International Journal of Innovative
Computing, Information and Control
Volume 9, Number 12, December 2013

 ISSN 1349-4198
pp. 4665–4679

REDUCING THE COST OF WINDOW OPERATIONS BY DOCKING WINDOWS

Hirohito Shibata and Kengo Omura

Communication Design Office, Research and Technology Group
Fuji Xerox Co., Ltd.
430 Sakai, Nakai-machi, Kanagawa 259-0157, Japan
{ hirohito.shibata; kengo.omura }@fujixerox.co.jp

Received August 2012; revised January 2013

Abstract. *When we open many windows on computers, it becomes difficult to find necessary windows. Moreover, we must switch windows frequently to compare information from different windows, which degrades work efficiency. To resolve these problems, an extended multi-window system called the Docking Window Framework is proposed herein. In this framework, users can connect multiple windows through a novel interaction technique called docking, which resembles plugging in a jigsaw puzzle piece. Using this feature, users can group windows of the same task together and thereby create workspaces for each task. To evaluate the system, we conducted two experiments. In window arrangement tasks, participants performed tasks faster when using the proposed system than when using a popular window system (Windows XP). Moreover, participants reported that they felt fun to use the system. In the second experiment using task-switching tasks, participants using our system performed multiple tasks in parallel more efficiently. We verified the effectiveness of our approach to support multitasking in computer work.*
Keywords: Window systems, Workspaces, Multitasking

1. **Introduction.** Alto, a personal computer (PC) that included a window system in its graphical user interface (GUI), was produced in 1973. Nearly 40 years have passed since then. Although the visual appearance of windows has changed during that time, the basic behavior of window systems has changed very little. Windows are moved by dragging their title bars and are resized by dragging their window frames. A hidden window is activated by clicking the window. Most people are well familiar with these operations because of the widespread use of PCs. However, are current window systems truly ideal for the operation of digital documents? Cannot further improvement be considered?

When performing a task using computers, one cannot necessarily complete the task using a single document or a single application. People often refer to multiple documents of various applications to perform a single task. For example, when writing an academic paper, people usually make use of their own papers or reports that they wrote previously, documents written by other people, dictionaries, drawing tools to create figures, and spreadsheets to calculate data, as well as a word processor to compose the paper. When programming, people usually use an editor to edit source code, a development environment to compile and debug the code, manuals of libraries, and a web browser to search as quickly as necessary.

In addition, people usually perform several tasks in parallel such as writing reports, exploring information, and reading news or emails. They conduct their work while switching their attention among these tasks. Furthermore, tasks are sometimes interrupted unintentionally by external events such as scheduled meetings or urgent requests from colleagues.

Such phenomena are known collectively as *multitasking*, which is frequently performed in office work [1-5].

In short, people generally need multiple documents to perform a single task and they perform multiple tasks in parallel by switching sets of documents. In this situation, many windows become scattered on the desktop. It becomes difficult to find a necessary window[1]. People must switch foreground windows frequently because many windows mutually overlap[2]. Moreover, people must move and resize windows repeatedly to lay out the windows side by side.

Window operation costs spent for such operations are not negligible by any means. We analyzed window operation logs of workers engaging in intellectual property management in their actual work environment [7]. Results showed that the workers spent 7.4-9.1% of time for window operations such as switching, moving, or resizing windows when they worked on computers. However, for most PC users, these window operations are not the primary purpose to use PCs. Therefore, it is desirable to reduce the window operations to the greatest extent possible.

Furthermore, we have conducted various experiments to compare the performance of reading (e.g., reading speed or error-detection rate in proofreading) when reading from paper and when reading from computer displays [8-10]. Results showed that reading from displays was inferior to reading from paper in cross-reference reading for multiple documents and reading with frequent moving between different pages[3]. And the difference of reading performance among media is caused by the difference of operability of documents such as arranging documents spatially or turning pages.

Window systems are used by almost all PC users irrespective of gender or nationality. Therefore, the improvement of the systems brings considerable value in total. To reduce window operation costs of digital documents, we propose an extended multi-window environment called the *Docking Window Framework* (*DWF*) [13]. This paper describes our approach, a prototype system, and its evaluation.

2. Approach.

2.1. Constructing workspaces.

Several systems have been proposed to support multitasking during PC work [14-21]. They enable users to create groups of windows called *workspaces* and to switch multiple windows easily and simultaneously. *Rooms* [14], a pioneer of systems of this type, is exemplary. It supports management of windows using a room metaphor. Each room corresponds to a workspace. Users can switch workspaces by moving virtual rooms. Other systems also support switching tasks, but they originally devised the mode of visualizing workspaces.

Nevertheless, previous systems have emphasized the support of task switching after creating workspaces. They have not supported creation of workspaces. To construct workspaces in previously proposed systems, users needed to determine what kind of workspace they should create, to specify which windows should be included in the workspace, and to organize the layout of windows in each workspace. In other words, users were required to formalize group structures of windows in advance for performing tasks. However, such formality often impedes the task performance [22].

[1]Hutchings and Stasko [6] analyzed window operation logs of 39 office workers. Results show that workers opened more than eight windows at the same time in most of their time when they operated PCs.

[2]According to the above Hutchings and Stasko's analysis [6], the average amount of time that any window was active was merely 20.9 seconds.

[3]These kinds of reading are frequently observed in a work situation [11,12].

