Role-Based Control of Shared Application Views

Lior Berry, Lyn Bartram and Kellogg S. Booth
University of British Columbia
Computer Science Department, Vancouver, BC, Canada
{berry,lyn,ksbooth}@cs.ubc.ca

ABSTRACT
Collaboration often relies on all group members having a shared view of a single-user application. A common situation is a single active presenter sharing a live view of her workstation screen with a passive audience, using simple hardware-based video signal projection onto a large screen or simple bitmap-based sharing protocols. This offers simplicity and some advantages over more sophisticated software-based replication solutions, but everyone has the exact same view of the application. This conflicts with the presenter's need to keep some information and interaction details private. It also fails to recognize the needs of the passive audience, who may struggle to follow the presentation because of verbosity, display clutter or insufficient familiarity with the application.

Views that cater to the different roles of the presenter and the audience can be provided by custom solutions, but these tend to be bound to a particular application. In this paper we describe a general technique and implementation details of a prototype system that allows standardized role-specific views of existing single-user applications and permits additional customization that is application-specific with no change to the application source code. Role-based policies control manipulation and display of shared windows and image buffers produced by the application, providing semi-automated privacy protection and relaxed verbosity to meet both presenter and audience needs.

ACM Classification H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

General Terms Design, Security, Human Factors

Keywords: CSCW ,Sharing, Application, Bitmap, Role, Policy, Privacy, Verbosity, View

INTRODUCTION
People working in groups increasingly rely on the ability to share views of an entire work session or a specific application for co-located or distributed cooperative work. Although new tools and frameworks introduced in recent years support a wide range of collaboration formats, the dominant format is still that of a single person, the *presenter*, sharing a view of her workstation while others, the *audience*, watch. This *generalized presentation* setting applies to people giving conference or classroom presentations, demonstrating software, training others, or engaging in collaborative work where a shared document on a public display is the focus for group discussion. The same pattern is applicable during turn-taking editing sessions where at any given time there is only a single editor (the presenter) and others are just viewers (the audience).

For co-located groups, view sharing is often done by replicating the video signal from the presenter's computer onto an external public display. For distributed groups bitmap-based screen sharing protocols such as VNC [12] provide view sharing. Functionally, both solutions are equivalent and afford three properties that more sophisticated collaboration-aware solutions may be hard-pressed to achieve: (i) collaboration is transparent because any application can be shared without requiring it to know it is shared; (ii) viewers do not need to install the application, which is often an unacceptable imposition due to licensing or other issues; and (iii) there is no need to synchronize views between the presenter and audience because only one copy of the application is running.

The last point is an important benefit, but it is also the chief drawback of these schemes. A strict "What You See Is What I See" (WYSIWIS) mode is imposed – all viewers are forced to share the exact same view that the presenter sees, despite the roles that they play within the group. This conflicts with the different needs of the presenter and audience, so a "relaxed WYSIWIS" mode has been suggested in the literature [14]. Unfortunately, none of the current bitmap-based solutions address this successfully.

Motivation
The initial problem motivating our work was the lack of support for presenter privacy. While generally interested in sharing a view with her audience, there are often interactions or document components a presenter would like to keep private. These may be interactions with other running applications on the desktop or with parts of the shared application that are deemed private (e.g. a file open dialog showing a private folder or a navigation history list). Recent work [6] shows that most people would like to take measures to minimize these exposures. Yet, it is clear that someone engaged in a live presentation might not attend effectively to her privacy at the same time. The need to limit publicly shared information will only intensify as collaboration technologies and application sharing are becoming part of day-to-day work.

UIST'05, October 23-27, 2005, Seattle, Washington, USA.

Presenters may find themselves in an ad-hoc presentation-like mode without having the time to prepare or while having to do other tasks on their computer in parallel.

A different problem arises when we consider the audience experience. Passive viewers may wish to control the type and level of information presented to them and they may require assistive cues to accurately follow the presenter's interactions with an application. Passive viewers often find themselves searching for the current point of interaction (a problem intensified on large screen displays [2]) or being forced to view tedious interactions by the presenter (such as searching for a menu item or adjusting display parameters) that are irrelevant to their interests. So we may need to display less information, or we may need to display more information to compensate. Some needs are already addressed by screen recording tools, like Camtasia.[1] Visual enhancements and cleanup can be applied to a recorded movie in a separate editing session. Being able to apply these enhancements in real time would be beneficial for effective collaboration.

These two problems are actually both manifestations of the need to provide role-appropriate views of an application to each group member. It is possible to use more elaborate collaboration-aware tools or to adapt existing tools to run in a synchronized mode [17], so that presenter and audience have similar but non-identical views. While these custom solutions enable one to flexibly craft views as desired, they fall short of meeting the advantages of existing bitmap-based protocols outlined before (namely requiring all parties to have a copy of the application and depending on application-specific features). The proof of this claim is the fact that time and time again people resort to bitmap-based sharing rather than more sophisticated solutions. Thus, any improvements to bitmap-based sharing modes are still very relevant to improving collaboration.

Our contribution

We have developed a novel framework for adapting the live shared view of applications to meet the presenter's privacy requirements and to provide viewers with suitable cues and level of detail, balancing concerns for privacy and awareness. Our system uses bitmap-based techniques to transparently share visual information, while allowing policies to be specified that control the generation of different views for the different roles within a collaborating group, by reusing the visuals from the running application.

The system conducts an "over the shoulder" monitoring of what the presenter is doing, actively manipulating the published visuals in three ways: (i) *spatial* and hierarchical transformations for selective sharing, repositioning and scaling of application components (including sub-window regions); (ii) simple local or global "*chromatic*" image filters, such as blurring, applied to the visual surface of the application; and (iii) *temporal* or application-state-based transformations applied to the timeline of captured interactions.

To make these manipulations more useful, some reliance on application-specific semantics is required to extract locations of semantic UI objects and to acquire the application's state. We designed a plug-in architecture and used a set of heuristics for obtaining such semantics without giving up too much generalizability.

