

# Privacy-aware shared UI toolkit for nomadic environments

Richard B. Arthur<sup>\*,†</sup> and Dan R. Olsen Jr

Brigham Young University, Provo, UT 84602-6576, U.S.A.

## SUMMARY

As computing becomes more nomadic, privacy becomes a greater concern. People use portable devices to annex displays in their environments so that they can share information with other people. However, private information such as usernames, e-mail addresses, and folder names are shown on foreign displays. In addition, foreign keyboards can be used to enter in passwords generating a significant privacy and security risk. Because nomadic users' sensitive data is constantly at risk for exploitation via the UI toolkit, a solution for protecting user privacy must include that toolkit. This paper introduces the XICE framework—a windowing toolkit that provides easy display annexing and includes a robust privacy framework to help protect users and their data. This paper discusses the exploits that annexing external devices introduces and how XICE mitigates or eliminates those threats safely and naturally for both users and developers. Copyright © 2011 John Wiley & Sons, Ltd.

Received 29 October 2010; Revised 9 March 2011; Accepted 17 March 2011

KEY WORDS: SPICE; privacy-aware; toolkit

## 1. INTRODUCTION

Computing is increasingly nomadic. Nomadic computing allows people to collaborate with each other in a wide variety of places and situations. A key feature for a nomadic user is the ability to have his own data, settings, and applications available wherever he is located.

A user could carry his data, settings, and applications via a *personal device* (e.g. laptop) so that he can interact with his data anywhere. However, portable devices tend to have constraints on size and weight and consequently on processing power and interactive richness. If the user can use his personal device to annex resources (e.g. projectors or desktop monitors) in his environment then he can overcome many of the limits of his personal device.

Normally annexation is performed via a VGA or DVI cable. Digital connections via wireless networks are more versatile, because the annexed devices may additionally supply input hardware and support multiple users. This paper contends that such network devices will replace VGA or DVI. The network-enabled, annexable computers are called *shared devices* and typically provide a screen but may also supply input devices such as keyboards and mice. Some of the common devices users might share are illustrated in Figure 1. Notice that some shared devices are public and not trusted while some shared devices are private and trusted. User Interface (UI) software must be able to tell the difference and present itself accordingly.

Key to annexing shared devices is distributing the UI from an application executing on the user's personal device to a shared device. There are several existing technologies that may be used to distribute a UI, such as X-Windows (X11) [1], Remote Desktop Protocol (RDP) [2], or Virtual Network Computing (VNC) [3]. These protocols transmit the output from the executing computer

\*Correspondence to: Richard B. Arthur, Brigham Young University, Provo, UT 84602-6576, U.S.A.

†E-mail: startether@startether.com

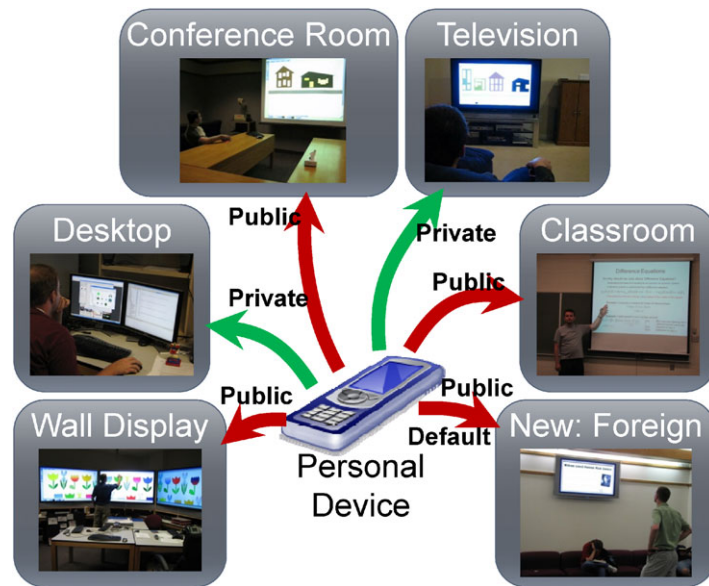


Figure 1. Shared devices that a nomadic user could annex. These devices are located in either public or private environments.

to the shared device (sometimes called a *display server*) and transmit input from the shared device back to the executing computer.

This network UI architecture introduces several privacy-related challenges. If a user annexes a shared device, that device has several potential attack vectors: stolen output, stolen input, false output, and false input. The shared device may have *crimeware* [4] installed, which could steal the output from the user's applications including any sensitive data shown in those applications (e.g. e-mail addresses or login names). If the user enters his username and password through the annexed input hardware, the shared device may steal those credentials. In addition, the shared device may falsify output or input in an attempt to compromise the user's device or expose his sensitive information.

Fortunately, because the user has a portable trusted personal device, there are some new privacy-related opportunities. The personal device may be used to view and input *sensitive data* (any data that should not be viewed by arbitrary users), whereas the shared device may be used to display *open data* (data that can be viewed by anyone). Most current privacy-aware software operates exclusively on one screen which it must treat as a *public screen* (viewable by anyone) or *private screen* (viewable by trusted parties). Allowing software to simultaneously operate on a public and a private screen opens many new issues for privacy-aware UI tools.