Furthermore, workspaces are often restructured while performing tasks. For example, while performing tasks, users might realize that they must refer to other documents to perform tasks, or they might find that some documents are not necessary. Workspace construction is not a one-time procedure that occurs only at the initial setup of tasks. It occurs dynamically during task performance. Therefore, we consider that improvement of the user interface to construct workspaces can improve work efficiency meaningfully during computer use.

Considering these facts, we aim to support workspace creation. Followings are design requirements of our approach.

- We must allow creation of workspaces without declaring group relationships of windows. Window layout within a workspace cannot be determined automatically without users' specifications. Therefore, it would be advantageous for users to construct workspaces using a procedure of arranging windows.
- We must provide an easy intuitive user interface for grouping windows.
- Reconstruction of workspaces should also be performed easily.

To achieve this, we propose an interaction technique called *docking* that enables connecting of windows as if users were to plug in a piece of a jigsaw puzzle. After connecting windows, connected multiple windows can be activated or moved simultaneously. Therefore, they can behave as workspaces. In this framework, users need not construct group relations of windows. When users arrange windows, multiple windows are mutually connected automatically and workspaces are created to support multitasking.

As a user interface to arrange windows next to each other, the window-snapping technique, which is implemented in MagnetWindow and Virtual Desktop for Win32, is well known. In these systems, when two windows are placed closer together, those windows are magnetized and connected to contact each other. However, the magnetized windows cannot be used as workspaces because they cannot be operated simultaneously. The docking technique not only sets up adjacency relationships but also connects windows. The metaphor of a jigsaw puzzle helps users to realize that docked windows are strongly connected each other. We use this technique as a user interface to create workspaces.

We also provide a feature to save and restore workspaces. When constructing a workspace, users must execute multiple applications, arrange the window layout, and open all necessary documents. They must perform this procedure to set up a task environment every time they start up a PC. Furthermore, users might want to return to a previous state of a workspace so that they can recall the situation of the work or they can redo their work again. Our framework provides features to save the status of workspaces such as applications, the position of windows, and documents opened in each window, and to restore the status of workspaces merely using a single action.

2.2. **Reducing window operation costs.** The previous section presented a description of the approach to support creating workspaces of multiple tasks. This section provides an approach to reduce the cost of window operations within a single task. As we described before, window operation costs during working on PCs are not small. This is not a problem that can be resolved using a large screen space. In general, as the screen space becomes large, users tend to open many windows [6,7,23], which increases window overlapping. Consequently, window operation costs might increase because users frequently need to switch, move, and resize windows to refer to multiple sources simultaneously.

Many systems and techniques have been proposed to improve the operability of windows. They facilitate the selection and switching of windows [24-27], coordinate the window layout [28], and support the exchange of data between windows [29,30]. They are all solutions for problems that arise in situations in which the contents of windows

are partially invisible because of the window overlapping. If windows do not mutually overlap at all, these solutions are unnecessary. Moreover, some evidence suggests that a tiled window system is superior to an overlapping window system. In an experiment conducted by Bly and Rosenberg [31], users were able to extract information faster from multiple windows when using a tiled window system than when using an overlapping window system.

Considering these facts, we adopt an approach to layout windows as a tile and avoid overlapping of windows. Currently, most window systems allow window overlapping. However, overlapping window systems were developed originally to display many documents in a small display area. Computer displays have become larger and cheaper recently. On the assumption that many people will use large displays or multiple displays in the future, it is important to provide an environment that supports performance of tasks efficiently by making use of large screen space effectively in contrast to earlier approaches that make use of small screen space efficiently.

3. **System.**

We developed an extended window system called the *Docking Window Framework* (*DWF*) based on our approach of the previous section. This system has four main features: a docking user interface to construct workspaces, multiple window operations, tile layout to avoid overlapping, and saving and restoring workspaces. It runs on .NET Framework 2.0 and is implemented with C#.

3.1. **Docking user interface.**

Figure 1 presents the docking behavior. The top of the figure shows the situation before docking. When a user drags a window, if another window is near the dragged one, jagged hooks appear on edges of both windows, as shown in the top of the figure. These hooks show that the two windows can be mutually docked and that the edges with the hooks would be connected if the windows were docked. When the user drops the window in this state, docking of windows starts. The bottom of Figure 1 portrays the docking result. After docking, the dropped window position and size are changed so that the two connected windows constitute a single rectangle. The process of changing the layout of the dropped window is visualized as an animation so that the user can follow the change of the window layout.

Two windows connected by the docking operation are activated and moved simultaneously. They behave as if they were a single window. Therefore, we call the connected windows a *docking window*. After docking, in addition to the connected two windows, a new title bar, called a *docking bar*, appears at the top of the two windows. It provides commands to operate all windows of the workspace such as closing the workspace, minimizing and maximizing the workspace, releasing the connected windows of the workspace, saving the workspace, and changing the title and icon of the workspace.

Users can connect windows without limitations in the number of windows. They can also connect two docking windows. Figure 2 presents two examples of a window layout consisting of four windows. To construct the left layout of the figure, users might dock window 2 beneath window 1, dock window 3 beneath window 2, and finally dock window 4 at the right of the created docking window. To construct the right layout, users might dock window 2 beneath window 1, dock window 4 beneath window 3, and finally connect the two docking windows horizontally.

