Window Brokers: Collaborative Display Space Control

RICHARD ARTHUR and DAN R. OLSEN, JR., Brigham Young University

As users travel from place to place, they can encounter display servers, that is, machines which supply a collaborative content-sharing environment. Users need a way to control how content is arranged on these display spaces. The software for controlling these display spaces should be consistent from display server to display server. However, display servers could be controlled by institutions which may not allow for the control software to be installed. This article introduces the window broker protocol which allows users to carry familiar control techniques on portable personal devices and use the control technique on any display server without installing the control software on the display server. This article also discusses how the window broker protocol mitigates some security risks that arise from potentially malicious display servers.

Categories and Subject Descriptors: H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*User interface management systems (UIMS)*

General Terms: Design, Algorithms, Performance, Human Factors, Security

Additional Key Words and Phrases: Wireless sensor networks, media access control, multi-channel, radio interference, time synchronization

ACM Reference Format:
Arthur, R. and Olsen, Jr. D. R. 2012. Window brokers: Collaborative display space control. ACM Trans. Comput.-Hum. Interact. 19, 3, Article 17 (October 2012), 21 pages.
DOI = 10.1145/2362364.2362365 http://doi.acm.org/10.1145/2362364.2362365

1. INTRODUCTION

People are nomadic and need to collaborate. Modern collaboration regularly involves discussing data. Portable computers are often a source of that data, because a user can guarantee that her data, software, and settings are always available wherever she is located.

An increasingly common form of collaboration is via a large, annexable, shared display space. When a user annexes a screen, she may then share individual windows to that screen so that others in the room may view and discuss the contents of those windows. To annex the screens and share content in the most flexible way requires a network UI distribution framework. Such a framework necessitates a network-connected dedicated computer called a *display server*. Ideally, these display servers (regardless of manufacturer) would use a consistent annexation protocol so that any *client machine* may annex them.

Figure 1 illustrates a situation in which three users wirelessly connect to a display server that controls a single display space. Each user is sharing at least one window on that space.

Problems arise in how the display server coordinates shared windows. For instance, should the person in the middle be allowed to enlarge his window and overlap one or

Authors' addresses: R. Arthur, Microsoft Corporation; email: startether@startether.com; D. R. Olsen, Jr., Computer Science Department, Brigham Young University.

DOI 10.1145/2362364.2362365 http://doi.acm.org/10.1145/2362364.2362365

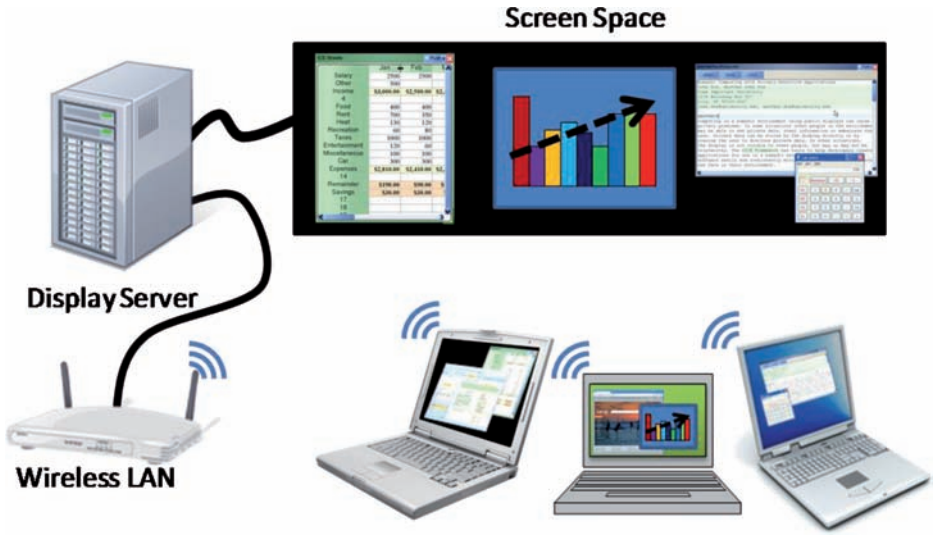


Fig. 1. Three users bring their laptops and annex a display space to share and discuss data. Via software controls on their laptops, users may rearrange the remote windows.

both of the other users' windows? The answer to this question depends on the situation. The answer may be 'yes' if the users are the only ones in the room, but 'no' when there are other people in the room who would have difficulty tracking the discussion if the windows overlapped. Alternatively, the answer may be 'no' if the users are dividing the display space, but 'yes' if the users are comparing windows.

Users could rely on purely social coordination (e.g., saying "Please don't do that") to take care of conflicts, which may work well for a close-knit group of people. Unfortunately, the social coordination approach is slower and is prone to abuse. Some users need an automated approach to controlling a display space.

Users also need consistent control. The display space may be located in a room controlled by a trusted institution (e.g., the employer), a room controlled by a less-trusted institution (e.g., a conference hall), or a room controlled by an untrusted institution (e.g., a competitor's conference room). In addition, coordinating logins and control before interacting with a display space is counterproductive to quickly establishing a collaborative group. Consequently, this control may need to be exhibited anonymously. Users need to be able to quickly and smoothly exhibit control on display spaces they may encounter only once and which they may not trust.

This article is about how to manage and flexibly enforce all the different control paradigms a user might need in a variety of collaborative environments.

1.1. Motivating Examples

There are an infinite number of collaborative situations users could encounter. For the discussions in this article, the following four example situations are helpful in understanding these issues: presenter, discussion, panel, and visitor. This article uses these situations to illustrate six different *control techniques* for managing the display space.

In the *presenter* situation, someone gives a presentation to her teammates. This person is in control of the presentation software and uses a control technique which shows only her windows. When she opens the discussion to her teammates, one team member has some relevant data he would like to share. The presenter then opts to

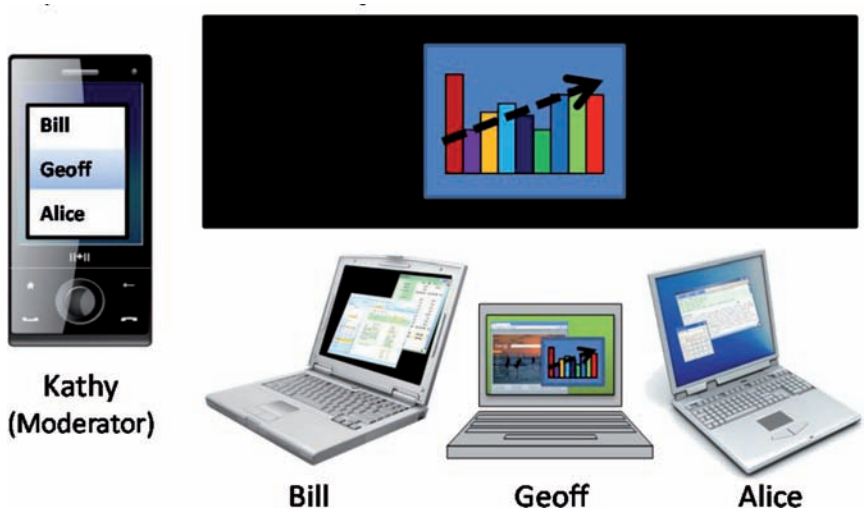


Fig. 2. Kathy, the moderator, chooses Geoff to have floor control, so the windows from Bill and Alice are hidden.

accept and show her teammate's shared window on the display space. In this situation, the presenter is in control and no one else may show content without the presenter's permission. This situation may also occur in a classroom or conference presentation.

