) is smaller than that of the screen area in width and in height. Then there is at least one available quadrant even if the event occurs at any location on any surrogate window.

Proof: From the proof of Lemma 2, the worst case in rotating the surrogate window is occurred when one corner of the surrogate window is positioned on one corner of the screen area and is rotated on this corner. Even if this occurs, by Lemma 1, at least two quadrants of the original window are available quadrants when the surrogate window is not rotated. Thus it can be said that even if the surrogate window is rotated by a user, either of those two quadrants is still an available quadrant.

4 Estimation of Applicability

We certified the applicability of our method in Section 3.2 by estimating the number of available quadrants in step 2 and the distance of the original window from the location of the event in step 4. For estimation, we set the size of the screen area to the standard size, i.e., 1920 pixels width and 1080 pixels height. The ratio of the original and surrogate windows to the screen area in width and height was varied between 10% and 90%. The surrogate window was displayed by moving it vertically and horizontally every 60 pixels on the screen area.

First, we estimated the number of available quadrants found by our method at every location of the surrogate window. Fig. 3(a) show the distribution of the minimal number of the available quadrants on the screen area, where the ratio of the original (and surrogate) window was 90%. The number in each cell indicates the minimal number of available quadrants when the center of the surrogate window was located in the cell and the window was rotated from 0 to 360 degrees at intervals of 15 degrees on the center, and the event was initiated on each location on the surrogate window. As a result of rotating the surrogate window, the cells, which had only one available quadrant, tended to converge on the left and right hand parts of the screen area. Fig. 3(b) shows the relative frequency of the cells having the minimal number of available quadrants within the total cells on the screen area by varying ratios of the original (and surrogate) window. While the ratio was under 90%, multiple available quadrants were found in almost every location on the screen area. Since each user does not intend to monopolize the screen area in co-located collaboration, it could be said that there is a high possibility to have multiple available quadrants for selecting the place, which can be far away from the location of the event, to reposition the original window.



(a) (b)

Fig. 3. A distribution map of the minimal numbers and relative frequency of the cells having the minimal number of available quadrants

Secondly, we estimated the distance of the original window from the location of each event on the surrogate window when the original window was repositioned by our method. Note that the maximal distance was equal to the diagonal length of the screen area, i.e., about 2202 pixels. Fig. 4 shows the probability of any occurring distance when the ratio of the original (and surrogate) window was 30, 50, or 90 % to the screen area in width and height. By observing the curves for 30% and 50%, as long as a user does not move the mouse cursor too fast (about 500 pixels, half of the height of the screen area, per update), we can say that our method avoids window interference caused by mouse movement at any given time. Even if the ratio was 90%, the possibility of avoiding window interference caused by any such mouse cursor speed was 77.6%. In the next section, we clarified that our method could work well in a prepared working environment without concern for mouse cursor speed and surrogate window size.

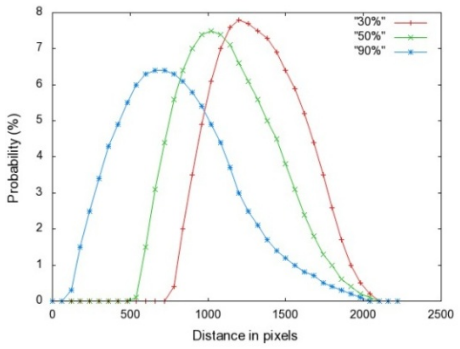


Fig. 4. The probabilities of occurring distances

5 Usability Case Study

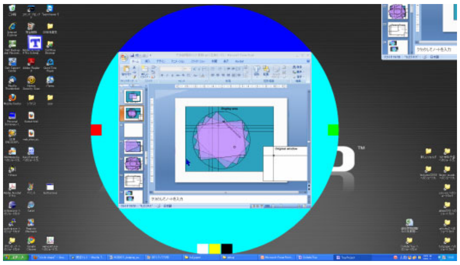
The mouse cursor speed and the size of the surrogate window can be changed by the users and thus there is a possibility of window interference. To clarify the usability, we prepared a working environment to utilize CollaboTrays in which this method was implemented. The PC used in the environment had an Intel Xeon processor running an MS Windows 7.

5.1 The Environment

The implementation of our method into a CollaboTray was easily done by attaching its mechanism to the mechanism of drawing a surrogate window. Fig. 5(a) shows an example screen shot of a new version of a CollaboTray, which has a circular shape, when it was loaded with the surrogate window of an MS PowerPoint. Here, the original window, which was made highly visible in order to highlight its position, was repositioned in the top right corner of the screen area to avoid window interference.