All docked windows can be detached using the release command of the docking bar. Detaching only a single window from a docking window is also possible. At that time the size of other remaining windows is coordinated so that they all constitute a single rectangle again. For example, when window 1 is detached in the left of Figure 2, window 2 expands vertically so that it covers the area of window 1. When window 4 is detached

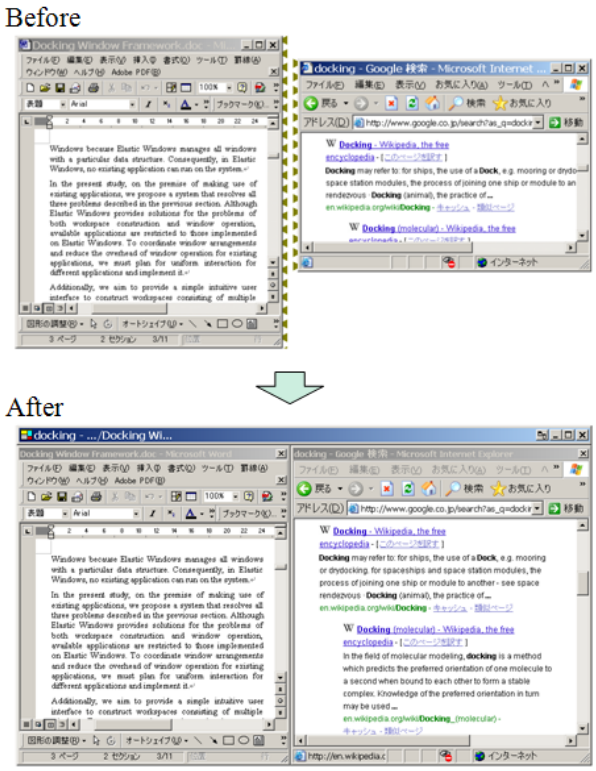

FIGURE 1. Before and after docking. The right window was docked to the left window.

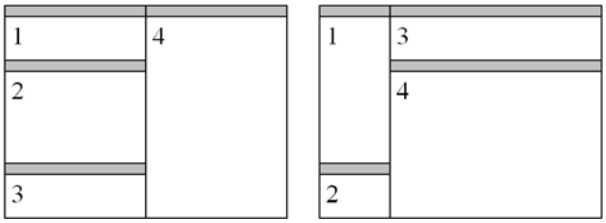

FIGURE 2. Examples of window layout in workspaces

in the left of Figure 2, windows 1, 2, and 3 all expand horizontally so that they cover the area of window 4.

Additionally, when docking is performed, two separate icons in the Windows task bar are integrated into a single one. This can prevent an overflow of icons in the task bar. The name of the icon, and therefore the title of the docking window, is created automatically by connecting each window title, but it can be modified later to facilitate discrimination among windows.

Users might want to put windows side by side sometimes without connecting windows. In such a case, if they drag windows while pressing the Shift key, then the jagged hooks do not appear and windows are not mutually connected.

A docking window consisting of multiple windows visually resembles a single window. In addition, users can activate and move all windows of a docking window simultaneously. A docking window can behave like a single window as a whole. Therefore, it can be used as workspaces to support multitasking because users can switch tasks easily by changing active docking windows.

In all previous systems introduced in the previous section, before performing tasks, users must determine what kind of workspace they create. They must also specify which

windows should be included in the workspace. In contrast, we provide a framework in which windows behave as a single window when multiple windows are docked and in which the connected windows are useful as a workspace. In this framework, users can construct a workspace through the action of juxtaposing windows without declaring a group structure of windows. In other words, when a user allocates windows side by side, then these windows are mutually connected and the workspace is created automatically without selecting any command to create a workspace and without arranging a window layout.

3.2. **Multiple window operation.** People often organize paper documents as piles and move them simultaneously. Similarly, users might want to operate on multiple documents within a workspace simultaneously. Not only do such operations reduce the window operation load; they also help users to realize a workspace consisting of multiple documents as a single window. DWF supports the following simultaneous operations for connected multiple windows.

Activating When users click any area of a workspace or the icon of the workspace in the task bar of Windows OS, all windows of the workspace are activated simultaneously. Using this feature, users can switch tasks easily with one click.

Moving When users drag any window or a docking bar of a workspace, all windows within the workspace are moved simultaneously. Using this feature, users can arrange and organize workspaces easily.

Resizing DWF always maintains the workspace shape as a rectangle, which helps users to realize a workspace as a single window. It also enables the effective use of the display area. When users change the size of any window, the sizes of other windows are changed to maintain the rectangular shape. Figure 3 shows a situation where the resizing of window B leads to the resizing of window A.

Enlarging/Narrowing DWF provides a feature to display a specified window as large (or small) as possible with all remaining window contents visible. This feature is called *enlarging* (or *narrowing*). It differs from maximizing (or minimizing) of traditional window systems in that maximized windows cover the entire display area such that users cannot view any contents of other windows (or users cannot view any content of minimized windows). Although maximizing and minimizing are features to use a small display space efficiently, enlarging and narrowing are features to use large display space effectively. When users enlarge a window, other windows change size, as shown in Figure 4. The process of layout change resulting from enlarging and narrowing can be visualized as an animation so that the user can follow the change of the window layout.