The *discussion* situation consists of a group of people working on a common project. Depending on the group members and the kind of discussion, there are multiple ways to digitally assist the discussion. We have chosen to discuss the following four control techniques (a subset of the possible options): free-form, moderated control, personal space, and tiled. The *free-form* technique is when window placement is unconstrained. Anyone may place any window anywhere. The *moderated control* technique is when one user is delegated as the "moderator" and she is the only person who may rearrange windows on the display space. The *personal space* technique is one where the display space is partitioned so that each participant has her own section of the display. Windows may reside only within the partition assigned to the window's owner. The *tiled* technique is one where windows may be placed anywhere on the display space, but if the window overlaps any other windows, the overlapped windows are moved out of the way or shrunk in size so that no two windows overlap. The tiled technique manages the display space similarly to the Flatland [Mynatt et al. 1999] whiteboard management tool—if one window's movement would overlap another window, the other window is pushed away or, if there is insufficient room to move, shrunk or hidden.

In the *panel* situation, a panel discussion is held at a conference. Several guests are invited to be on the panel, and one person is the moderator. The discussion moderator annexes the screen via her phone. Her phone has software installed which she can use to choose which of the guests has the floor. The *floor control* technique will only show windows owned by the guest who has the floor. Windows from other guests are hidden. This result is illustrated in Figure 2. Assume Alice is currently granted floor control. When the moderator, Kathy, changes the floor control to Geoff, then all of his windows are shown, and all of Alice's windows are hidden. Geoff may then show any content he chooses using whatever software he owns.

Suppose Kathy is hosting other panels at other institutions as part of an ongoing project. Rather than learning each institution's moderating software, in this visitor situation, she uses her personal moderating software at all the institutions.

These four situations illustrate six different control techniques: presenter, free-form, moderated control, personal space, tiled, and floor. A single display space may be called upon to support all of these different techniques and any others which users deem necessary. The display space should support techniques that are not known to the display space but are familiar to its users.

1.2. Solution Requirements

There are two paradigms for how control techniques are provided to users: via the display server or via a personal device. For instance, in the panel situation of Figure 2, Kathy used the floor control technique installed on her phone. But the display server could house the control technique instead. In this case, Kathy would use a computer embedded in the room's podium to select which of the presenters has the floor. While embedding the software in her phone allows Kathy to carry a familiar application with her wherever she goes, embedding the software in the display server ensures that Kathy does not need to carry anything.

If the display server supplies the control techniques, a user can only employ server-installed control techniques. If the user is previously unaware of an appropriate control technique, then the user must find and learn a new technique on the spot. If the user is already familiar with a technique that she wishes to apply, then she can encounter frustrations in foreign rooms. Consider Kathy transitioning from the panel situation to the visitor situation. She is accustomed to a specific piece of software at her home institution and must now moderate a discussion at a foreign institution.

In a foreign room, Kathy must first find the floor control technique. If the technique is available, she must recognize and choose it from the list of (potentially many) items. However, if the floor control technique is unavailable, then Kathy has three options: use an alternate technique, install the technique, or forego the technique. Kathy may choose a technique that she hopes is similar enough and possibly learn that technique on the fly, which is likely to be frustrating. She may install the technique she wants, but because most institutions keep careful control over what may be installed, installing new software will lead to interacting with the support (information technology or IT) staff. Interacting with the IT staff requires her to either install the software on the fly or ahead of time, which consumes time and energy. Finally, Kathy may forego using the floor control technique, because that choice is easier than the prior paths, but it requires her to spend more time giving verbal commands to the guests.

Because there are a potentially infinite number of control techniques that could be developed, it is unreasonable to expect the display server to have every possible version pre-installed. In addition, installing the software on the display server also increases the efforts of the maintenance staff in keeping the display server up-to-date and introduces users to potential versioning conflicts. For instance, a user may inadvertently choose an older version of a familiar control technique and then become frustrated with the bugs present or lack of features. Conversely, a newer version of the technique may have unfamiliar features or options that get in the way when the user is trying to accomplish some task. Consequently, the display space control techniques should not be built into the display server.

Users should be able to bring their own control techniques and apply them to any room, whether or not that technique was previously applied to that room. If Kathy brings her own familiar floor control technique, then she can apply it to any foreign room. This requires carrying a portable computer, but if that computer is her phone, then this barrier is low. Now she does not need to learn a new technique, interact with the IT staff, or deal with versioning conflicts.

In summary, a solution to the illustrated collaborative situations must meet at least the following points.

—*Automated display control* that supports automated control techniques (i.e., display space management techniques).
—*Plugin-free display control* that supports new control techniques without installing software on the display server.
—*Familiar display control* so that users do not need to learn new software to utilize a known control technique.

1.3. Proposed Solution

This article introduces the *window broker* protocol which provides automated, plugin-free, familiar, display control. Display servers do not have any control techniques installed but instead allow portable devices to supply a technique via network procedure calls to the user's personal device. Now a limitless variety of display space control techniques may be applied to any given display server.

To implement the window broker protocol, a display server designates one machine as a *broker*. For instance, in Figure 2, Kathy's machine is set as the broker. When Bill attempts to move the shown window, that attempt is forwarded to Kathy's software which denies that movement because Bill does not have floor control. However, if Geoff moves the window, the attempt is also forwarded to Kathy's software which approves it because Geoff has floor control. The display server implements the results of each forwarded call.

With the window broker protocol, users may carry their own display management software anywhere they travel. Displays may now transparently implement new control techniques without installing new software because the decisions underlying the techniques are made by the broker on a client machine.

1.4. Prior Work

There are several collaborative screen management technologies that have been developed for various installations, each with varying control techniques.

Some systems implement a free-form control technique and use the display space's computer mouse as the arbiter. The most basic systems that implement this are X-Windows (X11) [Scheifler and Gettys 1986], Remote Desktop Protocol (RDP) [Tritsch 2003], or Virtual Network Computing (VNC) [Richardson et al. 1998]. These machines place control over the placement of windows in the display server, and users must have physical control of the display server's keyboard and mouse to rearrange the windows. Other systems that take a similar approach include IMPROMPTU [Biehl et al. 2008], WinCuts [Tan et al. 2004], and Lacome [Liu 2007] which are designed for groups of individuals who may periodically want to share and discuss information via a shared display space. In such collaborative situations, a user is unlikely to go up to the shared display and grab the input hardware just to exert control; he would rather exert control from his own device using software he is familiar with.

Other systems include control techniques that facilitate specific cases. WeSpace [Jiang et al. 2008] provides an API that can be used to implement control techniques which are installed on the display server. One such control technique is implemented by LivOlay [Jiang et al. 2008]. With WeSpace, someone could implement the tiled technique in which no two windows are allowed to overlap. Unfortunately, the control technique must be installed on the display server, so people can only use a technique if it is installed on the server. This rigidity prevents users from portably using software such as LivOlay or the tiled technique wherever they need it.