In addition, the input from a shared device may be trusted or distrusted. *Trusted input* is input that the user (or someone the user trusts) has control over to prevent malware from stealing input or providing false input. *Distrusted input* is input from some foreign device which the user does not control. The term 'distrusted input' may synonymously be considered 'untrusted input.' This paper uses the term 'distrusted input' throughout.

Each device a user interacts with has a different *privacy state*—a combination of whether the display is public or private and whether the input is trusted or distrusted. With nomadic interaction, the portable device is assumed to be private and trusted. The user's desktop machine at work or home would be a shared device that is private and trusted. However, annexing a coworker's desktop display may be public and trusted; the user does not want IMs to appear on the coworker's display. In a conference room at the user's work the display is public, so sensitive data should not be shown, but any other open data should be shown. Because the shared device is controlled by the user's work, its input is trusted so the user can confidently access or expose sensitive data

as necessary. When presenting at another institution, however, the shared device is public and distrusted; hence, sensitive data should not be shown or altered. Shared devices in other locations such as restaurants or mall kiosks would also be public and distrusted.

Even though input may be distrusted, the user is not precluded from accepting input from the annexed device. For instance, a user may choose to annex a foreign display so that he can type up an e-mail consisting of open data because typing via a physical keyboard is faster than via his mobile device's soft keyboard. If a user chooses to accept input from a distrusted device then his software must actively protect him from potentially malicious activity. For example, the software must be aware of any input that may attempt to alter or expose sensitive data. Continuing with the e-mail example, the user should pick the recipients via his mobile device, and the e-mail addresses should be blocked on the public display (although the contact name may be shown). A malicious display server could attempt to expose the user's contact list on the public display so that it could steal the available e-mail addresses. The contact list exposure instruction should be confirmed in a non-exploitable way—for example via a dialog on the personal device—so that the e-mail application knows that the user initiated the contact list exposure. A worse violation would be if a malicious display could generate and send spam e-mails via the user's e-mail application. Consequently, the 'send e-mail' command should similarly be confirmed.

To enable privacy-aware development, application developers must be able to write software which changes its output and can filter any input based on the shared device's privacy state. X11 and RDP inform software when the UI is rendered on a shared device, while VNC does not. However, none of these protocols attaches a privacy state to the network connection; all shared devices are treated as private and trusted. RDP allows for excluding the shared device's input, but accepting and distrusting input is not possible. A shared UI protocol must be able to identify the privacy state of a shared device so that shared applications may alter their output and filter the input.

When using existing UI distribution technologies, developers have two major problems: tracking privacy state and augmenting an application's UI. Developers must independently add code to identify and track the privacy state of the shared device. Additionally, developers must make many privacy-related decisions with regard to application output. Without rigorous software development standards, these decisions may not be implemented consistently, leading to inadvertent privacy leaks.

An application-specific privacy state creates an inconsistent interface for users. If a user has more than one privacy-aware application executing on his personal device, then having different privacy management tools in each application can lead to more privacy leaks (e.g. the user changes the privacy state of one application but neglects to change the privacy state for the other applications).

This paper introduces the XICE Windowing Toolkit (eXtending Interactive Computing Everywhere pronounced 'zice') [5]. XICE has been built to allow any network-connected device to annex (via Wi-Fi, Bluetooth, etc.) interactive resources (e.g. screens, keyboards or mice) on other network-connected devices. XICE includes a straightforward, robust privacy framework that allows a user to specify whether a display/environment is public or private and whether he trusts its input hardware. The privacy framework also allows developers to write code that takes advantage of the user's privacy specifications. Providing a user-friendly solution for securing sensitive data in nomadic environments requires an interactive, understandable model and a UI toolkit that simplifies privacy-aware UI development.

This paper deals exclusively with the privacy issues inherent in a shared UI windowing toolkit, and not the broader privacy problem of cryptography, secure protocols, or online website privacy settings (e.g. facebook [6] or MySpace [7]). This paper and the XICE toolkit do not address these issues. This paper introduces a toolkit which helps protect the user and software from potential exploits inherent in sharing a window to a foreign display server.

Although the connection to a display server could be encrypted (e.g. using public/private key-pair encryption), XICE does not enforce this. Instead XICE assumes that any data transmitted out of the personal device is stolen, regardless of whether the display server or environment is trustworthy. For this reason, only UI output is sent to the display server and never any data files, and XICE's privacy framework is intended only to affect application output.

## 2. PRIVACY THREAT ANALYSIS

As people work nomadically, they interact with foreign devices. To protect sensitive data, the safest choice is to never annex devices belonging to others. However, users are then limited to resources on their personal devices. People should be able to use external devices, even suspect ones, in ways that protect their data. Malicious machines can take advantage of users in four ways: stolen output, stolen input, false input, and false output. These exploits, plus the potential for public embarrassment, comprise the *five key problems* inherent in nomadic computing.