To provide uniform interaction for windows of different applications, the Task Gallery [17] provides a solution that pops up original command buttons. Our system uses this solution. When a mouse cursor is on a title bar of any window, the system pops up a *floating bar*, as shown in Figure 5. It provides commands for the individual window positioned below the floating bar. The system currently provides the following commands:

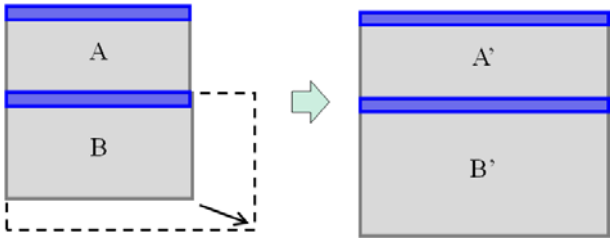

FIGURE 3. Resizing windows

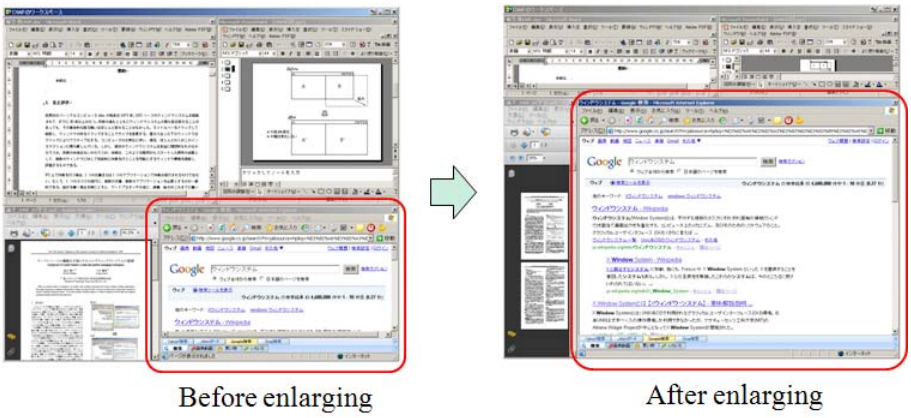

FIGURE 4. Before and after enlarging

FIGURE 5. Floating bar

detaching the window from the workspace, enlarging and narrowing the window, and closing the window.

3.3. **Tile layout.** In DWF, we use a tiling window approach that eliminates window overlapping. This approach sustains a tiling layout at all times. Even if users change the size of a certain window, the window layout remains tiled, which is to say that the change of the window size affects other windows. The total window layout is coordinated to avoid overlapping. The mode of window coordination is fundamentally the same as that of Elastic Windows [15] except for small differences (see [13] for details).

3.4. **Saving and restoring workspaces.** DWF provides a feature to save the state of workspaces (title, icon, applications, position and size of windows, and documents) to a file and to restore the previous state of workspaces later. Using this feature, users need not reconstruct workspaces from scratch whenever a PC is started up. We describe implementation of this feature later.

4. **Evaluation.** Using DWF, we expect the following effects:

A. users can arrange windows and construct workspaces easily using a docking user interface,

B. users can switch tasks easily by switching workspaces, and

C. users can perform tasks efficiently using the tile layout of windows and the feature of enlarging and narrowing.

To confirm the effect A, we conducted an experiment using a window arrangement task (Experiment 1). To confirm the effects B and C, we conducted an experiment for a task-switching task (Experiment 2).

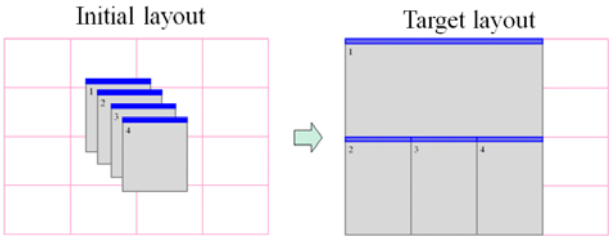

FIGURE 6. Initial layout and target layout in the window arrangement task

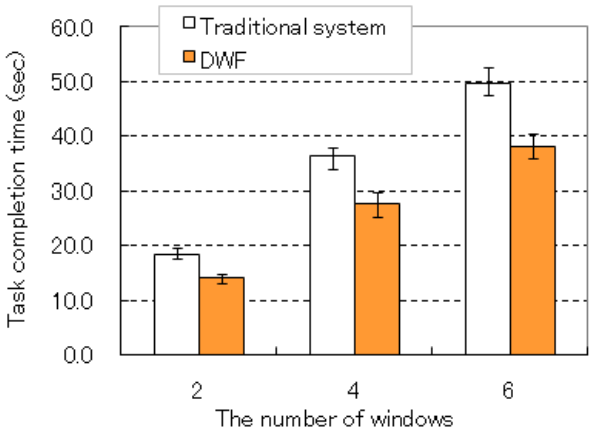

FIGURE 7. Task completion time in window arrangement tasks

4.1. **Experiment 1: Window arrangement.** To construct workspaces in DWF, users must connect windows and adjust the window layout. Our hypothesis is that DWF yields faster performance than traditional window systems for a window arrangement because it provides an easy docking user interface and multiple window operations after docking. We also hypothesize that the performance difference between two systems is remarkable when using many windows.

(1) Method. The experimental design was a 2×3 within-participants design. The first factor was the system condition (a traditional window system and DWF). The second factor was the number of windows (2, 4, and 6 windows). Each participant performed all conditions of tasks and performed two trials in each condition. The order of the system conditions and the number of windows in the series of participants' trials were counterbalanced to cancel the overall effect of the trial order.