Collaborative single-display software will often have case-specific control software. For instance, the Dynamo [Izadi et al. 2008] interactive display environment implements a sophisticated version of the personal space technique. The display space has several computer mice installed, and a user may use one mouse to carve out a personal

workspace from an available portion of the display space. However, interaction at the display space is limited by the number of computer mice installed and is the only control mechanism available. Limiting the number of users limits the collaboration possible. Although Dynamo provides a nice technique for managing multiple users, the implementation is rigid and cannot support the other control techniques discussed.

iRoom [Johanson et al. 2002a] with PointRight [Johanson et al. 2002b] provides an interactive multi-user environment but does not provide a display control system. PointRight is built on top of iRoom and controls what hardware device may provide input to an individual display. Input from any device in an iRoom setup may be directed to any display, but only a single set of input (keyboard and mouse) may be directed to a display at any one time. Fundamentally, this approach controls a single display at a time rather than the entire display space. For example, if one user is interacting with a window on one display, then all other windows on that display are inaccessible to other users. Consider also a user who is giving a presentation across multiple displays: that user may control one display at a time but cannot control all of the displays.

A new, flexible, display-space control mechanism is needed which allows for a wide variety of interactive collaboration techniques. This new control mechanism must support automated, plugin-free, familiar display space control so that institutions have lower maintenance costs and users have familiar controls in any interactive room.

1.5. Window Broker

The window broker protocol is a display space management protocol which provides automated, plugin-free, familiar display space control. The window broker protocol has a simple authorization mechanism which gives a single user's machine the power to control if and where windows are created and positioned on the annexed display space.

The window broker protocol is implemented as part of a screen sharing protocol called SPICE (SPaces for Interactive Computing in Education), but the techniques could be incorporated into other UI distribution protocols, such as X11, RDP, or VNC. The SPICE, and window broker protocols are built primarily in Java [Gosling et al. 2000] (version 1.6), and to ensure broader compatibility, a C# [Thai and Lam 2002] version of the core protocols also exists.

The window broker protocol design stems from an attempt to control changes which affect the overall display space. For instance, an instructor may need to blank the screen to gain access to a whiteboard or show the contents of the screen again. also, She also may need to change the volume of a rendered video or mute the screen altogether. Other changes that need to be controlled involve the arrangement of windows on the screen: create, move, show, hide, destroy, or change z-order. The protocol supports the following controllable changes.

—Blank/Show screen (whiteboard access).
—Mute/Allow screen sound.
—Change shared-audio volume.
—Create a window.
—Move a window.
—Shelve/Unshelve (hide/show) a window.
—Destroy a window.
—Bring window to front.
—Send window to back.

The problem is how to authorize changes to these nine types of display space changes and how to dynamically change the authorization mechanism. Automated authorization allows a variety of display server control techniques. Dynamically changing those control techniques allows users to have greater familiarity.

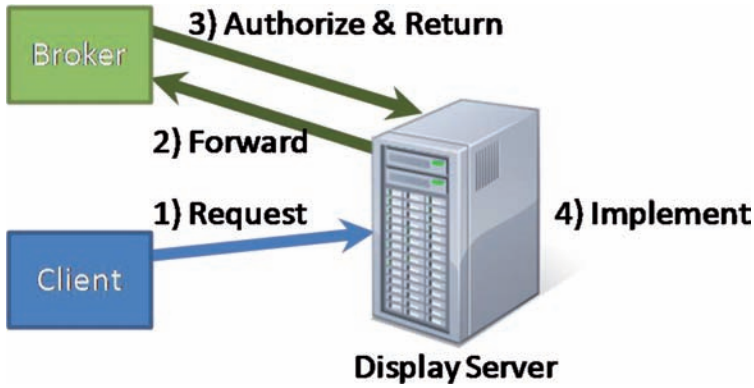


Fig. 3. The display server forwards requests to the broker for authorization and then implements the results.

The window broker protocol designates one of the connected client machines as the *broker*, which is also called the *broker machine*. Clients ask to be the broker and are granted on a first-come first-serve basis. If there is no broker and no client asks to be broker, then the display server implements the free-form control technique.

When there is a broker, the display server forwards requests affecting any of the nine controllable changes (called *forwarded requests)* to the broker machine. The software on the broker may allow, alter, or deny any request. The results of the *broker software*'s decisions are sent back to the display server for implementation. This process flow is illustrated in Figure 3.

Requests that are not forwarded to the broker machine are those that change the content within a window: images, audio, video, and other graphics primitives. Those requests must come from the window's creator and are always honored by the display server.

The broker machine may have client software which attempts to make changes to the display space. For simplicity of software development and the toolkit, the client software is separate from the broker software. The client software requests are routed by the window broker API directly to the broker software (bypassing the display server). The results of calling the broker software directly are sent to the display server and implemented.

The window broker protocol provides automated, plugin-free, and familiar display space control. The broker software provides automation, because it can automatically enforce a control technique by modifying requests. For example, a broker algorithm could enforce boundaries on a particular user by changing any requests that extend outside that user's boundaries into requests that stay inside the boundaries. Because the broker software runs on a personal device, a control technique can be enforced without requiring a new plug-in to be installed on the display server. Including the notion of forwarded requests also provides familiarity by allowing users to carry and use their own broker software.

1.5.1. The User Experience. The SPICE framework is designed for environments in which display servers are unlikely to provide hardware input. The display server may not provide input because it is physically distant from the users (e.g., a large conference hall) or otherwise inconvenient for supplying input. Therefore, the user's personal device provides the input necessary to rearrange windows.

Instead of using PointRight's [Johanson et al. 2002b] approach of treating the display space as an extension of the user's desktop, SPICE treats the display space as a