### 2.1. Stolen output

Any information shown on a shared device can be stolen by that device. At worst, a malicious device could steal data like user credentials. For example, if an e-mail application has trouble authenticating with its e-mail server, the application may show a dialog similar to the one in Figure 2. This dialog shows the user's e-mail address, e-mail server, and password length. The shared device now has enough information to perform a brute-force attack on the user's e-mail account. An annoying, but less injurious consequence might be that the shared device could scrape the e-mail address and spam it.

To protect against stolen output, sensitive data should not be shown on a public screen. At minimum, if a window that typically contains sensitive data is shown on a public screen, the sensitive data it contains should not be transmitted to the shared device. It is better if that sensitive data is redirected to a private device.

To facilitate protecting sensitive data, the UI toolkit will need to identify public screens and provide that identification to applications. Each screen must be tagged either public or private. Applications can then use that state to prevent sensitive data from showing on public screens.

Identifying a public screen must be within the user's control. The shared device cannot specify the privacy state because the shared device could easily provide a false privacy state. Instead, the private state must be specified through the personal device, preferably by the user.

### 2.2. Stolen input

If a shared device is used to provide input, the device can steal that input. For instance, many people use internet cafes or hotel computers to check e-mail. These machines could easily harbor key loggers that steal usernames and passwords.

If an application is unaware that the input is distrusted, the application will request input through the shared device, regardless of whether sensitive data is included. For instance, if a web browser encounters a page that has login boxes, those boxes will show on the shared display. Then, users will likely enter their credentials directly via the shared device's input hardware, compromising sensitive data. Applications must be aware of public displays and any widgets that request sensitive

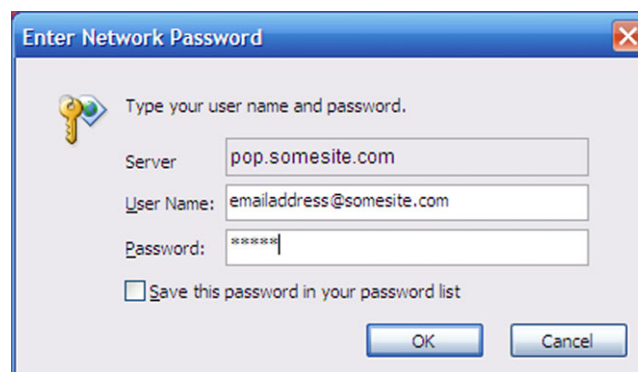


Figure 2. An inappropriate dialog for a public display—it shows private information. The display now has the user's e-mail server, e-mail address, and password length.

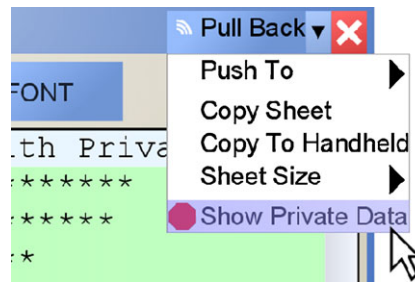


Figure 3. Options such as ‘Show Private Data’ can be exploited by a display that falsifies input.

data (e.g. password boxes). Requests on public devices for sensitive data may then be denied. The sensitive data request and input binding can be moved to the secure personal device. The user must be notified via the public device that sensitive data is requested on the personal device. Users need software to intuitively guide them with privacy and security [8].

### 2.3. False input

A shared device can control any application without the user’s consent. For example, if a shared device has malicious software that is familiar with a user’s application, the shared device could expose sensitive data by sending input to the application’s window causing changes to the user’s privacy settings. The ‘Show Private Data’ menu option in Figure 3 causes an application to expose all of the sensitive data within the window. To exploit this menu option, the shared device could send a mouse click event to expose the menu, and then another click to ‘Show Private Data’.

One way to prevent the shared device from supplying malicious input is to categorically deny the device’s input. However, input from a shared device could be much richer than input on a personal device. For example, a physical QWERTY keyboard on a shared display is much easier to work with than the soft keyboard on a personal device. The user must be able to use the input on the shared device if he chooses.

Another way to prevent false input is to remove all widgets that affect sensitive data from a window. However, such prevention limits what the user can do. For instance, a user may trust the environment for some of his sensitive data. When the ‘Show Private Data’ option (Figure 3) is attached to the window it affects, the user knows which window he is exposing when he selects that option, because he is exposing the window the menu option is attached to. If that option is not shown on the public display then there must be an abstraction on the personal device that he can use to expose that window. The user may then inadvertently use the wrong widget and expose the wrong window.

A better way to prevent false input is to securely confirm sensitive input. For example, if the user clicks on the ‘Show Private Data’ option, a dialog could be shown on the personal device to confirm that selection. Through the trusted input on the personal device, the confirmation dialog ensures that the user wants to expose a particular window’s sensitive data. If the display supplied malicious input to try and expose that window, the user would be alerted to that request by the unexpected appearance of that confirmation dialog on the personal device. By confirming actions that affect sensitive data, the display is discouraged from supplying false input.

### 2.4. False output

Shared devices can overtly change a window’s output in an effort to conceal malicious activity or to coax users into exposing sensitive data. As an example of concealing malicious activity, if no confirmation dialog is provided in response to false input, the malicious display could click the ‘Show Private Data’ option, and then immediately send mouse clicks to undo that option. The device may only present the last concealed version of the window so that the user is unaware that his privacy has been compromised.