We prepared a working environment in which a university student polished up his research presentation slides through face-to-face interaction with the members of his research group, across a tabletop display connected to a PC and which had the same size screen area as in Section 4. Each of four groups had 3 members and total 12 subjects used the CollaboTrays. Each member of every group utilized his/her own mouse individually and two of them one keyboard. Note that each subject used a standard USB mouse and adjusted the mouse cursor speed as he/she liked before work.

Fig. 5(b) shows a female subject and two male subjects in one research group collaboratively polishing up the presentation belonging to the male on the right. This subject loaded a CollaboTray with an original MS-PowerPoint for displaying his prepared presentation slides, and handed its clone CollaboTrays over to the other subjects. Everyone freely changed the content, magnified his/her surrogate window, rearranged the slide configuration, etc., during their about 60 minute collaborative work.


(a)

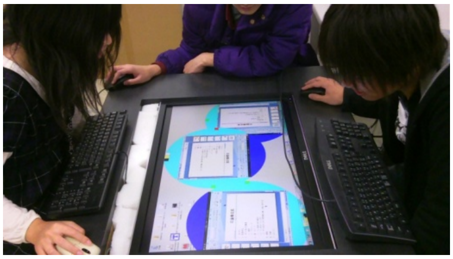

(b)

Fig. 5. CollaboTrays

5.2 Results

Although many subjects magnified the surrogate window on his/her CollaboTray, its average ratio was about 50% to the screen area in width and height. No one magnified his/her surrogate window to the size of the screen area. We also logged the moving distance of the mouse cursor per update (the moving distance for short), and the distance of the original window from the location of every event (the window distance for short). Table 1 shows the average distances during collaborative work performed by each research group. For each group the average window distance was statistically superior to the average moving distance ($p< 0.05$). As a result, window interference never happened. From these observations, we are able to deduce that our method is a practical approach which satisfies the requirements for dodging window interference.

Table 1. The moving distance of the mouse cursor and the distance of the original window from the location of the event

	Group			
	1	2	3	4
Moving distance in pixels	19.6 ± 83.9	19.1±83.0	19.1 ± 88.9	16.3±72.8
Window distance in pixels	1070.0 ±258.9	1029.3±237.9	1205.3 ±233.0	1234.3±169.6

After the work each subject filled in a questionnaire which included two questions: “Were you able to do collaborative work smoothly?” and “Did you feel comfortable with the mouse movement?" Each subject answered on a 5-point Likert scale ranging from very acceptable (scale = 1) to very unacceptable (scale = 5). For each question, over 75 percent of the subjects gave positive answers (scale = 1 and 2) and this percentage was statistically superior to that of negative answers (scale = 4 and 5). From the result of the former question we could say that the repositioning of the original window has the added advantage of being able to freely share the original application and to help boost collaboration. As for the latter question, we could also say that in our method the mouse movement is not problematic.

6 Conclusions

The method described in this paper allows users who are sharing off-the–shelf-single-user applications in co-located collaborations to "dodge window interference." This method is indispensable for developing application-sharing systems based on a centralized architecture. The applicability of the proposed method was certified both theoretically and through practical co-located collaborative work in a university research laboratory. The users of the CollaboTrays employing the proposed method were helped to boost their co-located collaboration.

From this point on we need to improve our method for utilization in distributed collaboration. At present users having an original application will suffer interference from other users when each of them operates another original application through his/her surrogate window on a distributed PC. The applicability of our method should be clarified to realize application sharing in such distributed collaboration.

References

1. Abe, Y., Matsusako, K., Kirimura, K., Tamura, M., Nakashima, M., Ito, T.: Tolerant sharing of a single-user application among multiple users in collaborative work. In: Companion Proceedings of the ACM Conference on Computer-Supported Cooperative Work (CSCW 2010), pp. 555–556. ACM Press, New York (2010)
2. Hank, B.: Empirical evaluation of distributed pair programming. International Journal of Human-Computer Studies 66, 530–544 (2008)
3. Kruger, R., Carpendale, S., Scott, S.D., Greenberg, S.: Roles of orientation in tabletop collaboration: Comprehension, coordination and communication. Computer Supported Cooperative Work 13(5-6), 501–537 (2004)
4. Nomachine, NX, `http://www.nomachine.com/documents/getting-started.php`
5. Richardson, T., Stafford-Fraser, Q., Wood, K.R., Hopper, A.: Virtual network computing. IEEE Internet Computing 2(1), 33–38 (1998)
6. Roussel, N.: Ametista: a mini-toolkit for exploring new window management techniques. In: Proceedings of the Latin American Conference on Human-Computer Interaction, pp. 117–124. ACM, NY (2003)
7. Symantec, `http://www.anyplace-control.com/pcanywhere.shtml`
8. Tee, K., Greenberg, S., Gutwin, C.: Artifact awareness through screen sharing for distributed groups. International Journal of Human-Computer Studies 67, 677–702 (2009)
9. Ultra VNC, `http://www.uvnc.com:8080/`