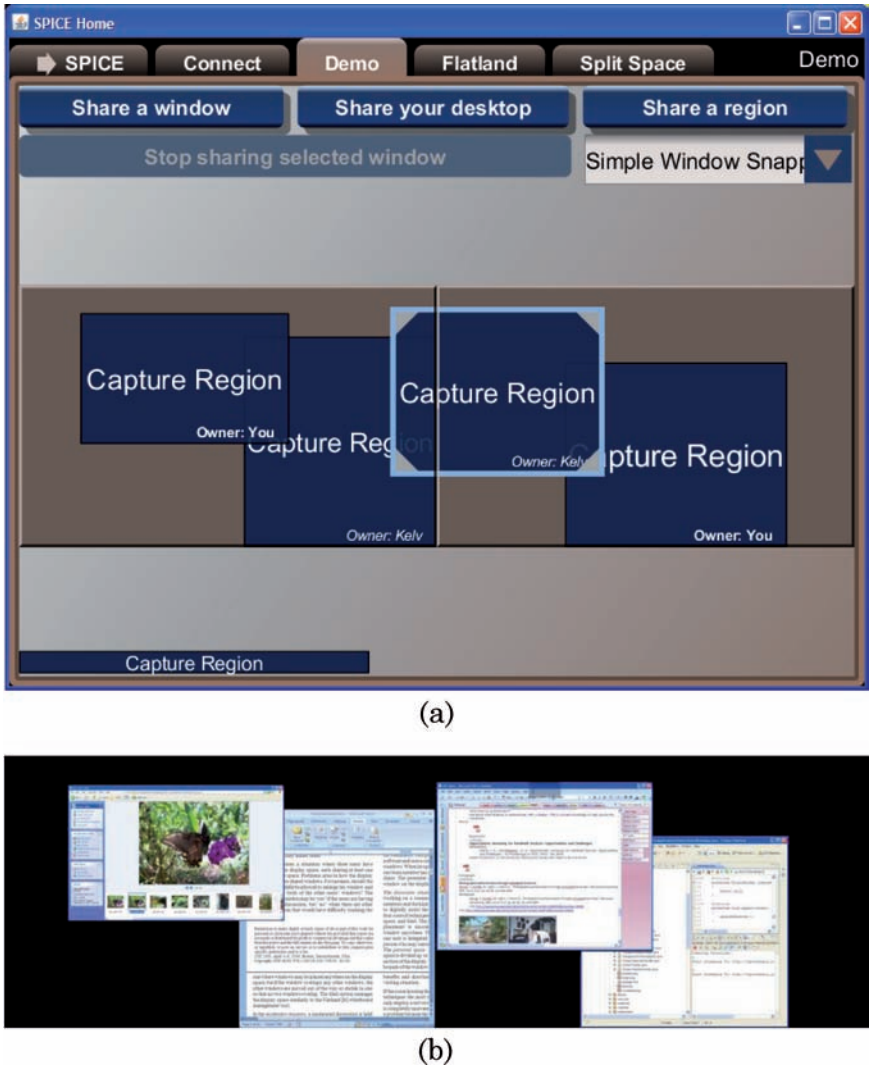


Fig. 4. Window arranger. (a) The icon view for managing the screen space. It shows a representation of the windows shared on a dual-screen display server. (b) A screenshot from the display server.

separate screen space. On the user's personal device is a UI which shows a representation of the display space and its windows—typically a world-in-miniature display. The widget used to represent the display space and the windows is called a *window arranger*. Figure 4 shows a UI that users may encounter when annexing a display server. In the middle of Figure 4(a) is the window arranger which shows all the different windows that are shared on the display server. Other widgets in this UI are for sharing individual windows to the display server or applying an individual control technique. Figure 4(b) shows a screen shot of the actual display space represented in Figure 4(a). In Figure 4(a), the user has selected a window in the middle which she may move, resize, or close. In addition, there is one hidden window which is listed at the bottom of the window. Note that the world-in-miniature view shown in Figure 4(a) is only a potential UI and is not required by the window broker protocol or windowing toolkit.

Client software could implement any of a variety of user interfaces that accomplish this task. A given user will always see the UI that they are most familiar with.

Moving or resizing a window via the window arranger is not performed live. Instead, the window arranger shows a preview of where the user is attempting to move a window. When he releases his mouse button, the window arranger attempts to move the window, to that destination location. To move the window, the software creates a move request and sends it to the display server, which forwards it to the broker machine, which authorizes the request and returns a response to the display server, which then implements the response. If the move request is denied, then the display server sends the client a "rejected" notification. When the client software receives the rejection, the software causes the moved icon of the window to snap back to its previous location, which informs the user that the request was denied. If the broker software moves the window to a different location, then the new location is sent to the client and similarly reflected in the window arranger.

If the broker user is the user trying to move a window, the process is slightly simplified. The broker user still sees, and the same feedback that a typical user sees, and the broker user's requests are treated like any other user's requests. However, the requests bypass the display server and are forwarded directly to the broker software. The broker software's results are reflected to the broker user, and any changes to the display space are sent directly to the display server and performed. Subsequently, all other client devices are notified by the display server.

1.5.2. Solving the Four Situations. The window broker architecture can be used to implement the four example situations (from Section 1.1) via a separate broker algorithm for each situation.

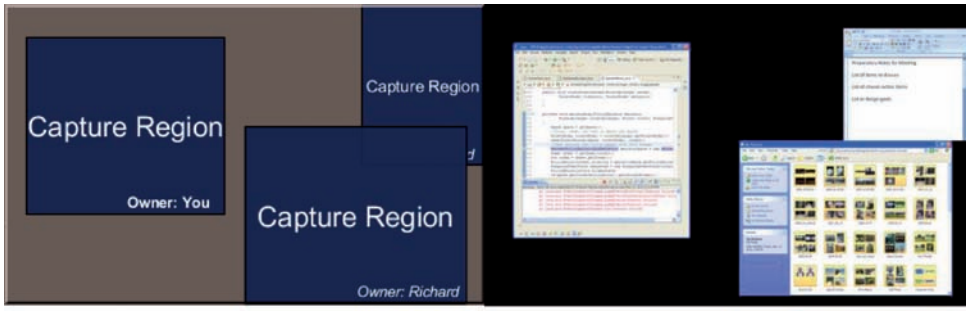
Broker software has one or more sets of rules which govern how the software handles forwarded requests. A set of rules is called a *policy*, and broker software may have more than one policy, but the software will enforce only one at a time. Policies are analogous to control techniques.

Each of the various brokers and policies developed for solving the four situations delineated earlier are discussed next.

Presenter situation. In the presentation situation, the user employs software that has a *presenter broker* algorithm which handles all forwarded requests. The presenter broker uses one of two policies: exclusive or audience. Because the presenter should not be interrupted while giving a presentation, the exclusive policy is the default policy and denies all forwarded requests. With the exclusive policy in use, audience members cannot directly affect the presentation (e.g., show unsolicited material on the display space). Once the presenter finishes her presentation, she may switch to the audience policy. The audience policy denies all requests except create requests. However, the audience policy *shelves* the created window (i.e., hides the window so it is not rendered on the display server). An example of a shelved window is the single window shown along the bottom of the UI in Figure 4(a). The bottom of the UI is also called the *shelf*. The presenter software is informed of the created window so that the presenter may choose to drag that window from the shelf onto the display space, showing the window. In this way, audience members may still participate in the discussion and cannot show unsolicited material.

Discussion situation. In the discussion situation, a group of people work on a common project. This article illustrated four control techniques for managing this space: free-for-all, moderated control, personal space, and tiled. These techniques are described next.

Free-for-all and moderated techniques. The discussion broker software implements the free-for-all and moderated control techniques. The discussion broker software can



(a) The screen management tool a user could have.

(b) The display space output.

Fig. 5. Split space enforcement. Two users share the same display space. Each user's windows may only appear in that user's half of the display space.

employ one of three separate user-chosen policies: moderated-control, only-owned, or free-for-all.

All three policies allow users to create windows. However, like the audience policy, the *moderated-control* policy shelves all created windows, and only the group leader may choose to show any of the hidden windows. The moderated-control policy denies all other forwarded requests. The only-owned policy honors requests that affect windows that the requester owns and rejects requests that would affect other windows. The free-for-all policy allows anyone to affect anything shown on the display space. Now the various group members can share and discuss individual windows on the display space using either the free-form, only-owned, or moderated-control technique.

Personal space technique. The personal space technique is implemented as separate broker software called the *split space broker*. This broker has a single policy with three broad rules.

—Divide the space equally among the participating users.
—Allow users only to move windows they own.
—Keep a user's windows within that user's partition.

This split space policy acts similarly to that of the Dynamo display space management paradigm, except that the space is automatically allocated to each user rather than "carved out" by each user.

Figure 5 illustrates what the display space partitioning looks like. Figure 5(a) shows a clipping from the window arranger with window placements, while Figure 5(b) is the screenshot of the display space represented in Figure 5(a). In this example, two people are connected to the display space, so the policy divides the space horizontally. Both users get equal-sized partitions of the space that are as close to square-shaped as possible (minimum perimeter length): one partition is on the left, the other on the right.