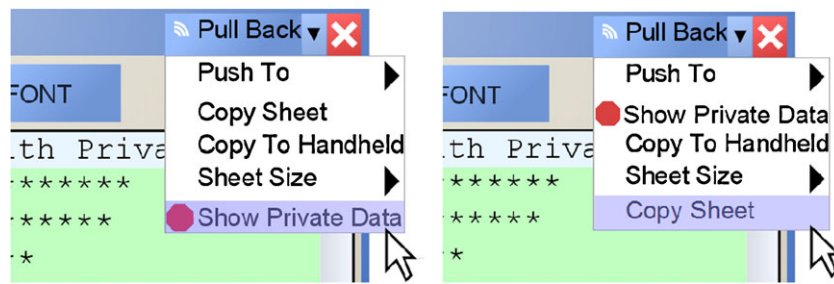


Figure 4. A shared device could swap the 'Copy Sheet' and 'Show Private Data' options. The image on the left is what the personal device transmits to the shared device. The image on the right is what the shared device shows.

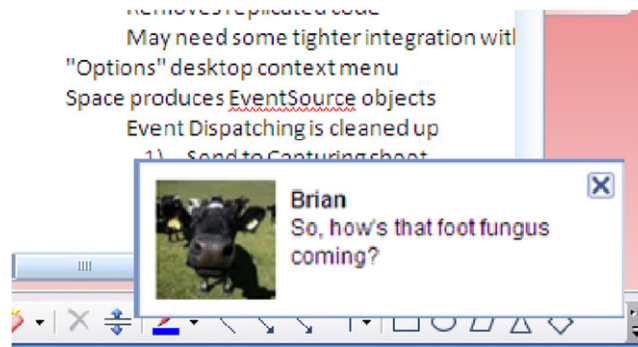


Figure 5. An embarrassing Instant Message shown during a collaborative session.

Providing convincing false output to coax the user into exposing sensitive data is difficult, but a malicious display may resort to such behavior when the user does not allow input from the shared device. For example, when the user provides all input from his personal device, the display only knows when the mouse moves or when the UI changes. If the malicious device accurately predicts a user's click on the 'Copy Sheet' option in Figure 3, the shared device could swap the graphical output of that option with the 'Show Private Data' option. The user sees an altered view (on the right of Figure 4). The personal device will interpret the user's click as 'Show Private Data,' because the personal device has a model of the graphical output sent to the shared device (on the left of Figure 4). The personal device automatically trusts this input from itself; consequently, the shared device tricks the user into inadvertently exposing sensitive data.

False output cannot be prevented, but it can be detected. If a user performs an action that affects sensitive data, that action should be confirmed on the personal device. Confirmation prevents exposure of sensitive data to a public device and enables the user to discover the malicious display. Since the user does not expect a confirmation dialog, its appearance alerts him to the display's malicious activity.

## 2.5. Embarrassment

In collaborative situations, users bring data to share with other people using shared displays. Even if sensitive data—as identified to applications—are protected, *embarrassing data*—relating to the social appearance of the user—may be exposed. Disclosure of either data type is unacceptable. Consequently, for the remainder of the paper both are considered sensitive.

One situation in which a user's embarrassing data may be exposed is when he receives an instant message on a shared display. Such messages are typically benign, but a message like the one in Figure 5 might be embarrassing. If the icon in that message were offensive, the social situation could be disastrous.



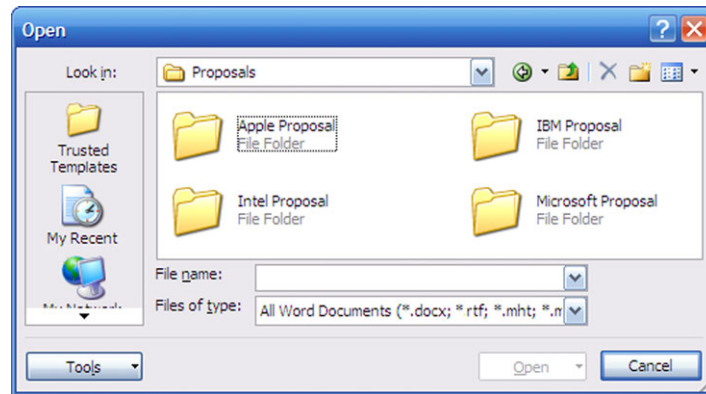


Figure 6. A file dialog with sensitive folder names.

In addition, many e-mail applications show notifications of an e-mail's arrival. The process through which these notifications appear is similar to instant messages.

Instant messaging clients are somewhat intelligent with regard to new messages; the latest versions of these clients will not show notifications when the user's active window is full-screen. Not showing notifications is useful when he plays a full-screen game or gives a presentation. But, if he demonstrates software in a typical window, the shared display is treated as private and the notifications appear. Another indication of the flaw of using window state to determine privacy state is when the user views a full-screen application on a private display (such as at his desktop), the notifications do not appear. The software misinterprets the full-screen application as a public situation, whereas the user is in a private situation where he wants to receive notifications. However, if applications are aware of the user-specified privacy state, the applications can intelligently choose when and where to show notifications.