We implemented a prototype of the system and demonstrated how it can be applied to several commercial off-the-shelf popular applications, disproving to some extent the misconception that bitmap-based application sharing forces strict WYSIWIS (see discussion in [3]).

A SAMPLE SCENARIO

Bob and Carol are both managers and Ted and Alice are team members in a group of employees. Bob, the presenter, is discussing the team's budget using a spreadsheet on his laptop that is being projected onto a large shared screen viewable also by Ted and Alice. Carol views Bob's laptop remotely, using VNC. Some of the data, parameters and interactions in Bob's spreadsheet are confidential and should not be exposed to Ted and Alice, but should be available to Bob and Carol (see Figure 1). Bob resolves some parameters using other applications (e.g. an IM client with Carol). A key requirement is that Bob must see the private parameters and change them, but without exposing them to Ted and Alice.

Bob could extract just the relevant data to a new spreadsheet and project it on a shared auxiliary screen with the master spreadsheet visible solely on his laptop. In theory this solves the problem, but not in practice. Extracting the appropriate information with its dependencies is possible (though not trivial). However, synchronizing the spreadsheet versions when changes are made is time-consuming, error-prone and requires redundant computations (even when using automated scripts). Bob will still need to make sure that updates do not reveal private information. The cognitive overhead of managing the session will diminish his ability to focus on the budget.

Carol has to watch Bob's verbose UI interactions, some of which are distracting or take up her screen space. Ted and Alice, on the other hand, only see the results of Bob's manipulations echoed in the secondary spreadsheet and are missing critical interaction cues.

This scenario demonstrates the need for role-based viewing policies. Bob and Carol need to see a different view than Ted and Alice, because of privacy concerns. All passive viewers need an augmented, "cleaned up" version of what Bob sees in order to comfortably follow his actions.

PRIVACY CONCERNS

In the context of live generalized presentation scenarios, we make a distinction between two types of privacy concerns. There are privacy concerns that affect security (e.g. revealing one's credit card number), and there are privacy concerns that do not pose a real security threat, but may put one in an awkward position. As Palen and Dourish [11] note "... in video conferencing, shared calendar management and instant messaging communications, concerns most salient to users include minimizing embarrassment, protecting turf (territoriality) and staying in control of one's time." Our system supports both types of privacy concerns.

[1] http://www.techsmith.com

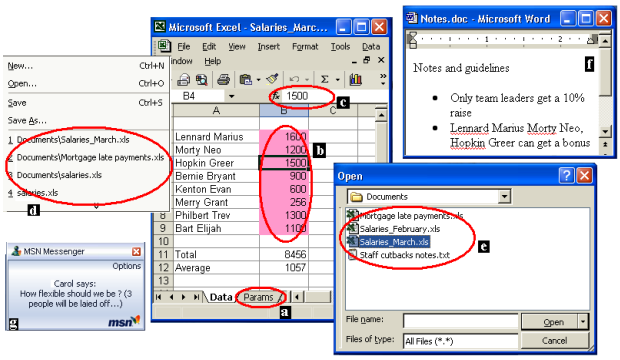

Figure 1: Bob's privacy needs: The "Params" worksheet (a) is entirely private, the salaries cell range (b, c) is also private, and the file dialog and menu (d, e) expose private files. Bob uses his notes (f) and an IM client with Carol (g).

Private information appearing on the screen can be dealt with in several levels:

Task – A presenter needs only to expose the windows and components that are part of the shared task. This may entail sharing several applications or only one instance of an application (e.g. a single document). It is often not necessary (or desirable) to expose the entire desktop.

Window – An application usually comprises more than just a single window. There are dialog boxes, menus, palettes, toolbars and sub-window frames. In many cases these contain private information (file browsing dialog), appear at awkward moments (error dialogs) or just take up screen space. Clearly, not all of these components should be shared.

Visual Surface – Information bits visible on a window's surface, such as underlying document objects or the contents of UI widgets. Our approach demonstrates how these can be dealt with in the *image buffer* level.

When working on our system prototype we realized that in some cases it is more effective to define a state of the application as private and freeze updates on the public copy until the application exits the state. This is useful when private data is mapped to externally inaccessible objects or associated with a large set of objects that cannot be treated individually (e.g. switching to a private worksheet or a show-comments mode or when an arbitrary error occurs).

Some of the types of on-screen private data that require handling are:

Semantic objects – Visual representations of document model objects and their attributes (a specific range of cells in a spreadsheet, a paragraph in a text document or their properties dialog). Often, a presenter would like to specifically mark these objects as confidential or private (one of the difficulties is that an object may have more than one visual representation even at the same time)

Peripheral data – Sensitive data that is not part of the object model of the shared document and appears in the application's UI as a byproduct of the presenter's interactions (recent files, browser navigation history, auto-complete text boxes).

Many applications introduce personalized convenience features that cache users' preferences and selections and appear without explicit action on the part of the user. Other applications couple sensitive and non-sensitive UI controls (a settings dialog that contains both Color and Security settings)

Interactions – Some of the interactions a presenter makes may be deemed private because they affect his reflected image, regardless of the data they operate on. Some examples are committing syntax errors or other mistakes, searching for the right menu item, or struggling with a wizard.

Some exposures of private elements are an immediate outcome of the presenter's direct manipulations and fit well within the presenter's mental model of the application. These may be avoided or bypassed by the presenter at the price of forcing clumsier interactions or more careful preparation ahead of time. Other exposures are byproducts of agents that work on behalf of the user (e.g. an error message or the contents of an auto-complete text widget). These are less predictable and require more automated help to avoid accidental exposure. In either case it is disclosure to viewers we need to control, not the appearance or content of these elements.

It may seem that some of the private interactions described above are brief, so the amount of information viewers can extract is limited. However, it is common for shared sessions to be recorded, allowing later analysis (ephemeral information becoming persistent [11]). It has also been shown that viewers are more likely to notice sensitive text on a large-screen public display, [15].