The split space broker software must be able to handle adding and removing participants, particularly when the display space is already divided among some participants. Suppose some user connects to the display space not to share content, but to copy content and take notes. Connecting to the display space should not cause a rearrangement of all the windows on the display space, which can be frustrating to the users who already have space allocated and are utilizing that space. Consequently, the broker software allows the user—or *broker user*—to select which of the connected clients are considered participants.

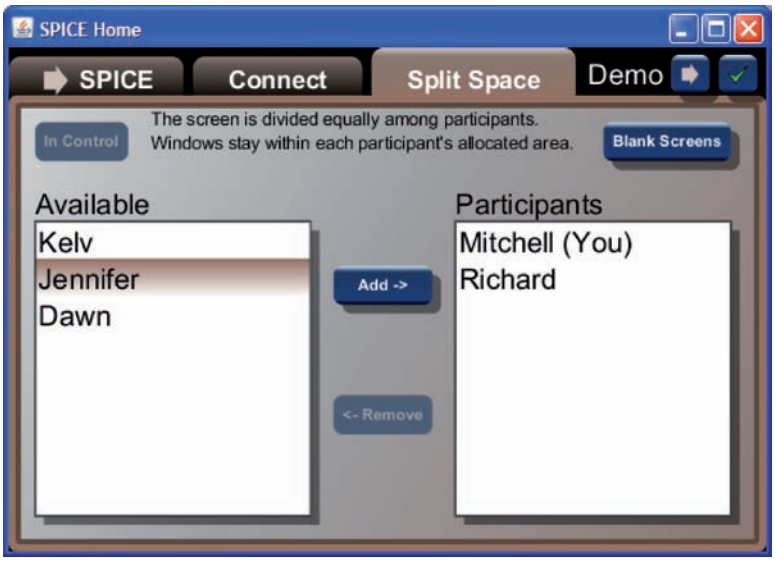


Fig. 6. Participant selection UI. The broker user chooses participants from the list of connected machines.

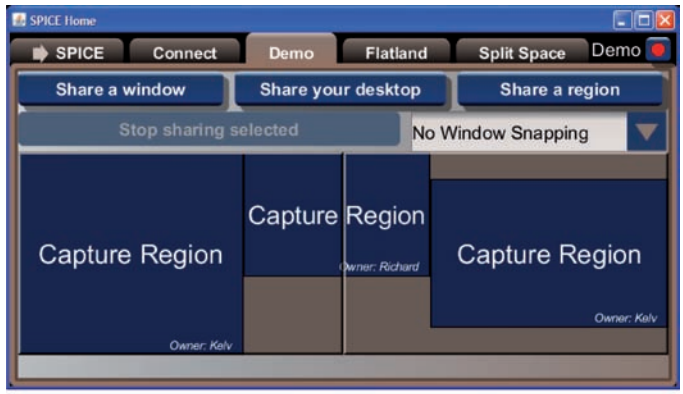
To allow the broker user to choose which client machines are participants, the split space software provides the UI in Figure 6. When a client becomes broker, the display server supplies that client with a list of all the connected clients and notifies the broker machine when a client connects or disconnects.

The split space broker software uses these connection notifications to update the participant selection UI shown in Figure 6. This UI lists all the non-participating clients in the left column, and the broker user may move any of those clients to the "participants" category on the right.

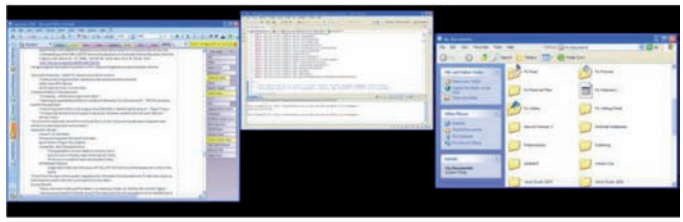
With each selection or removal of a participant, the split space broker software re-divides the display space. The software attempts to preserve the window arrangements within divisions and attempts to keep the divisions as close to their original location as possible.

Tiled technique. The tiled technique is implemented as the only policy in the *flatland broker*. This broker allows anyone to share any information on the display space. The only constraint is that no two windows may overlap. Rather than enforcing this constraint by preventing window movement, any window may be moved to any location on the display space, and all other windows are move out of the way. Figure 7 shows a screenshot from the client machine in Figure 7(a) and from the display server in Figure 7(b).

Moderated situation. The moderated situation is a combination of the presentation situation and the personal space technique. No audience members can interrupt the various presentations, only the presenters may show windows on the display space, and windows can only be shown and arranged by the client with floor control. The moderator launches her *moderator broker* software, becomes the broker for the display space, chooses who the participants are via a UI similar to the one in Figure 6, and then moderates the discussion. She chooses one person at a time to have floor control and, thus, absolute control of their own windows on the display space.



(a) The icon view for managing the screen space on a dual-screen display server.



(b) A screen shot from the display server.

Fig. 7. Flatland window management where no two windows may overlap.

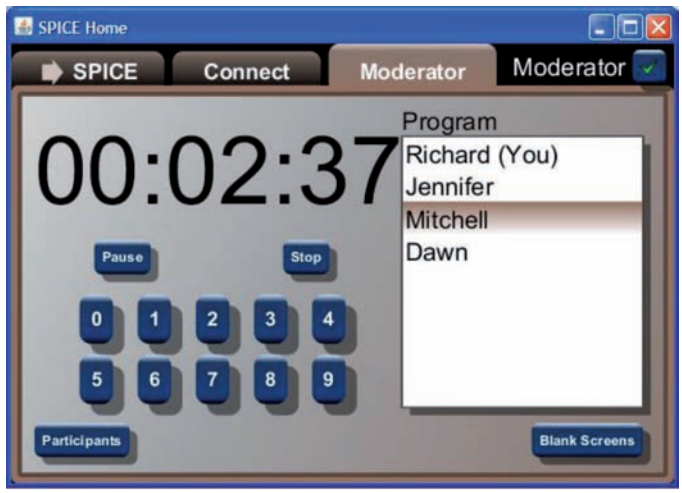


Fig. 8. Possible moderator software UI. The person selected on the right is the floor. The countdown on the left is used to time the current speaker.

The moderator uses the UI in Figure 8 to select the participant to assign as the floor. The chosen participants are shown down the right side, and the selected one is the floor. For example, Mitchell has the floor in Figure 8. The moderator can also time Mitchell's presentation using the clock on the left, and give him subtle feedback about his progress.

Visitor situation. When the moderator is asked to travel to a new location, she can become broker for that display space. Because of the window broker API, she does not need to install any software supplied by that display space's owner onto her personal device, nor must she install new software on the display space to support her moderator software. The only software that must be consistent between her client machine and the display server is the window broker protocol. With such consistency, she can confidently manage the display space for a separate discussion.

Situation solutions summary. In each of these situations, the broker software is always installed on a user's personal device. The display server is always just a display server and performs the tasks required to display information. Because the display server can forward requests to the broker device via remote procedure calls, the display server does not need to change to support a new interactive technique.

On the other side of the issue is the fact that users always bring their own broker software with them. This broker software is always the software that they know and are familiar with. The software presents a UI on the user's personal device via a known screen and input devices, which keeps the software familiar regardless of the environment.