When a user opens files, he sees dialogs like the one in Figure 6. If he makes a presentation to IBM, he might not want IBM employees to notice presentations for Intel, Apple, and Microsoft.

To prevent embarrassment, potentially sensitive data should not be shown on public devices. Privacy-aware applications can make appropriate privacy decisions when using public screens instead of assuming like instant message applications do. When annexing a public screen, applications should show potentially sensitive data on the personal device. When annexing a private screen such as one's own desktop, applications should show sensitive data on that private screen.

### 3. SOLUTION REQUIREMENTS

The authors envision new environments and styles of interaction that will emerge as users become more nomadic and computers change to facilitate such users. The core challenge for nomadic computing is to carry a portable computer but annex share devices and safely gain significant resources. An extreme, but potentially common, situation is a user carrying a handheld device to execute his software, but interacting with that software on a much larger screen (e.g. wall-sized). The annexed resources must provide more screen space or input than the user is carrying; otherwise, the user is unlikely to annex those resources because they do not offer any additional power.

The user should have consistent personal settings on any annexed machine. This consistency means that his personal settings and applications should be available wherever he is. The user must be able to carry his data, settings, and applications with him so that these pieces are uniform everywhere.

Furnishing data, settings, and applications to an annexed machine exposes the user to the five key problems. The data, settings, and applications should be available without being exposed, which requires trusted processing wherever the user is located. A personal device should process the user's data, settings, and applications.

When annexing foreign devices, users may frequently interact with sensitive data. Rather than exposing sensitive data on public screens, users need privacy-aware software to help protect their sensitive data. Users could restrict themselves to web-based applications and use kiosks for all computing needs. However, those who interact exclusively with kiosks are exposed to the five key problems. For example, a user must supply his username and password to the kiosk each time he visits a personalized website. Not only can the kiosk steal those credentials, it can swipe sensitive data (e.g. e-mail addresses) on those personalized sites. A user needs a trusted device through which he can safely view and enter sensitive information, regardless of whether the environment is private or trustworthy.

A privacy-aware application is more capable of protecting the user's sensitive data when it is informed about the privacy state for a shared device. The application must be able to show non-sensitive data on a public screen and simultaneously show sensitive data on the personal device's screen. Splitting the UI between the annexed device and the personal device has the potential to considerably enhance the user experience and greatly reduce the attack surface a shared device may exploit, because malicious shared devices can only steal sensitive data from application outputs rather than from data within the user's files. Applications with sensitive data that are aware of public screens can better protect the user's sensitive data.

The privacy behavior should be consistent and universal through all applications on the user's device. An inconsistent experience creates more privacy failures and user frustration. The Mac [9] has shown that to gain consistency across applications the toolkit must provide the consistency rather than relying on end developers. Thus, to gain consistent privacy-aware software, the privacy-aware decisions must be moved to the windowing toolkit as much as reasonably possible.

The nature of the nomadic computing environment necessitates windowing toolkits and applications that are privacy-aware. This must be dealt with before deploying a Dynamo- [10] or Personal-Server-like [11] environment. With a privacy-aware shared UI toolkit and applications, the user's software becomes aware of the privacy state he applies to his current environment and the software can augment itself accordingly.

### 3.1. Coding privacy

The granularity at which to integrate shared UI privacy into an application is an application-specific decision. Different applications and users will have varying needs for protection. For example, a word processor may treat entire documents as sensitive or open, while a spreadsheet may treat individual cells as sensitive or open.

The best granularity for privacy is unknown. There is a continual tradeoff between privacy violation and over-protection. Reducing barriers makes software easier to use and more transparent, but also increases the opportunity for violations and surprised and angry users. Creating more barriers increases the protection for users, but frequently makes users frustrated as they must navigate these barriers and make decisions. Developers need the freedom to experiment with their software designs so they can create the best software for their users. As will be shown in this section, the question of when to prevent sensitive data from appearing on an annexed screen is broad and has numerous potential solutions.

Currently, there is little to no windowing-toolkit-wide shared UI privacy protection available, so applications must make these decisions independently. However, with a good toolkit which provides system-wide privacy information, developers have greater freedom to explore the privacy-aware shared UI software space.

What follows are five examples of privacy granulation issues. This section discusses each option and proposes several theoretical approaches. None of these approaches has been tested, and each approach has varying levels of difficulty for both users and developers. The purpose of this (Section 3.1) is to show why it is necessary to have a windowing toolkit which makes these options possible.

*3.1.1. Word processor:* A word processor could have privacy integrated at different levels within the application, from fine-grained to coarse-grained. A word processor could be designed to protect



individual characters, words, sentences, paragraphs, pages, or even the entire document. Embedded images may require protection as well, which could be at the image level (block the whole image) or pixel level (block out someone's face using a rectangular or irregular region).