IMPROVING VIEWER EXPERIENCE

In our scenario passive viewers follow Bob's interactions. We can customize each person's view by adding, deleting, or modifying the application's presentation (bitmap) to provide a more useful experience.

Inadequacy of single-user GUIs for passive viewers

Passive viewers must follow the interactions performed by the presenter, but there is a perceptual gulf between presenter and viewers. While the presenter translates her intentions and semantic-level operations into GUI interactions, the passive viewers are doing the reverse process, inferring the underlying intentions and semantics from the interactions. This is not an easy process, even when verbal explanations are provided by the presenter (these are usually insufficient, inconsistent, and they require extra effort on the presenter's part).

One of the root problems is that the visual language of most GUIs is highly tuned for a single active user, ignoring the needs of passive viewers. For example, when a presenter decides to perform a contextual menu selection, the cue for a passive viewer that some interaction is about to take place is the appearance of the menu, by which time it already obscures most of the context for the operation. Another problem is that passive viewers tend to follow the presenter's point of interaction (often highlighted in GUIs), yet in some cases the presenter would like to draw their attention to other regions.

Other low-level parameters such as cursor size or shape, the time a menu or dialog remain on-screen after release, or interaction without a specific visual indication (e.g. keyboard shortcuts), are tuned for the performance of a single active user. These are not suitable for passive viewers, who are trying to follow the interactions without the benefit of knowing the intention of the action or experiencing the kinesthetic feedback of mouse or keyboard interaction.

Controlling verbosity

Viewers may have different levels of expertise and familiarity with a shared application. It is beneficial to adapt their view to this level. A key aspect to be controlled is the *verbosity* of interactions. For example, if Bob is to teach Ted and Alice how to fill out a report using an application unfamiliar to them, exposing the fine details of his interactions (menu selections, dialog boxes, etc.) and adding cues (like keyboard shortcuts and change highlighting) could be crucial. On the other hand if Ted and Alice are experienced users, exposing detailed interactions will prevent them from concentrating on the report semantics. From a pedagogical point of view, it sometimes makes more sense to show one logical interaction unit as a single visual step so the high-level semantics are not obscured by the low-level details.

Mitigating visual clutter

We have already implied that it is desirable to share only relevant windows or components, rather than the entire desktop. This can assist viewers in making better use of their screen space (especially if they work with or view other applications in parallel or multiple users bring up applications on a shared display [16]). However, if all sub-windows, dialog boxes and menus are shared as well, it can quickly clutter a viewer's display. This is somewhat like violating acoustical privacy with cellular phones [11]. The presenter imposes his "conversation" with the application on the viewers, much like a person talking on a cellular phone imposes on others in a public area. A viewer should be capable of controlling how much of this conversation penetrates his display and replace some interactions with other "low volume" representations. In all cases viewers need to maintain some level of awareness of the presenter's actions, but not always in a one-to-one manner.

SYSTEM DESCRIPTION

We have implemented a prototype of the system in C# on Windows and tested it with three widespread commercial applications (MS Excel, Word, and Internet Explorer). The principles apply to any modern operating system and they work with any application, although some "semantic glue" layers that we describe later may be required.

Cloning Windows

In order to support differentiated views, the system grabs the visual surface of shared application windows on the presenter's machine and conveys a manipulated version of the bitmap to the public display (published in *clone windows*). This can be done using a modified version of the RFB protocol [12], adapted to work on separate windows. In the prototype, we simply relied on timer-based device context copying (similar to MSR's Wincuts [16]), which matched our initial focus on co-located scenarios (where all displays are on a single machine). In both solutions the viewer client is a simple image buffer player (Figure 2).

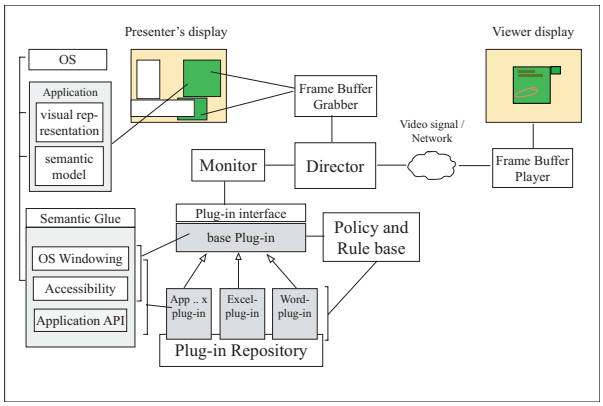

Figure 2: System architecture

We use an extended desktop mode. The public display is a continuation of the presenter's desktop (although often physically located on a wall behind the presenter). A clone of each of the shared application's windows is created by querying the system's list of windows and making bitmap copies. The clones are automatically placed on the part of the desktop lying on the public display. The presenter can move any of the application's windows on her display and cover these with other windows without affecting the published clones.[2] The novelty is in how we modify the bitmap images and window set before they are placed on the public display.

Applying "semantic glue"

To alter the shared view along the lines discussed in previous sections, the system needs to monitor a shared application. It should be able to tell where visual representations of private elements or elements that need verbosity adjustments are on the visual surface and determine if the application is in a private state. This requires methods for obtaining information about the application's GUI components, the underlying semantic objects, and their visual representations. The following query layers are used.

L1: OS windowing queries – Enumerating all windows belonging to a specific application (or process), detecting creation/destruction of such windows, visibility, titles and locations is possible in a modern operating system. Many of the widgets used in an application are themselves windows and can be accessed the same way. This layer also supports capturing of keyboard and mouse events.

L2: Accessibility API – These are common APIs often targeted to sight-impaired users. They enable third-party tools, such as screen readers, to systematically expose information about UI elements. We have successfully re-purposed these APIs as a resource to expose elements that should be kept private or highlighted (e.g. UI widgets, menus and menu items). These APIs are supported by many commercial tools and UI toolkits.[3] Our use of them can be further generalized.