1.6. New Challenges

With the window broker protocol, some new challenges present themselves. These challenges include situations without brokers, wresting control from a broker, distrusted display servers, asynchronous requests, and client-side broker preview.

1.6.1. Situations without Brokers. Some collaborative situations may not need a window broker. In such cases, the display server performs all of the requests from any connected device without forwarding the requests. This approach effectively results in the free-for-all control technique. Conflicts that occur between users are expected to be resolved socially.

1.6.2. Wresting Control from a Broker. Envision an English professor who has just finished instructing a class. The English professor neglects to relinquish broker control as he leaves. A math professor teaching the subsequent class enters the room a few minutes later, attempts to become broker, and discovers that someone else has control. The math professor must be able to quickly and easily gain control of the display space.

Or, imagine the case in which the math professor attempts to take control of a display server and finds that someone else in the room is already broker. This student may be attempting to surreptitiously interfere with the math professor's presentation by allowing the professor to give his presentation, but during the presentation, the student rearranges windows or shows additional content.

There are two options for resolving these situations: ask for or take control. To ask for control means physically and personally asking the English professor to relinquish control. Unfortunately, the math professor may not be aware that the English professor is the prior professor in the room, or know the English professor. Worse yet, the English professor may have granted control to a student who also left; in such case, tracking that student down may be difficult. Instead, the window broker protocol must allow the math professor to take (or wrest) control from the current broker.

The math professor is in the same room as the display space, so wresting control should be straightforward. The default SPICE display server software provides a UI which can discard the current broker and possibly assign a specific machine as broker. Other installations may have a different setup but a similar goal. This UI may be a physical button on the display server which, when pressed, disconnects the current window broker, or the UI may be a more sophisticated interface listing

the currently connected users from which the math professor may select himself. The key is that physical control of the display server can override any currently assigned broker.

1.6.3. Distrusted Display Servers. Within a controlled environment (e.g., a corporate or classroom environment) users can probably expect that a display server is completely trustworthy. The maintenance staff will keep the display servers free of malware that could damage client devices. However, not all display server environments would necessarily be as controlled or safe. For instance, a display server embedded in a restaurant table, in a mall kiosk, or a hotel room may not be as well maintained. These display servers may have malware that was transmitted to the display by a previous user intentionally or unintentionally. The window broker protocol should not be available as a virus vector so that future users may be protected from infection.

The window broker protocol does not act as a virus vector because it never transmits software. Instead of transmitting code, the window broker protocol supports a limited set of commands which client and server machines interpret.

Flaws in the window broker protocol implementation (e.g., buffer overruns) may still exist on the display server or client machines, but the design of the protocol does not invite infection.

1.6.4. Asynchronous Requests. Some personal devices may generate situations with high network latency or temporary disconnections from the display server. The display server should not lock up during these situations and neither should the personal devices. Within the window broker protocol, there are two main sources for high network latency or temporary disconnections: small devices and manual authorization.

Small devices, such as smartphones, tablets, and netbooks, could become broker machines. These machines might not share windows on the display server, but they could still manage the screen space. For instance, in the moderator situation, the user employs her smart phone to moderate the discussion without showing any windows. These small devices often have slower processors than laptops and may be limited to slower network speeds.

In some cases (e.g., iPhone [Apple 2010] or Windows Phone 7 Series [Microsoft 2011]), these devices can only execute one application at a time. Although iOS 4 claims support for multitasking, it still can execute only one application at a time, there by suspending any application that is not the foremost application. This means that if the first application is the window broker, then it could be swapped out at any time; the user may temporarily be swapping to another application and could be back soon. As a result, the window broker software must be prepared to resume as broker at any time.

If the display server drops the smartphone as the broker because the window broker software is swapped out, then the smartphone may not be able to resume as broker. Consequently, someone else may take control in the interim. If the display server stops accepting requests—or locks up—while the broker software is swapped out, then the display server will not be able to update content supplied by other attached users. For example, if the moderator swaps to another application momentarily to take a note, then the person with floor control may not be able to change to the next slide. The display server must allow the smartphone to continue being broker and must not lock up while waiting for that broker application to resume.

The window broker protocol employs a simplified network remote procedure call protocol. Remote procedure calls are typically treated as a standard (albeit slow) method call. To ensure that the call waits until a response is available, most RPC systems rely on the call stack and TCP connection to maintain state. Because the blocking thread is frequently the UI rendering thread, an application can appear to freeze or lock up.

If a slower device is the broker and if all requests to the device are synchronous (meaning the display server blocks on a thread and socket while waiting for a response), then the slower device becomes the bottleneck in the experience for all other users of a display space. For instance, the display space could lock up while waiting for a response to a "move window" request. In addition, the client requesting to move that window would lock up while waiting for the display server which is waiting for the broker software to respond.

Locking up while waiting for a response from the broker machine is not acceptable. The protocol must support broken connections and occasionally long periods of time between requests and responses.

Related to the problems created by slower devices are manually authorized requests. In most cases, broker algorithms are likely to be completely automated. In some cases, however, the broker algorithm may make decisions based on user input. For instance, the algorithm may show a prompt when a client requests to blank or show the screen. In such cases, the display server and other client software should not block while waiting for the broker machine to process the request.

To support the high latency and temporary disconnection required for slower devices and manually authorized requests, the window broker protocol requests are always asynchronous (meaning that the display server and client do not block on a thread and socket while waiting for a response). The source of each request assigns that request a unique ID and the broker tags the corresponding response with the same ID. Then the display server can match a response with the original request and implement the response accordingly. When the broker machine is temporarily disconnected, the display server queues any incoming forwarded requests and transmits the requests to the broker the next time it connects. In the meantime, any client software waiting for requests to be authorized are still interactive, but the display server will not reflect the requested changes until the broker responds.

To detect when a temporary disconnection is actually a permanent disconnection—regardless of whether that client is the window broker—the broker protocol uses a technique similar to that of HTTP session management. If a client does not reconnect within a small timeout—two minutes by default—then the client is assumed to have disconnected (e.g., the owner turned off his personal device and left the room without explicitly disconnecting). The display server then dumps all pending messages for that client. If the client is the window broker, then the display server processes each pending request, denying each. Each request is denied so that the display space maintains its current state instead of trying to merge all the pending state-change requests; having the display server suddenly update to match all the pending requests can be a visually jarring experience for the other clients. After clearing all pending queues, another client may become broker.

1.6.5. Client-Side Broker Preview. The broker can greatly affect the actions that other clients may perform. It may accept, deny, or alter any of the forwarded requests. To help a non-broker user make good decisions, the window arranger should provide feedback about possible actions. For instance, if the flatland broker is currently enforced, a non-broker user might like to see a preview of how the other windows would be positioned when she moves a window.

Consider Figure 9, which shows the client-side view of the display server's state. In this case, the current broker is the split space broker, and the user is enlarging the window by dragging the bottom-right corner. As can be seen in Figure 9(a), this move is illegal per the split space broker because the window encroaches on another user's partition. It would be better if the client were presented with the legal move shown in Figure 9(b).