Somehow the sensitive data must be specified. With finer-grained protection, the software may require the user to specify each piece of sensitive data. For example, the user must highlight text and mark it 'sensitive.' On the other hand, developers may produce software that automatically detects and protects potentially sensitive information. For example, it may discover and protect e-mail addresses, phone numbers, social security numbers (SSN), and financial information. The sensitive data would not appear on a public screen unless specified by the user.

On the other hand, maybe it is easier for the user to specify which data is open rather than which data is sensitive. A developer may create a word processor that protects all data by default, unless the user specifies otherwise. However, the user specifies the open data, the sensitive data must be protected from appearing on public displays.

When the software protects characters, words, or images, it must provide feedback to the user that such information is protected. The software may highlight that information in some way while the user is working in a private environment.

When the user moves to a public environment, the sensitive data must be protected while potentially giving feedback about its existence. The word processing application may hide all sensitive data and not show that it even exists. Or the software may show gray rectangles where the sensitive data is located. 'Greeking' sensitive text may be preferred, in which case 'lorem ipsum' text may be inserted. A more radical case may be to collapse the sensitive data and show a note down the side of the page pointing at it.

*3.1.2. Application-independent privacy awareness.* There are many widgets that have privacy issues independently of the content. For example, showing the file dialog in Figure 6 on a public display is unacceptable. Content-assist menus that show recently used words may need to be blocked. Menus within the application that show the names of recently accessed documents may need protection. Launching an e-mail application to send a document may be more appropriate on a private display. Clearly some dialogs, menus, menu items, or even applications need to be protected.

Properly protecting the user's privacy requires that the user, developer, or software detect *privacy boundaries*—clear differentiation between open data and sensitive data. For example, specific data within a document may be considered sensitive, hence, the boundary between the surrounding open data and the sensitive data must be respected. Another clear privacy boundary is shown in the following example. Two people are discussing a document on a shared display space. When the user who owns the document opens the file dialog to choose another document to view, that user has crossed a privacy boundary. He thinks about the file system differently from his word processor, and the ways of protecting the file system are different from protecting data in the word processor. Detecting such boundaries in the UI design process and then protecting them in the implementation is important.

*3.1.3. Spreadsheet.* A spreadsheet has many of the same privacy-related design decisions as the word processor. Protection could be performed on the level of characters within cells, cells themselves, rows and columns, or even the entire document. For the user, marking individual rows or columns may be less tedious than marking individual characters, but the user may find any marking tedious. Developers may add code which automatically protects data based on the format. For example, it could protect all financial information, e-mail addresses, and phone numbers from appearing.

Some shared UI privacy-related decisions are different than a word processor. For example, a user may not want to share individual employee salaries, but may be willing to share aggregate personnel costs. A developer may write an algorithm that treats financial information as sensitive, except in cases where an aggregation (e.g. sum or average) is used on some minimum number of items (e.g. 5 or more). Graphs may also be treated as sensitive if they express data from cells

marked sensitive, unless the data is sufficiently abstracted from the original values. Clearly these decisions are not the same as the decisions for a word processor application.

Whether each of these spreadsheet-application-specific ideas is good or bad or even appropriate is unknown. But a good toolkit will allow developers and users to explore these decisions themselves.

Other similar issues are how to indicate at a private display that sensitive data will be protected if shown when connected to a public display and protecting some dialogs, menus, menu items, and applications.

*3.1.4. E-mail manager.* An e-mail application is likely to contain sensitive data especially because the data is generated by other users. Expecting other users to properly tag data as sensitive may be foolhardy. But requiring a receiving user to individually tag information within each e-mail message will likely frustrate him. More proactive protection tools must be necessary. For example, the e-mail application may actively prevent the e-mail list and individual e-mails from showing on public displays.

E-mail is frequently shared with other users. For example, in a collaborative situation users might show e-mail messages to each other. The e-mail application itself will appear on the personal device, and the user can elect to share individual e-mails.

When an e-mail is shown on a shared device, e-mail addresses in the 'To' and 'From' fields represent another privacy boundary and should not be sent to the shared device, otherwise the addresses could be stolen. Two new situations arise when sensitive data is blocked from appearing on a public device. First, the user may want to privately access those e-mail addresses; this scenario is called the *reviewing situation*, because the user may need to review sensitive data without exposing it on public screens. Second, the user may want to explicitly show those e-mail addresses anyway because he trusts the audience; this scenario is the *field override situation*.

*3.1.5. Web browser.* A web browser offers some interesting privacy-related challenges. One such case is when a user visits his bank's website. He does not want his sensitive data to show on a public display. Or consider a user who visits a news site which requires a user id and password to view articles. The pages within the site may be considered open while the login page should be sensitive.

Web standards may change so that sites can specify pages or content that is sensitive, but until that point browsers have several approaches for privacy-aware shared UI designs. These approaches include protecting data at the window level, the page level, widget level, or at the content level.

The browser may protect a specific window the user has created, regardless of the tabs shown within that window. Consequently, when the user shares that window on a public screen the content shown in each of that window's tabs are also shared. If the user chooses to share a window containing a sensitive page, the URL for that page may be blocked from appearing in the address bar so that others in the room have a more difficult time finding and accessing that site.