Participants were 12 people (6 men, 6 women). Their ages were 21-38 years (avg. 28.0). Each had three or more years' experience using a PC. The vision of each, after correction, was better than 0.7.

The PC used in the experiment (Dimension C512; Dell Inc.) was connected to a 23-inch TFT display (FlexScan; Eizo Nanao Corp.). The OS was Windows XP.

The experiment task was to arrange windows. The left of Figure 6 shows the initial window layout: a cascade layout. We presented the target window layout as shown in the right of Figure 6, and requested that participants allocate windows to match the target layout. We drew a four-by-four lattice in pink on the desktop and required allocation of the specified number of windows to the specified position.

(2) Results and discussion. Figure 7 presents the task completion times. The error bar shows plus or minus one standard error from the average. Two-way repeated measures analysis of variance was conducted to assess the task completion time. Results

show that the main effects of the system condition [$F(1, 11) = 26.3$, $p < .001$] and the number of windows [$F(2, 22) = 144.5$, $p < .001$] were significant. Interaction of the two factors was significant [$F(2, 22) = 3.4$, $p < .05$]. Then we tested the simple main effects for each number of windows. They were all significant [for 2, 4, an 6 windows, $F(1, 11) = 17.2$, $p < .01$; $F(1, 11) = 21.4$, $p < .001$; $F(1, 11) = 12.4$, $p < .01$, respectively]. In DWF, participants arranged windows 22.9%, 23.4% and 23.4% more quickly than when using the traditional system for 2, 4 and 6 windows, respectively.

We compared the accuracy of window layouts among the system conditions. The distance between two windows was defined as the sum of the four Euclidean distances between corresponding vertices of the windows. Furthermore, we defined the window layout accuracy as the distance between the target layout and the actually allocated layout. Two-way repeated measures analysis of variance was conducted to assess the accuracy of the window layout. Although the main effect of the number of windows was significant [$F(2, 22) = 59.8$, $p < .001$], the main effect of the system condition was not significant [$F(1, 11) = 0.4$, $p > .1$].

According to the results, participants arranged windows with DWF 20.9-23.4% faster than when using the traditional window system. No difference in the accuracy of the window layout was found between DWF and the traditional window system. Results show that users of DWF can arrange windows rapidly without sacrificing accuracy.

A noteworthy point in this experimental task is that DWF users not only arranged windows but also created workspaces by connecting windows. In DWF, users adjust the window layout after creating a workspace, but they can still arrange windows more than 20% faster than when using the traditional window system. We consider the following two points as reasons for this faster performance.

First, the operation of window allocation doubles as the operation of workspace construction. Therefore, no cost accrues to construction of the workspace, where we can consider that the cost to construct a workspace is included in the cost to allocate windows.

Second, in DWF, users can operate multiple windows simultaneously after the construction of workspaces, which engenders rapid window organization. Furthermore, because DWF creates a tile layout automatically, users need not devote attention to trivial layout adjustment of windows after they merely set up a rough window layout.

In this experiment, the target layouts were all tiled. For arranging windows in layouts of other types such as cascading layouts, DWF requires users to continue to press the Shift key to avoid docking windows unintentionally. Although that requirement might bother users, our framework is designed on the basis that the tiling approach is superior to an overlapping approach when used for a large display space.

We are also interested in the comprehensibility of systems because DWF provides an unusual user interface in terms of the docking and the tile layout. In the post-task interview, one participant described that he "became accustomed to DWF right away after actual use of it". Some participants reported that it "felt fun to use DWF" and that they "felt a pleasurable sensation when docking windows". These comments suggest that the framework of window docking is easy to learn; it also provides a pleasurable experience for users sometimes.

4.2. **Experiment 2: Task switching and performing tasks.** Although the most characteristic point of DWF is in the mode of workspace construction, the aim of constructing workspaces is to support task switching. The second experiment takes up a multitasking situation and compares the user performance between DWF and a traditional window system.

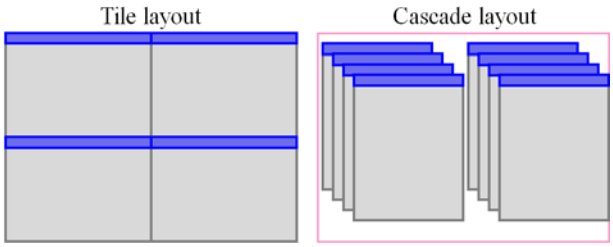

FIGURE 8. Initial window layout in task-switching tasks

Multitasking requires multiple documents. Therefore, the initial layout of windows is an important factor affecting the user performance. In the traditional window system, we set two initial layouts: the tile layout and the cascade layout. Furthermore, observations revealed that physical paper effectively supported the reading of multiple documents [12,32,33]. We are interested in whether or not DWF is superior to physical paper when performing multitasking. Our hypothesis is that DWF yields faster performance than the conditions of the traditional window systems, but it does not reach the level of physical paper.

(1) Method. The experimental design was a one-way within-participants design. We set the following four conditions as factors: using paper documents (Paper), using our system (DWF), using a traditional window system in which the initial window layout was a tile layout, as shown at the left of Figure 8 (Tile), and using a traditional window system in which the initial window layout was a cascade layout, as shown at the right of Figure 8 (Cascade).