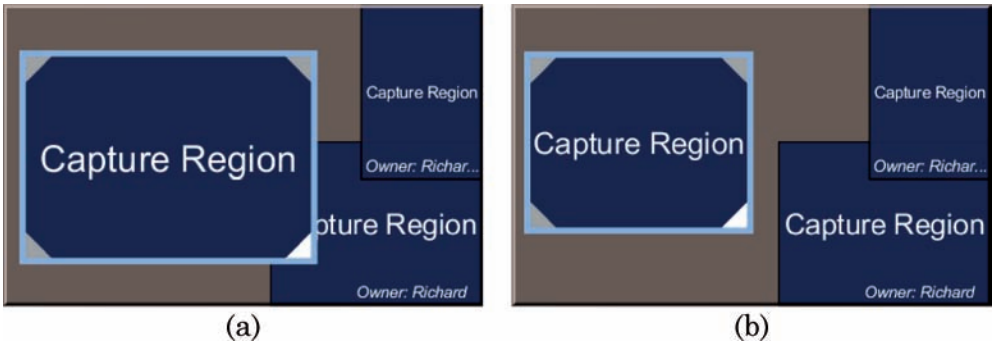


Fig. 9. Reflecting split-space constraints on a client machine. (a) Shows an illegal move that an uninformed UI could present. (b) Shows the proper, legal action where the window is constrained to the left half of the display space.

The window arranger provides direct feedback about potential actions. One option would be to ask the window broker directly about the limits or consequences of a user's actions. Unfortunately, because the broker could be disconnected or the network could be a little slow, asking the broker directly for feedback about potential actions may not give timely information.

Another option would be for the broker to transmit software via the display server to each of the other clients. The code transmitted would be some common framework, such as JavaScript [Flanagan 2006], Java, ActionScript [Moock 2007], or MSIL [Thai and Lam 2002]. However, this is a highly insecure option. The window broker could transmit malicious broker software to the display server, which it would then transmit to each client, or the display server could have malicious software installed, which it transmits to clients in lieu of the actual broker software. The only way to mitigate this approach is to painstakingly sandbox the environment for hosting the transmitted software.

Instead of these prior two options, the window broker protocol makes a compromise, that is, transmitting static information about the window broker and possibly using the same broker software if it is already installed on the client machine. To implement this, the window broker protocol has two additional features: policy hints and policy emulators. *Policy hints* supply a static, simplified version of the currently enforced policy and give client software a broad idea of potential actions. *Policy emulators* give information about the actual policy being enforced so that clients that have that policy installed can instantiate the policy and use it to emulate what the window broker software would actually accomplish.

Policy hints. Many brokers implement similar rules, although sometimes with slightly different implementations. For instance, the discussion broker's only-owned policy is similar to the split space and moderator broker policies. Likewise, the discussion broker's only-broker policy is identical to the presenter broker's audience policy (which creates windows but shelves them) and similar to the presenter broker's exclusive policy (which does not allow creation at all).

Most of the decisions these window brokers make are straightforward, simple decisions. These decisions are usually to allow or deny and are usually based on ownership or participation. In some cases, extra processing is needed for the broker software to make a decision (like the split space or flatland brokers which keep windows within a partition or move other windows in response). The window broker protocol takes advantage of the similar simple decisions of many brokers while allowing more complex decisions by other brokers.

Participants Only		*False*
Create Window		*Brokered*
Change Volume		*Denied*
Change Mute		*Denied*
Change Blank		*Denied*
	Owned	**Not Owned**
Move	*Brokered*	*Brokered*
Shelve/Unshelve	*Brokered*	*Brokered*
To Front	*Allowed*	*Allowed*
To Back	*Allowed*	*Allowed*
Destroy	*Allowed*	*Allowed*

Fig. 10. Policy hints for the tiled control technique in the flatland broker. This is a participant-based broker, and users are allowed to alter any windows, but the broker may adjust a window's final positions.

To inform client machines about possible actions without transmitting code, the window broker protocol supports transmitting *policy hints*. These hints are key-value pairs of a hint type and its value.

There are 15 hints, 14 of which represent a forwarded request. The 15th is described later. For each request that affects a window (move, destroy, shelve, to front, and to back), there are two hints: one for window owners and another for non-owners. The value for each hint is one of three values: allowed, denied, or brokered. *Allowed* means that the request will always be authorized by the broker software. *Denied* means that the request will always be denied by the broker. *Brokered* means that the result is too complicated to express as either allowed or denied; the client can request that action, but the broker could allow, deny, or alter that request. The "create window" policy hint supports a fourth option called *Shelved*. Shelved means that window creation is allowed, but the window is hidden.

The 15th policy hint is a Boolean which expresses whether the broker software is participant based. If the software is participant based—such as the moderator and split space brokers—then participants use the 14 other hints, and non-participants' software assumes every action is denied.

Consider the policy hints for the tiled control technique, which are shown in Figure 10. This control technique constrains the movements of the affected window (by keeping it within the display space's bounds) and arranges any neighboring windows. Hence the "Brokered" tag on window creation, movement, and shelving.

Also observe the policy hints for the only-owned policy (which is part of the discussion broker) shown in Figure 11. This policy is participant based and only allows users to affect their own windows. Creating windows is brokered because the participant with the floor can show those windows immediately, while all other created windows are shelved. Similarly, only the participant with floor control can change the shelved state of her windows.

For a window arranger on a client machine to give proper feedback about what options are allowed, the window arranger must know whether its executing client machine is a participant. Client machines do not have a direct connection to the window broker because the display server is an intermediary. Consequently, the display server must inform each client machine about whether it is a participant. The display server can only know which clients are participants if the window broker exposes that particular information to the display server. So, a participant-based window broker must inform

Participants Only		*True*
Create Window		*Brokered*
Change Volume		*Brokered*
Change Mute		*Brokered*
Change Blank		*Brokered*
	Owned	**Not Owned**
Move	*Allowed*	*Denied*
Shelve/Unshelve	*Brokered*	*Denied*
To Front	*Allowed*	*Denied*
To Back	*Allowed*	*Denied*
Destroy	*Allowed*	*Denied*

Fig. 11. Policy hints for the moderator technique in the moderator broker. This is a participant-based broker, and users are allowed to alter their own windows but not others.

the display server when a participant is added or removed. Then the display server can inform each of the clients of changes in the participants list.

Exploitability. The policy hints portion of the window broker protocol is difficult for a malicious display server or window broker to exploit. Client software interprets the policy hints without executing server- or broker-supplied software because each policy hint is a static value and contains no code. Although the display server may alter the broker-supplied values, altering those values provides no visible benefit to the display server toward exposing sensitive data or infecting a client machine.

Policy emulators. The policy hints effectively express the simple decisions that brokers may make. However, brokers, such as the tiled and split space brokers, have sophisticated algorithms that are too complicated to describe using policy hints. These algorithms cannot be transmitted using the policy hints portion of the window broker protocol.

Rather than transmit the window broker's code, client machines could use software they already have installed. Envision a group of three people using the split-space control technique. The first worker is the broker user and has the split space broker software installed, the second user works at the broker user's institution, and the third user is visiting. The second user has the discussion broker installed, while the visitor does not. When the first user's personal device annexes the display space, it requests to be broker and informs the display server of the policy hints, which include the name of the split-space control policy.

The second-user annexes the display server, and her personal device is informed that the split-space control technique is in use. Her personal device finds and executes the locally installed split-space control code so that the window arranger can use it to emulate a preview that keeps her windows within her partition, similar to what is illustrated in Figure 9(b).