[2] See [16] for more details.

[3] Supported by Microsoft Accessibility, Java APIs, OS X and more. Some level of accessibility is now required by law and will increase over time.

L3: Application specific API – Many commercial applications provide an API for integration and automation. These APIs can be used via COM (Windows), JavaBeans, AppleScript and other frameworks. Within our system, we used these APIs in a simple manner to extract information on the application's state and identify the visual representations of semantic objects. While writing some code to work with the API is required, our experience when developing the prototype shows that this is a focused effort with a limited amount of coding. Modern APIs already provide methods for locating an object on the document surface or an Application object can usually be queried for its current state. Furthermore, coding occurs only once and can then be used in flexible ways.

L4: Extracting information from surface drawing operations – This is a technique introduced in [10] (see the related work section). Its requirements are quite problematic, especially for the commercial tools we worked with, therefore this technique was not used and it is not shown in Figure 2. However, it is still a possible semantic glue layer that could be used in some cases.

To create a generalizable framework, we chose to provide all of these methods using a plug-in architecture for our system. Each shared application has a middleware plug-in that functions as the semantic glue. A plug-in encapsulates the knowledge about a specific application and its monitoring, and supports a common API that the Monitor module (Figure 2) can use. The PAPI (Plug-In API) provides methods that return text or keyword-based descriptions of the current state, dialog or menu. Other methods extract lists of areas of the visual surface containing private information that need highlighting or specific sub-window areas to be displayed (instead of the full window). Each of the areas are accompanied by keyword descriptions that can be used to control display parameters. A plug-in translates these general PAPI queries into appropriate queries in one of the four layers.

A default *base plug-in* provides a set of general capabilities to track common UI entities. It serves as a toolbox for developing more specifically tailored plug-ins. Namely it runs a background service that searches for dialog boxes, menus and other widgets (like dropdown boxes) of a shared application by tracking window creation events. It then uses L1 calls to extract their window class (type), title and location and the accessible object associated with the window. L2 calls are then used to extract accessible name, role, text, state, selection and location for the object and its child elements (e.g menu items or dialog fields) on the "accessibility tree". Hints on how to handle these items can then be obtained by testing this information against keywords specified in an application-specific plug-in without additional coding.

For example, our Excel plug-in defines the keywords "open", "save as" and "options" in a dialog title as private. When the base plug-in finds these in a dialog a private state will be issued and the dialogs will not be exposed. The keyword path "Format Cells/ Protection" will tell the base plug-in that the tab widget titled "Protection" in the "Format Cells" dialog is private and its entire contents should be blurred.

In addition, a path-regexp "File/(^.*\.xls$)" orders all items in the "File" menu that match a file pattern to be blurred (i.e. recent files).

To get regions for blurring or highlighting, Excel-specific API calls (L3) are used to locate cell ranges marked in a specific background color, the selected cell range and its surrounding table, or changed cells. The on-screen bounding box is then computed (an Excel.Range object provides color, bounding box and value properties that are simple to use and an Application object has methods to get selection and changed ranges). If the selection is in a private cell range, additional L1 and L2 API calls are used to locate the formula edit box or locate the sub-window frame containing the document (these can be identified by their window class or name and position in the window tree). Once a bounding box is obtained the base plug-in provides generic blur and highlighting filters that work on any image buffer.

Our web browser plug-in defines the keyword "favorites" in a menu as private. It also defines keywords matching the navigation history dropdown and auto-complete box (their window class, accessibility name and role) as private. Thus when any of these is opened and identified by the base plug-in, a private state will be issued and they will not be echoed on the public screen. We targeted our plug-in to Internet-Explorer, but a similar plug-in for a different browser only needs to replace the keywords with the appropriate names (e.g. "bookmarks" instead of "favorites").

The Monitor module directs its calls to a plug-in repository manager, which loads the appropriate application plug-in at run time (possibly even from remote servers). If no appropriate plug-in exists the default base plug-in will be used, offering some monitoring capabilities. (It could query the presenter before displaying any menu or dialog box, and then apply "program by example" techniques.)

The Director module handles the published representation to be played on the public display. It uses the Monitor module to track the application and extract descriptions of its state and visible elements. It then applies policies that determine how to manipulate the visuals.

Policies and rules

When instantiating a policy, a tuple comprising the application, the state or element, and the viewer's role is the input. The output is a rule that determines what manipulations will be applied to the published visual representations.

We must determine how private elements, states or elements that need verbosity control can be extracted, assuming shared applications do not know about privacy. There are two complementary approaches to consider.

The first approach (taken in the initial prototype) is letting the presenter mark these elements explicitly. When working with a document in an editor we can readily support what we call a *"Magic Marker"* that maps a visual property of an object to a privacy state (most editors have a notion of object style properties). For example, a presenter can mark a document object as private by coloring it with a specific color, using the native application tools (e.g. a background color

for cells in Excel or a highlight color for paragraphs in Word as shown in Figure 3). When writing in this color the semantic glue layer will recognize these objects as private. A policy that regulates blurring for marked objects will create the effect of a marker that cannot be seen by viewers, while the presenter can interact normally (as opposed to using black on black writing that will not be visible to the presenter). This mode provides visual feedback and awareness on what the audience cannot see (demonstrated in [13] to be useful).

Other means for coding attributes can also be used (like adding a "Private" prefix for a worksheet's name or comment text). Another option is to use the application's built-in selection mechanism, so for example the paragraph containing the insertion point can extracted by querying the application.

A second approach is to use a rule-guided search for privacy *leaks*. We have experimented with searching for private text in a spreadsheet or document (phone numbers, names, etc.) and automatically blurring them. Another interesting domain are web pages, where we partially implemented a search of the HTML code for private UI widgets and content elements (e.g. examining all form field names and blurring ones that may contain private information, such as userids, credit card number, etc.).