Alternatively, the browser may protect individual pages from showing, requiring the user to explicitly authorize each page for public screens. Or the browser may be designed so that the user must explicitly authorize specific domains, sub-domains (e.g. for blogs, or sub-sites that may contain sensitive information), and possibly URL folders. As the user navigates away from an authorized domain/sub-domain/folder, the user must re-authorize. If the user logs in to a site, the rest of that site may be considered sensitive unless specified otherwise. A site that is accessed through HTTP may be considered open while a site that is accessed through HTTPS is considered sensitive. Or developers may come up with a more manageable page-level privacy boundary.

Content within the page may need to be protected. For example, login boxes on a web page should probably not be available on a public display, but still available to the user (a variant of the reviewing situation). After logging in, easily-identified sensitive data (e.g. e-mail addresses) within a page may be discovered and protected by the browser, even though the rest of the page is considered open.

Assistance from the browser may also contain sensitive information. A privacy-aware browser may be designed to hide content-assist drop-downs, recent history, or bookmarks, while still

showing most menus. Or, if the user marks certain pages or sites as sensitive, those specific pages may be blocked from appearing in content-assist drop-downs, the history, and bookmarks. There are clearly a wide variety of shared UI privacy issues that affect browsing and the features expressed by the browser.

*3.1.6. Instant messenger.* As discussed in Section 2.5, instant messaging software can present several problems. It is unlikely that users will tag individual messages or content within the messages as sensitive. So the software must treat all such messages as potentially sensitive. When the user is at a private screen all his messages should appear, but when in a public environment his messages should not appear publicly.

Similarly, the contact list should not be shown on a public display unless the user explicitly chooses to share that list. On the other hand, the user may choose to share a specific conversation thread to a public screen.

### *3.2. XICE privacy-aware strategy*

Specifying where privacy boundaries should be placed is a difficult problem. Clearly the shared UI privacy issues in a word processor are not the same as the issues in a spreadsheet, e-mail manager, web browser, or instant messenger. Each of these tools should have consistent mechanisms for allowing users to share a window, protect sensitive data, review sensitive data, and show sensitive data anyway. Therefore, the XICE toolkit is designed to incorporate many privacy-aware shared UI decisions for both users and developers.

The XICE toolkit provides important facets for privacy-aware software design. One facet is that the toolkit provides information to applications about how the user feels about an annexed display space (the privacy state). Another is that the toolkit provides a set of tools available to developers from public/private windows to fine-grained widgets that can help the developer protect the user's sensitive data on public screens and from distrusted input.

Critical to implementing effective privacy-aware applications is lowering the overhead required for developers to implement privacy awareness. If developers make fewer decisions when implementing privacy awareness, but can more consistently achieve their desired results, then users will also have a better experience. Currently, developers must start from scratch when building privacy-aware applications, so moving many decisions into the toolkit is beneficial.

Considering that no such windowing toolkit exists (as will be shown in Section 4), the XICE toolkit is compared against the current state of no options for developers (other than build-from-scratch), no information for applications (no system-wide privacy state), and no control for users. For example, when writing an e-mail application, the developer may want to declare new e-mails as sensitive and show them only on a private display. Previously these decisions were difficult but with the XICE framework they are straightforward.

In summary, the overall protection of nomadic user privacy has *five parts* that the toolkit and applications must implement. First, the toolkit must be able to distribute the UI between the personal device and the display, allowing the application to independently render to both displays, and giving the user a consistent, richer experience. Second, the toolkit must allow the user to identify an annexed display's privacy state, and the toolkit must inform applications of that privacy state. Third, applications must avoid showing sensitive data on public devices, while still showing open data. Fourth, applications should either not allow sensitive input from distrusted devices or confirm such input on the personal device. Finally, users must be able to resolve the reviewing situation and field override situations by explicitly overriding privacy controls.

## 4. PRIOR WORK

The solution for protecting a nomadic user's privacy must incorporate the five parts enumerated in the preceding section. A wide variety of privacy-aware applications have been developed, each with novel aspects in how they protect sensitive data, but few are implemented in the toolkit or

incorporate UI distribution. A generalized solution that can be used simultaneously by a wide variety of applications is preferred.

Examples of privacy-aware applications include PrivateBits [12] and Privacy Blinders [13]. PrivateBits has novel tools for blocking sensitive data contained in a web browser. For example, PrivateBits filters the browsing history and auto-complete hints when the browser is in public mode. Privacy Blinders renders moveable black rectangles over the top of sensitive information within a web page. The source web server tagged that information as sensitive. But neither approach incorporates UI distribution to shared devices, so input cannot be controlled, nor can they resolve the reviewing or field override situations. If the user were to use X11 or RDP to annex shared devices, Privacy Blinders fails to prevent sending sensitive data to the shared device. Because the black rectangles are rendered after rendering the sensitive data, the user sees the information as being blocked but the display server sees the rendering calls for the sensitive data.