Participants were 18 people (9 men, 9 women). Their ages were 21-39 years (avg. 29.5). Each had three or more years' experience of using a PC. The vision of each, after correction, was better than 0.7.

Each participant performed two trials in each condition. The order of conditions and document sets in the series of participants' task trials was counterbalanced to cancel the overall effect of the order.

The display and the PC used in this experiment were the same as those described for Experiment 1.

In this experiment, participants performed two independent tasks in parallel, where each task is to count products that match the specified conditions.

We created documents used in the experiment related to product specifications such as digital cameras, printers, and TFT displays. For each task, four one-page documents were used. Three of the four were product lists. The remaining one was a question list for which participants were asked to count products matching the condition in the product lists. We created eight document sets consisting of these four documents for the experiment and two document sets for exercises.

In the product list, each item was created using product names, manufacturers, weights, prices, sales rankings and other attributes of products. Each page of a product list consisted of 16 products; all products in each counting task included 48 items in total. Each document set included six questions. Samples of questions are shown for TFT displays in the following.

- How many displays for which the manufacturer is ACER are there?
- How many displays for which the power consumption is less than 20 W are there?

The experimental task was to perform two counting tasks in parallel while switching them. Figure 9 shows that participants answered the initial three questions of the first

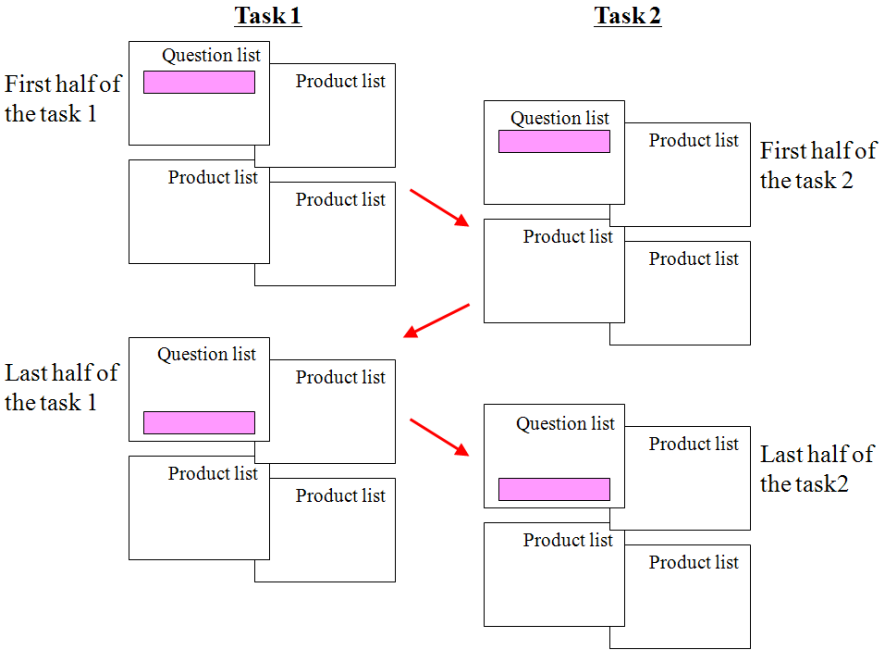

FIGURE 9. Order of performing task-switching tasks

task and then answered the initial three questions of the second task. Next, they answered the last three questions of the first task and then answered the last three questions of the second task. This procedure was written in question lists to prevent mistakes in the order of answering questions.

In each task, participants switched between the two counting tasks three times. They were asked to perform this procedure as quickly and as accurately as possible.

In the paper condition, electronic documents were printed on one side of A5 paper in black and white. Electronic documents were all given in PDF and displayed with Adobe Reader 9. We adjusted the character size of electronic documents to be the same size as those of paper documents. We prohibited changing of the display character size.

In all conditions, we delivered answer sheets of question lists printed in paper and requested that participants write down answers on the sheets. We required that they refer to the answer sheets only when they wrote down the answers and they referred to electronic and paper documents for performing tasks. Participants wrote down answers on additional answer sheets to unify the mode of writing between paper and electronic conditions, thereby excluding the effect of the mode of writing. In electronic conditions, participants need to change the device from a mouse to a pen to write answers. Therefore, we prohibited participants from performing tasks while holding a pen when counting products in the paper condition.

In the paper condition, the initial documents were provided as two piles corresponding to two counting tasks. The initial layout of the DWF condition was two workspaces, where each consisted of four tiled windows, as shown on the left of Figure 8. The initial layout of the tile condition was the same as that in the DWF condition, except that all windows were independent and unconnected. The right half of Figure 8 shows that the initial layout of the cascade condition was two stacks of cascading windows. In all electronic conditions, participants were allowed to change the size and position of windows when performing tasks.

(2) Results and discussion. Figure 10 presents the task completion times in each condition. The error bar shows plus or minus one standard error from the average.