We already described the keyword based services of the base plug-in. These can be extended to conduct rule-guided search for private information in any menu, dialog or wizard (e.g. search for error messages, field names related to security, network settings or personal information)

We believe that a combination of these approaches is required for better privacy protection. Together with the visual manipulations (described later), it allows a flexible range of rules: "do not expose any dialog related to files or the network in any application to any public viewer," "blur any document element in any application marked in pink to group A members" or "if there is any viewer from group B, do not expose in a public view any web page not coming from company servers." Also useful is an inclusive policy of "blur everything unless specifically marked by the presenter" (in the prototype we worked with exclusive policies).

Our prototype is a work-in-progress that serves as a proof-of-concept. We have concentrated on the system architecture and a collection of manipulations (discussed in the next section) and illustrated the value of having role-based policies to control these. Our goal has not been to develop a robust mechanism for describing policies. In fact, earlier work on access control in collaborative systems (see survey in [18]) already suggested some schemes. We intend to adapt these schemes to control viewing rather than control access. We will discuss a potential extension of our system that can apply access control to collaboration-unaware applications.

MANIPULATING THE VISUAL REPRESENTATION

The Director component takes in the "raw" captured frame buffers grabbed from the application windows and applies one or more of the following manipulations.

Blurring

When private elements are visible, the challenge is guaranteeing viewers cannot see them, while the presenter works freely. The PAPI can extract the locations of such elements on the visual surface at any time (with attributes and hints, such as the suggested blur effect to use). In some cases a private information unit may appear in several places (e.g. the contents of a selected private spreadsheet cell will also appear in the formula bar). This demonstrates why tighter integration with application semantics is crucial for ensuring privacy.

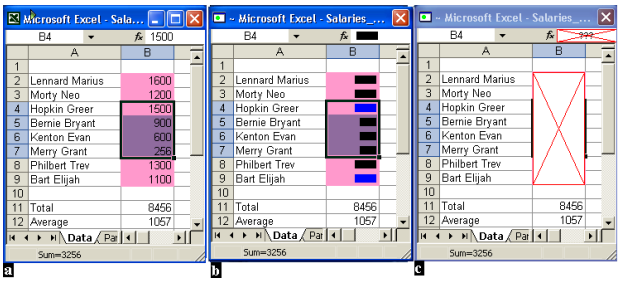

Figure 3: (a) The presenter view of a spreadsheet, (b) Greeking cells marked in pink, exposing selection and style, and (c) fully concealing a cell range.

The Director can apply several image blurring operators on extracted private zones (Figures 3 and 4). Since blurring occurs at the frame buffer level it can be applied regardless of what the underlying element is (UI control, text, image etc.) or how the bitmap was drawn. Different filters offer different visual affordances, balancing between the presenter's privacy and the audience's awareness.

- **Draw over** – Invoked for full privacy, with no awareness.
- **Greekify** – Creates a "Greeked text" effect by searching text line boundaries on the image and replacing them with filled rectangles. Useful for exposing structure, style and some notion of the presenter's interactions (such as selection).
- **Pixelize** – This is a general purpose filter, mostly useful for image-based content. It provides awareness cues for viewers, but may be insufficient for full privacy.

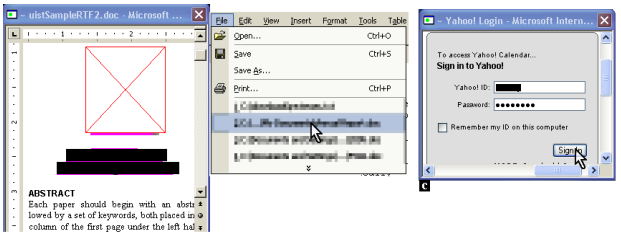

Figure 4: (a) Word document with blurred paragraphs and image, (b) recent files menu items blurred, and (c) login web page where the userid field was detected and Greeked.

Salience and highlighting

The system supports a highlighting mechanism that is independent of the shared application's own selection and highlighting tools. In Figure 5a the presenter is interacting with the tools palette, but wants to keep viewers focused on a specific paragraph. The PAPI provides a method, through which

the shared application can be queried for regions to highlight. We found it useful to highlight the context for the active selection as changes are made. This is application-dependent (the paragraph, sentence or section containing the insertion point in Word, the table surrounding the selected cells or dependent cells in Excel and the active dialog field). The highlighting effect is application-independent and works on the image buffer by placing a semi-transparent colored mask on top of non-highlight areas (visually similar to [10]). Detaching the highlighted object from selection is also useful and can be done by caching the previous highlighting bounds or by caching a pointer to the previously selected object within the plug-in.

Another mode of highlighting makes changes more salient to viewers (mostly indirect changes in parts of the visual surface far from the presenter's interaction). A considerate presenter would point out these changes to an audience and perhaps even mark them on the screen. To assist the presenter, the semantic glue layer can extract such changes and provide automatically generated highlighting (Figure 5c shows hand-style circling of changed cells that also exposes changes in blurred data).

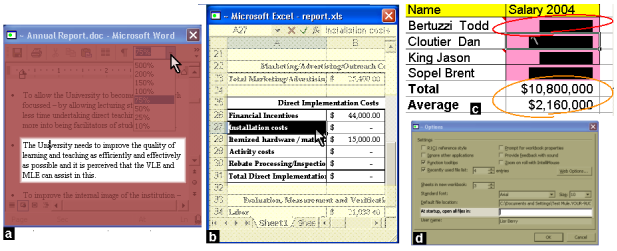

Figure 5: Auto-highlighting of active context: (a) active paragraph is highlighted, (b) outlining the table surrounding selection,(c) "by-hand" style circling of changed and dependent cells, and (d) highlighting active dialog field.

Other manipulations that can affect salience and attention are magnifying relevant regions of the visual surface or even re-rendering of textual elements in a bigger font (many of these underlying texts can be extracted through the semantic glue layer).