When the visitor's device annexes the display server, it is also informed of the policy hints, including the name of the split-space control policy. However, the visitor's machine does not have the split space software installed; therefore, akin to what is shown in Figure 9(a), she can drag windows around, but the preview does not constrain the windows to her partition. However, she only sees the constraint after the response finally arrives from the broker. Thankfully, because of the policy hints, her

window arranger reflects that she cannot drag or resize any of the other people's windows.

To inform clients of the current broker software, the window broker protocol transmits two extra strings with the policy hints. The first string uniquely identifies the class that may be used to instantiate the broker software, called a *policy source*. This policy source is usually a fully qualified class name, such as those used in Java or .NET [Thai and Lam 2002]. The runtime uses this class name to find and instantiate that policy source via reflection. The second string is the *policy version* and is passed to a single method on an instance of the policy source. The method (called getPolicyEmulator) takes in the policy version and returns an instance of a *policy emulator*. The policy emulator can be used to emulate the execution of the current broker's policy.

The policy emulator should exactly emulate the broker software. However, the broker software frequently has more information in its model than just the state of the display server. For instance, the split space broker assigns a rectangular partition to each participant. Although the display server can inform the policy emulators of the chosen participants, the display server is generalized enough that it cannot inform the emulators of the partitions. The emulators could deduce the partitions from the current window arrangement, but if more than one client has no windows shown, then the emulators cannot precisely deduce the partitions for all clients. The broker software must be able to expose additional state data via the display server. The additional state data allows policy emulators to accurately emulate the broker software.

To transmit extra state information to the clients, the display server has a generic annotation mechanism. The broker software may attach key-value pairs of strings as metadata to window handles or the broker handle. The broker handle is an object exposed through the window broker API which represents the actual broker machine and software. This handle provides clients access to the broker's policy hints and the name of the person who is the broker user.

When the broker software annotates a window or the broker handle, that key-value pair is transmitted to the display server and subsequently to each of the connected clients. To extract the annotation, each client machine's policy emulator may then inspect the window handles or the broker handle.

For the split space broker software to inform each client of how the display space is divided, it serializes the array of partitions to a string and then annotates the broker handle with that string. The display server transmits this annotation to each client, where the software on the second user's personal device can decode the partitions. Now the split space emulator on the second user's personal device has the complete model that the broker user's software has and can present an accurate preview of window movements.

Exploitability. Because no code is transmitted from the broker machine to each of the clients, the clients are protected from potential infection. In addition, the toolkits on the client machines do not need to be designed to sandbox the broker software.

The policy emulator approach is less secure than the policy hints but more secure than transmitting code. If there is a known exploit in a particular window broker, a malicious display server could easily transmit the exploitable broker's policy source and policy version strings instead of the current window broker. Then, the server could attempt to exploit the weak policy emulator by transmitting malicious serialized model data.

Developers of such network-enabled software should always create code that protects itself from such exploits. Client machines can protect themselves from exploitable

window brokers. A well-maintained (i.e., regularly patched) client machine would have a list of known exploitable broker sources and would prevent those sources from ever being instantiated.

1.7. Summary

This article introduced the window broker protocol. This protocol allows a separate machine to participate in the window arrangement on a display server. In particular, users may bring their own machine to a display server and enforce new window arrangement rules. This flexibility provides automated, plugin-free, portable display space control.

This article also illustrates several different automated broker software implementations: presentation, discussion, split space, flatland, and moderator brokers. Each broker is implemented in Java and demonstrates the portable, plugin-free nature of the display space control afforded by the window broker protocol.

In addition, the window broker protocol is designed to support small devices that may be intermittently connected to the display server. This support is implemented by messages in the protocol which operate asynchronously.

To provide feedback for client machines which may or may not have the broker software installed, the window broker protocol also provides policy hints. These hints let a client machine know what broker software is being used (in case the client has that software available for emulation), and some of the basic decisions which the broker makes (in case the client does not have that software available).

REFERENCES

Apple. 2010. iPhone, homepage. http://www.apple.com/iphone/. (Last accessed 10/10).

Biehl, J. T., Baker, W. T., Bailey, B. P., Tan, D. S., Inkpen, K. M., and Czerwinski, M. 2008. Impromptu: A new interaction framework for supporting collaboration multiple display environments and its field evaluation for co-located software development. In *Proceedings of the 26th Annual SIGCHI Conference on Human Factors in Computing Systems (CHI'08)*. 939–948.

Flanagan, D. 2006. *JavaScript: The Definitive Guide.* O'Reilly Media, Inc, Sebastopol, CA.

Gosling, J., Joy, B., Steele, G., and Bracha, G. 2000. *Java Language Specification: The Java Series*, 2nd Ed. Addison-Wesley Longman Publishing, Boston, MA.

Izadi, S., Brignull, H., Rodden, T., Rogers, Y., and Underwood, M. 2003. Dynamo: A public interactive surface supporting the cooperative sharing and exchange of media. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technol (UIST'03)*. 159–168.

Jiang, H., Wigdor, D., Forlines, C., Borkin, M., Kauffmann, J., and Shen, C. 2008. LivOlay: Interactive ad-hoc registration and overlapping of applications for collaborative visual exploration. In *Proceedings of the 26th Annual SIGCHI Conference on Human Factors in Computing Systems (CHI'08)*. 1357–1360.

Johanson, B., Fox, A., and Winograd, T. 2002a. The interactive workspaces project: Experiences with ubiquitous computing rooms. *IEEE Pervasive Compu. 1*, 2, 67–74.

Johanson, B., Hutchins, G., Winograd, T., and Stone, M. 2002b. PointRight: Experience with flexible input redirection in interactive workspaces. In *Proceedings of the 15th Annual ACM Symposium on User interface Software and Technology (UIST'02)*. 227–234.

Liu, Z. 2007. Lacome: A cross-platform multi-user collaboration system for a shared large display. Computer Science, University of British Columbia. http://hdl.handle.net/2429/378.

Microsoft. 2011. Windows Phone 7 Series, homepage. http://www.windowsphone7.com/. (Last accessed 6/10).

Moock, C. 2007. *Essential Actionscript 3.0*. 1st Ed. O'Reilly, Sebastopol, CA.

Mynatt, E. D., Igarashi, T., Edwards, W. K., and LaMarca, A. 1999. Flatland: New dimensions in office whiteboards. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems: The CHI Is the Limit (CHI'99)*. 346–353.

Richardson, T., Stafford-Fraser, Q., Wood, K. R., and Hopper, A. 1998. Virtual Network Computing. *IEEE Internet Comput., 2*, 33–38.

Scheifler, R. W. and Gettys, J. 1986. The X window system. *ACM Trans. Graphics, 5*, 2, 79–109.

TAN, D. S., MEYERS, B., AND CZERWINSKI, M. 2004. WinCuts: Manipulating arbitrary window regions for more effective use of screen space. In *Proceedings of the CHI'04 Extended Abstracts on Human Factors in Computing Systems*. 1525–1528.

THAI, T. L. AND LAM, H. 2002. *.NET Framework Essentials 2nd Edition*. O' Reilly, Media, Sebastopol, CA.

TRITSCH, B. 2003. *Microsoft Windows Server 2003 Terminal Services*. Microsoft Press.

Received May 2011; revised January 2012; accepted February 2012