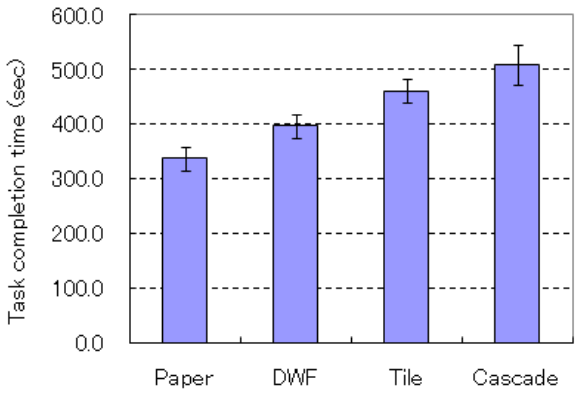

FIGURE 10. Task completion time in task-switching tasks

Repeated measures analysis of variance was conducted to assess the task completion time. Results show that the main effect of conditions was significant [$F(3, 51) = 20.35, p < .001$]. According to multiple comparison using the LSD method, the task completion time in the paper condition was significantly shorter than in the DWF condition [$p < .01$]. The task completion time in the DWF was also significantly shorter than in the tile condition [$p < .01$]. A tendency was also apparent by which the task completion time in the tile condition was shorter than that in the cascade condition [$p < .1$]. In the paper condition, the tasks were performed 14.9% faster than in the DWF condition, 26.8% faster than in the tile condition, and 33.6% faster than in the cascade condition. Additionally, regarding DWF, the tasks were performed 13.9% faster than in the tile condition and 21.9% faster than in the cascade condition.

We compared the accuracy of product counting. The percentage of the correct answers was highest in the cascade condition (79.9%) and lowest in the tile condition (71.5%). However, the difference was not significant [$p > .1$].

We provide the following three suggestions based on the results. First, using DWF, the participants were able to perform multitasking more efficiently than with traditional window systems. According to the participants' report, they felt that, using DWF, they were able to switch tasks quickly, understand what tasks they engage in, and grasp an overview of all tasks. With DWF, users can switch workspaces with one click and can view whole contents of the document by enlarging and narrowing them. The use of these features was observed frequently while performing tasks. We consider that these features engender efficient task performance.

Second, even if users used the same window system, the task completion time differed depending on the initial window layout. Users were able to perform multitasking more efficiently when using the tile layout than when using the cascade layout, which demonstrates that the tile layout is superior to the cascade layout in multitasking when opening many windows. This result underscores the validity of the DWF approach, which avoids window overlapping.

Third, the use of paper enables performance of multitasking far more efficiently than electronic environments, which demonstrates that physical paper is an excellent tool to support multitasking. DWF is an extended window system that provides many features that physical paper cannot provide, such as tile layout, multi-window activation, and enlarging. However, paper remains superior to DWF, which indicates that we can learn a methodology to support multitasking by analyzing human interaction with paper.

5. **Implementation.** In the DWF implementation, a resident application called a *DWF manager* monitors the behavior of all windows and controls the positions and sizes of windows.

To save and restore the states of workspaces, including files opened in each window, and to make the task state persistent, the DWF manager must ascertain which file is opened in each window. However, in the current Microsoft Windows architecture, no general solution for this issue functions for all application [23]. Therefore, DWF takes an approach by which each application running on DWF sends messages when opening files and closing files to the DWF manager. Existing applications can be run on DWF by being added as plug-ins to send messages.

We briefly describe the development cost of applications running on DWF, which we call *DWF clients*. When we implement a new DWF client, programmers merely add one line of source code to specify the use of a DWF library. To make an existing application run on DWF, a user must implement a plug-in for the application. However, the logic of plug-ins is simple. We can easily implement plug-ins with a small quantity of source code (about 50 lines). We have implemented plug-ins for MS Office and Internet Explorer, and have confirmed their reliable operations.

However, this architecture is one example implemented within the restriction of Windows OS. We do not regard it as ideal. We believe that our framework should be implemented as one module of an OS from the perspective of reliability and processing speed.

6. **Conclusions.** We proposed a framework to reduce window operation cost on PC work called the Docking Window Framework (DWF). The most characteristic point of the system is the means of constructing workspaces by connecting windows through a simple intuitive user interface called docking. In this framework, users can construct a workspace through the action of juxtaposing windows without declaring a group structure of windows.

We verified the effectiveness of our system using two experiments. The first experiment of window arrangement tasks revealed that DWF enabled setting up of a window layout more than 20% more quickly than when using a traditional window system. The second experiment of performing multitasking revealed that users were able to perform multiple tasks in parallel more than 10% faster when using DWF than when using a traditional window system.

After the experiments, not a few participants expressed their strong desire to use our system in their daily activities. We are currently preparing the use of DWF in real-world settings. Additionally, our system can be extended so that connected windows exchange data and change behavior according to the status change of other windows. We would like to examine the methodology of such coordination among windows.

REFERENCES

[1] B. O'Conaill and D. Frohlich, Timespace in the workplace: Dealing with interruptions, *Proc. of CHI'95*, pp.262-263, 1995.