Spatial manipulations

One set of manipulations allows the presenter to share only a partial view of an application's window. This is useful for reducing screen space use and clutter, and in addressing privacy. The PAPI provides a method through which the window part to be shared can be accessed. Computing this window part can take into account several policies.

- **Excluding the UI** – Remove UI layers such as toolbars and embedded windows that take a substantial amount of screen space (a plug-in will use L1 and L2 calls to search the application window tree)
- **Active context** – Share only the active context, based on the presenter's selection as described in the previous section.
- **Sharing a specific element** – Sharing only a specific semantic object (paragraph, table) or UI element chosen by the presenter.

The semantic layer guarantees that the window part computed will adhere to the stated policy and will take into account changes to the window's dimension, scrolling or UI changes, as opposed to the manual definitions presented in [16] or to fixing a portion of the screen to be shared.

Another set of manipulations uses affine transformations. Rotating windows is useful for single tabletop display sessions, where viewer orientation should be part of the policy. Automatically scaling down the size of dialog boxes, palettes and other secondary windows (identified by PAPI), together with a careful placement of these next to the full sized main window can assist in reducing clutter and support privacy (Figure 6c). Combining several spatial transformations together can be quite powerful. For example, it is possible to publish only the selected paragraph context, flipped vertically so a viewer on the opposite side of a table can follow the discussion without requiring replication of the full document window(Figure 6a).

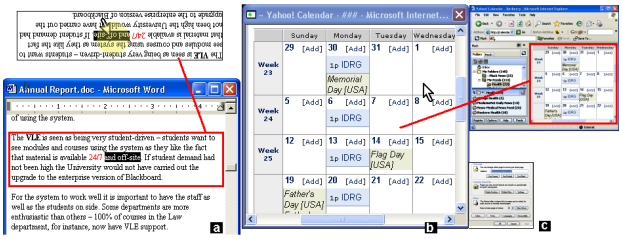

Figure 6: Spatial manipulations: (a) publishing only active context (paragraph containing insertion point) + vertical flip, (b) auto-exclusion of toolbars, menus and embedded frames from an Explorer window, and (c) automatically downsizing an "options" dialog.

Temporal manipulations

There are situations when it is more reasonable to define the entire state of the application as private, rather then extensively applying blurring transformations (e.g. when the presenter interacts with a private worksheet, uses the file open dialog, or works with a wizard).

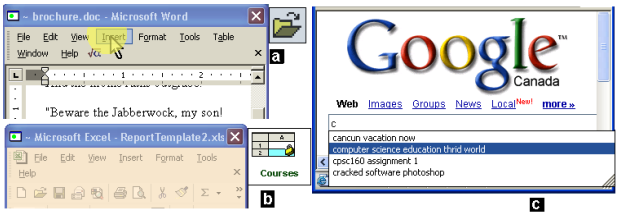

Figure 7: (a) File open dialog is dynamically replaced with an iconic representation. (b) A private worksheet replaced with an iconic indicator, and (c) an auto-complete text box will be detected and will not be exposed.

The semantic glue layer can query the application state. If it matches the privacy policy, the system can trim the interaction timeline by not sending updates to the viewer's display until the presenter exits the private state. To keep some level of awareness for the viewer, the system can display an animated iconic representation summarizing the state. For example, an open file icon appears instead of exposing the

dialog itself (Figure 7a) to prepare a viewer for a document change or an icon indicating interactions on a private worksheet gives the viewer hints about what the presenter is doing.

In other situations, it is better to not provide any indication at all, maintaining complete privacy. For example, when the presenter interacts with an auto-complete text box (Figure 7c), an error dialog, a private comment or commits syntax errors when taking notes in public. Some of these states are quite unpredictable by the presenter (e.g. an error dialog popping up or auto-complete suggestions), so automatic detection of these is crucial.

Another set of timeline manipulations can be applied to the pace at which certain operations (as extracted by PAPI) are played on the public display or by letting viewers roll back recorded interactions (for instance using the semantic glue to tag recorded interactions for quick roll-back). One example we implemented involves menu selections, discussed in the next section.

Handling Menus

Menus are fundamental interaction components that are often problematic in a generalized presentation scenario. They create a lot of clutter (being arbitrarily long, regardless of the size of their parent window), and they often bundle private information (recent files, bookmarks). More importantly, menus are becoming highly tuned for the active user and less for a passive viewer (adaptive menus with personalized order and gestural menus that do not show on the screen). As described previously the base plug-in obtains the relevant menu attributes (name, items, locations, selection) from any menu through L2 calls and can issue blurring on specific items or prevent a menu from showing on the public display (these are highly generalizable techniques)

A critical moment is when the presenter makes a selection on a menu. From her point of view there is no need for the menu anymore and it is taken away. The system, however, can capture the selection event and pause the timeline so that the menu lingers on the public screen for an extended "decay" period suitable for passive viewers (possibly with animated highlighting of the selected menu item). It is also possible to identify the creation of a context menu, and smoothly move its clone to a neutral placement that does not block the operation's context (Figure 8a). We also used these techniques to stall a dialog on the public screen after the presenter closed it.

Watching interactions with menus is not the best way to convey operations to viewers. In many cases, replacing a menu selection with a different feedback, such as specifying the selection in a semi-transparent subtitle is better (Figure 8b). Consider a presenter who scrolls through a menu until finding a specific item. It is hard for a viewer to tell if the menu was closed because a selection was made or the menu was released with the ESC key. On the other hand, the subtitle scheme reports only when a selection is actually made. This scheme still works if the presenter uses keyboard shortcuts or gestural menus because the semantic glue translates these back to a menu item description for display.

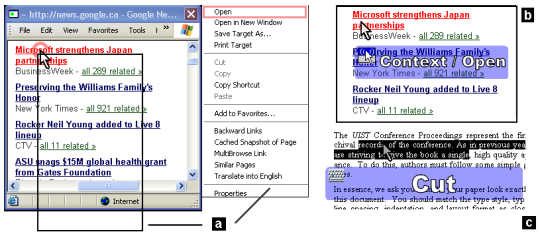

Figure 8: Manipulating menus: (a) moving a context menu to the side + highlighting selection after release, (b) replacing a menu selection with a subtitle below the mouse cursor, and (c) reporting a keyboard shortcut command

Access Control

Tools like [12], [8] and remote desktop solutions enable the presenter to give a viewer full control over keyboard and mouse input to her machine (coarse floor control). This is highly undesirable, since a viewer may make changes to non-shared applications or make unwanted changes to the application shared. We integrated the basic functionality of PointRight into our system, yet we are able to use the semantic layer to identify locations of widgets, menus and controls that should not be accessed. Thus when a viewer sends a mouse click event on such a control, the system will not pass this event to the OS. Similar treatment can be applied to keyboard events (this requires more queries on the application). This scheme allows finer grained access control policies to be applied to applications that were not designed for it.

RELATED WORK

There is a large corpus of research on collaboration-aware tools and frameworks that allow custom view sharing, including commercial tools (see a survey in [3] or [17]). Our system is targeted at existing collaboration transparent single-user tools, that were not designed for multiple views and cannot accommodate code changes. In this context, it is useful to classify these based on a mapping of the sharing architecture space. The Zipper model, presented in [4], looks at the common layers comprising an application: User, Screen, Window, Widget, View, Model to determines the share branching level. In each architecture one of these layers is the branching point. All layers below it are shared and all layers above it are replicated.

Closest to our approach are tools that replicate the screen or window layers (known as centralized tools), such as VNC [12], NetMeeting[4] or XTV [1]. VNC shares the entire screen, including windows of applications a presenter might like to keep private. NetMeeting and XTV share all windows of a chosen application, but lack the ability to keep some dialogs or palettes private. In terms of supporting different views, NetMeeting allows a presenter to manually "pause and play" the view sharing. It requires the presenter to identify privacy concerns, some of which are unpredictable, while he is "on-the-air" and is therefore error-prone. In our solution the presenter specifies rules for controlling these private elements in advance. All of these tools lack any ability to systematically change the contents of the replicated screen parts and therefore cannot handle sub-window elements or provide au-

[4]http://www.microsoft.com/windows/netmeeting/

tomated highlighting. Some tools do allow manual highlighting or sketching on the application surface that will become invalid once the window is scrolled or resized.

Flexible JAMM [3] is a framework for transparently replicating the widget layer. It is based on replacing some widgets and components of existing single-user applications with multi-user versions that can be synchronized with some degree of differentiation. This approach puts different constraints on the running environment and underlying code that make it unsuitable for commercial off-the-shelf applications.

Recently some research efforts have focused on synchronizing two running single-user applications (i.e. replicating the Model layer), using operational transformation techniques. A prominent example, working with an off-the-shelf editor (MS Word) is CoWord [17]. Each user has independent control of her copy while the system synchronizes the underlying document models using the application's API (typically without support for private document parts). This solution assumes complete independence of views, which is not suitable for presentation scenarios. Still, with some effort views can be synchronized as well (replicating the View layer).

The major drawback of JAMM, CoWord and other replication-based solutions is that both presenter and viewer need a copy of the application (or two instances running on a single computer) which is not always possible. These solutions also need to synchronize the application replications, which can be a hard problem [4], requiring resource locking or forcing expensive calculations to be carried out multiple times.

Wincuts [16] is a bitmap-based system that provides some spatial manipulation of shared windows. It allows a presenter to manually select a region of the window and publish only that part. Similar ideas are presented in regards to window layouts and screen space use in [7]. While still allowing a lot of flexibility and addressing clutter and privacy issues, this approach quickly breaks down when a presenter needs to resize or scroll a window. Our system completely subsumes these approaches, automating spatial manipulations and blending them with other filters.

A limited set of manipulations to the application's visual surface were presented in [10]. Some of the manipulations are similar to ours (e.g. region highlighting), but they are entirely targeted at supporting the work of a single user. Our system demonstrates how these can be used in a multi-user shared view scenario. On the technical side, this approach requires overriding some of the low-level drawing routines and consistent ordering and grouping of component drawings; it cannot reason about information that does not go through the display pipeline (like field names). These demands are not fully met by off-the-shelf tools and are quite hard to support under some operating system (Windows). Still, this may be a viable way for performing some of the semantic glue operations and other manipulations. Our plug-in architecture allows us to incorporate these techniques into our system.

In regards to privacy and clutter, Pebbles [9] offers to replicate some application components on a handheld device. Thus a presenter may choose to conduct some interactions on her handheld or auxiliary computer to avoid exposure. This approach requires extra hardware to be present and it does not address viewer needs for awareness cues. Most importantly this approach is limited in the type and complexity of components that can be recreated on the handheld (text fields and menu work well, but part of a worksheet relying on other spreadsheet parts may not).

Research into single display privacyware has resulted in several platforms enabling each user to have a different view of a shared display [13]. However, these systems still require running software that is capable of supporting differentiated views, so collaboration-aware tools must be used. Our solution enables using such systems with off-the-shelf applications.

Finally, commercial presentation authoring and playback tools such as Microsoft PowerPoint or Apple Keynote have recently taken advantage of multi-display technology to play the presentation on a public screen, while providing a private view to the presenter (where she can view her notes or check other slides). Our approach can provide similar advantages to other single-user applications.

DISCUSSION AND FUTURE WORK

Our prototype showed that the proposed techniques, especially relying on Accessibility APIs, can work well with off-the-shelf applications. Some limitations of our approach and required improvements were also evident.

Private information is handled only on the visual surface level and not in the underlying document model level. This implies that tracking and handling all cases where private information can leak is not always trivial and is bound by the capabilities of the application's API and the quality and integrity of the Accessibility information or requires extra marking effort from the presenter. Furthermore, blurring or eliminating private information on the public display can still expose some private properties of the data (the Greeking effect in Figure 3c exposes orders of magnitude and concealing information is itself an indication on the nature of the data).