[2] J. M. Hudson, J. Christensen, W. A. Kellogg and T. Erickson, I'd be overwhelmed, but it's just one more thing to do: Availability and interruption in research management, *Proc. of CHI'02*, pp.97-104, 2002.
[3] M. Czerwinski, E. Horvitz and S. Wilhite, A diary study of task switching and interruptions, *Proc. of CHI'04*, pp.175-182, 2004.
[4] V. Gonzalez and G. Mark, Constant, constant, multi-tasking craziness: Managing multiple working spheres, *Proc. of CHI'04*, pp.26-29, 2004.
[5] G. Mark, V. Gonzalez and J. Harris, No task left behind? Examining the nature of fragmented work, *Proc. of CHI'05*, pp.321-330, 2005.
[6] D. R. Hutchings and J. Stasko, Revisiting display space management: Understanding current practice to inform next-generation design, *Proc. of Graphics Interfaces'04*, 2004.
[7] H. Shibata, Operational efficiency of single and multiple display systems in an actual work environment, *Proc. of IDW'11*, 2011.
[8] H. Shibata and K. Omura, Effects of paper in moving and arranging documents: A comparison between paper and electronic media in cross-reference reading for multiple documents, *The Journal of Human Interface Society*, vol.12, no.3, pp.301-311, 2010 (in Japanese).
[9] H. Shibata and K. Omura, Comparison between paper and electronic media in reading documents with going back and forth through pages, *The Journal of the Human Interface Society*, vol.13, no.4, 2011 (in Japanese).
[10] K. Takano, H. Shibata and K. Omura, Microscopic analysis of document handling while cross-reference reading for multiple documents, *The Journal of the Human Interface Society*, vol.14, no.4, 2012 (in Japanese).
[11] A. Adler, A. Gujar, B. Harrison, K. O'Hara and A. J. Sellen, A diary study of work-related reading: Design implications for digital reading devices, *Proc. of CHI'98*, pp.241-248, 1998.
[12] A. J. Sellen and R. H. Harper, *The Myth of the Paperless Office*, The MIT Press, 2001.
[13] H. Shibata and K. Omura, Docking window framework: Supporting multitasking by docking windows, *Proc. of APCHI'12*, 2012.
[14] J. A. Henderson and S. K. Card, Rooms: The use of multiple virtual workspaces to reduce space contention in a window-based graphical user interface, *ACM Transactions on Graphics*, vol.5, no.3, pp.211-241, 1986.
[15] E. Kandogan and B. Shneiderman, Elastic windows: Improved spatial layout and rapid multiple window operations, *Proc. of AVI'96*, pp.29-38, 1996.
[16] E. Kandogan and B. Shneiderman, Elastic windows: Evaluation of multi-window operations, *Proc. of CHI'97*, pp.250-257, 1997.
[17] G. Robertson, M. Dantzich, D. Robbins, M. Czerwinski, K. Hinckley, K. Risden, D. Thiel and V. Gorokhovsky, The task gallery: A 3D window manager, *Proc. of CHI'00*, pp.494-501, 2000.
[18] B. MacIntyre, E. Mynatt, S. Voida, K. Hansen, J. Tullio and G. Corso, Support for multitasking and background awareness using interactive peripheral displays, *Proc. of UIST'01*, pp.41-50, 2001.
[19] G. Smith, P. Baudisch, G. Robertson, M. Czerwinski, B. Meyers, D. Robbins, E. Horvitz and D. Andrews, GroupBar: The TaskBar evolved, *Proc. of OZCHI'03*, 2003.
[20] G. Robertson, E. Horvitz, M. Czerwinski, P. Baudisch, D. Hutchings, B. Meyers, D. Robins and G. Smith, Scalable fabric: Flexible task management, *Proc. of AVI'04*, pp.85-89, 2004.
[21] T. Matthews, M. Czerwinski, G. Robertson and D. Tan, Clipping lists and change borders: Improving multitasking efficiency with peripheral information design, *Proc. of CHI'06*, pp.989-998, 2006.
[22] F. M. Shipman and C. C. Marshall, Formality considered harmful: Experiences, emerging themes, and directions on the use of formal representations in interactive systems, *Proc. of CSCW'99*, pp.333-352, 1999.
[23] G. Robertson, M. Czerwinski, P. Baudisch, B. Meyers, D. Robbins, G. Smith and D. Tan, The large-display user experience, *IEEE Computer Graphics and Applications*, vol.25, no.4, pp.44-51, 2005.
[24] M. Beaudouin-Lafon, Novel interaction techniques for overlapping windows, *Proc. of UIST'01*, pp.503-512, 2001.
[25] G. Faure, O. Chapuis and N. Roussel, Power tools for copying and moving: Useful stuff for your desktop, *Proc. of CHI'09*, pp.1675-1678, 2009.
[26] C. Tashman, WindowScape: A task oriented window manager, *Proc. of UIST'06*, pp.77-80, 2006.
[27] Q. Xu and G. Casiez, Push-and-pull switching: Window switching based on window overlapping, *Proc. of CHI'10*, pp.1335-1338, 2010.

[28] G. J. Badros, J. Nichols and A. Borning, SCWM: An intelligent constraint-enabled window manager, *Proc. of AAAI Spring Symposium on Smart Graphics*, 2000.
[29] P. Dragicevic, Combining crossing-based and paper-based interaction paradigms for dragging and dropping between overlapping windows, *Proc. of UIST'04*, pp.193-196, 2004.
[30] O. Chapuis and N. Roussel, Copy-and-paste between overlapping windows, *Proc. of CHI'07*, pp.201-210, 2007.
[31] B. Bly and J. Rosenberg, A comparison of tiled and overlapping windows, *Proc. of CHI'86*, pp.101-106, 1986.
[32] K. O'Hara and A. Sellen, A comparison of reading paper and on-line documents, *Proc. of CHI'97*, pp.335-342, 1997.
[33] K. P. O'Hara, A. Taylor, W. Newman and A. J. Sellen, Understanding the materiality of writing from multiple sources, *International Journal of Human-Computer Studies*, vol.56, no.4, pp.269-305, 2002.