We suggested some basic automated extraction of private elements, states, and augmentation hints, but clearly more work is required. One possible direction is applying user modelling and machine learning techniques to learn what elements are considered private by a presenter and apply these to search the widget space and document model. Another direction is harnessing "programming by example" techniques.

There is an innate limitation on the public display update rate, since we copy image buffers. In theory performance should match that of VNC. The overhead of adding semantic glue queries and image buffer filters can slow down the update rate. Our experience showed these were sometime noticeable but reasonable. While image filters were quite efficient, it is clear that some of the APIs we use were not designed with performance in mind or even to work in all application states (API calls varied in execution time between application states and even resulted in errors in some cases). It may be required to synchronize the semantic queries and the image buffer captures more carefully.

We intend to replace the inefficient timer-based updates with a paint event based solution to improve performance and also support sharing of image buffers over the network. There are different possible models of where blurring and other window manipulations occur (presenter's machine, viewer's machine, a trusted third party server) that affect performance, privacy and security. We could also enable the sharing of more than one public view (a limitation of the current implementation).

We will run a formal user study on the next version of the system to evaluate its benefit to viewers and presenters. To date we have verified the problems identified through informal studies and have tested the framework with several collaboration practitioners.

CONCLUSIONS

We have introduced a unified solution for privacy concerns and verbosity control to assist a presenter and her audience in generalized presentation scenarios. These growing concerns are not addressed by current single-user application sharing modes. Our design introduces role-driven views for each type of participant, balancing between the presenter's privacy needs and the audience's awareness needs. Our system is based on applying image filters and spatial and timeline manipulations to bitmap representations of shared windows, in contrast to similar techniques that have been recently shown to be unsuitable for addressing privacy in video [5].

The system's framework is general and works with off-the-shelf applications, requiring only a limited glue layer. A prototype of the system was created and tested with several commercial applications. As part of our work we discovered how Accessibility APIs can be leveraged to address privacy concerns. Our system allows additional intermediate sharing layers to be introduced through an extensible plug-in architecture beyond the conventional screen, application or window layers of the Zipper model [4].

The system improves the quality of generalized presentation sessions. It protects a presenter from exposing private information and elements, allowing her to work normally and comfortably. It assists viewers in maintaining a suitable level of awareness and in better understanding the presenter's intentions.

ACKNOWLEDGEMENTS

The authors thank the Natural Sciences & Engineering Research Council of Canada and NECTAR for their financial support, and Maureen Stone for providing interesting insights and suggestions.

REFERENCES

1. Abdel-Wahab, H. M., and Feit, M., XTV: A framework for sharing X Window clients in remote synchronous collaboration. In Proc. of IEEE Conference on Communications Software, 1991.

2. Baudisch, P., Cutrell, E., and Robertson, G., High-Density Cursor: A visualization technique that helps users keep track of fast-moving mouse cursors. Proc. of Interact 2003, 236–243.

3. Begole, J., Rosson, M. B., and Shaffer, C. A., Flexible collaboration transparency: Supporting worker independence in replicated application sharing systems. ACM Trans. Comput.-Hum. Interact., 6(2), pages 95-132, 1999.

4. Dewan, P., Architectures for collaborative applications. In Trends in Software: Computer Supported Cooperative Work, pages 169-193, 1999.

5. Greenberg, S., Neustaedter, C., and Boyle, M., Blur filtration fails to preserve privacy for home-based video conferencing. ACM Trans. Comput.-Hum. Interact., 2005.

6. Hawkey K., and Inkpen K. M., Privacy Gradients: Exploring ways to manage incidental information during co-located collaboration. In Proc. of CHI 2005, 1431–1434.

7. Hutchings D. R., Stasko, J., Revisiting display space management: Understanding current practice to inform next-generation design. Proc. of GI 2004, 127-134.

8. Johanson B., Hutchins G., Winograd T., and Stone M., PointRight: experience with flexible input redirection in interactive workspaces. Proc of UIST 2002, 227- 234.

9. Myers B. A., Peck C. H., Nichols J., Kong D., and Miller, R., Interacting at a distance using semantic snarfing. In Proc. of UbiComp 2001, pages 305-314.

10. Olsen, D. R., Hudson, S. E., Verratti T., Heiner J. M., and Phelps M., Implementing interface attachments based on surface representations. In Proc. of CHI 1999, 191-198.

11. Palen L., and Dourish P., Unpacking privacy for a networked world. In Proc. of CHI 2003, 129-136.

12. Richardson, T., Stafford-Fraser, Q., Wood, K.W., and Hopper, A., Virtual Network Computing IEEE Internet Computing. 2(1), pages 33–38. 1998.

13. Shoemaker G. B. D., and Inkpen K. M., Single display privacyware: augmenting public displays with private information. Proc of CHI 2001. 522-529.

14. Stefik, M., Bobrow, D. G., Foster, G., Lanning S., and Tatar, D., WYSIWIS revised: early experiences with multiuser interfaces. ACM Trans. Inf. Systems 5 (2), pages 147–167. 1987.

15. Tan, D. S., Czerwinski M., Information voyeurism: social impact of physically large displays on information privacy. In Proc. of CHI 2003, 748-749.

16. Tan D. S., Meyers B., Czerwinski, M., Wincuts: manipulating arbitrary window regions for more effective use of screen space. In Proc. of CHI 2004, 1525-1528.

17. Xia S., Sun D., Sun C., Chen D., and Shen H., Leveraging single-user applications for multi-user collaboration: the CoWord approach. In Proc. of CSCW 2004, 162-171.

18. Tolone W., Ahn, G.-J., and Pai T., Access Control in Collaborative Systems. ACM Comput. Surv. Vol. 37, No. 1, 29–41, 2005.