Berry, Bartam, and Booth [14] have developed an approach that uses simple UI distribution. Their tool integrates with Microsoft Word, Excel, and Internet Explorer and exposes a replicated, public view via VGA. Users tag information as sensitive using the application-provided highlighter tool, and then the public view blocks that sensitive data. In addition, file dialogs are prevented from showing on the public display. However, the public view only annexes as much visual space as the personal device has. Consequently, the shared device provides no additional screen space. The applications are also not informed of the privacy state and cannot take action to protect the user's sensitive data.

Symbiotic Displays [15] is a wrist-watch application that can annex a display to show e-mails. This approach effectively distributes the UI across two displays with different privacy states. It also supplies an interesting resolution to the reviewing situation by allowing the user to select sensitive words (which are blurred) on the shared display and view the words on the wrist-watch's screen. Unfortunately, although other people in the room cannot see any sensitive data, all of the sensitive data is transmitted to the shared device. The shared device is the computer that makes a decision as to how to block the sensitive data. Consequently, this approach is still vulnerable to stolen and false output and input.

Oprea *et al.* have a solution for safely interacting with personal data on a shared device [16]. The personal data is stored and processed on a remote computer, and the personal device establishes a connection between the shared device and the remote computer, as illustrated in Figure 7. The personal device is augmented with an optical mouse to supply all pointing input. Unfortunately, this solution only protects against false input and some stolen input (users may still use the shared device's keyboard to enter credentials), but not stolen output or embarrassment. In addition, because it relies on the Internet and VNC, displays without Internet access cannot be annexed and large distances slow the interactive experience.

Sharp *et al.* provide a similar solution to Oprea, except that Sharp protects against stolen output [17]. Oprea shows everything the remote computer renders while Sharp blurs all text before transmitting the UI to the shared device, including all non-sensitive text. To view any of the

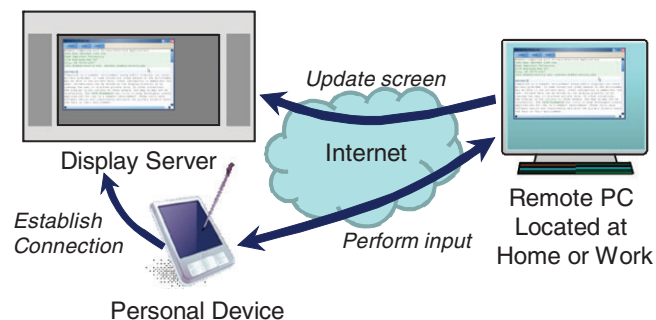


Figure 7. Oprea's annexing solution. The personal device brokers a connection between a VNC display server and a remote PC, and then supplies all pointing input.

rendered text, the personal device shows a complete, scrollable, un-blurred view of what the remote machine renders. However, most text within an application is not sensitive (open) and is useful if shown on the shared device. In this case, applications are not privacy-aware so the toolkit cannot intelligently block only sensitive data.

Mobile Composition [4] provides a web-based privacy solution that splits web pages across two machines. The shared device shows all of the user's open data, but areas of a page containing sensitive data are tagged as sensitive by the website designers and are encrypted such that only the personal device may decrypt them. In this way, Mobile Composition protects against the four problems of stolen and false output and input. However, this solution does not support applications outside of the web and relies on the web for all processing, which prevents the user from interacting with software when Internet access is unavailable.

SessionMagnifier [18] takes a similar approach to Mobile Composition, except that SessionMagnifier uses the PDA as a proxy web server for the shared display. Then SessionMagnifier can filter information on a web page. However, this approach suffers from the same problem as Mobile Composition—SessionMagnifier only operates with web applications.

## 5. A PRIVACY-AWARE UI TOOLKIT

Nomadic computing necessitates UI distribution from personal devices to shared devices. For the goals outlined in this paper, a significant failing of most existing privacy-aware technologies is that they do not distribute the UI from a personal device to a shared device.

XICE was developed using Java 1.6 to create a seamless nomadic computing environment. XICE handles the details of annexing shared devices and distributing application UIs to them. The graphical output of each application window is abstracted as a scene-graph (a tree-structure display list) so that windows can be rendered on any shared device without regard to display size or resolution. A scene-graph is sometimes called a presentation tree. Scene-graphs have been studied for several decades in tools such as Graphics Kernel System [19] or Programmer's Hierarchical Interactive Graphics System [20]. Rather than using the traditional damage-repaint rendering model an application builds a tree structure of drawing commands and hands that tree over to a rendering framework which renders the tree independently of the application. Toolkits such as Silverlight [21], Windows Presentation Foundation [22], and JavaFX [23] are popularizing scene-graphs as integral components in GUI development for desktop applications. As measured in prior research, the scene-graph also dramatically reduces the computational burden on the personal device relative to RDP, VNC, and X11, making the protocol more suitable for small personal devices [5]. Via XICE, users can annex shared devices in airports, restaurants, personal offices, living rooms, etc., and push windows to them. In addition, XICE has the appropriate hooks to make privacy a first-class citizen.

### 5.1. Windowing toolkit interaction

Users working nomadically take personal devices to shared devices and annex them. The personal devices run XICE, and the shared devices have the hardware and software necessary to establish network connections and to process XICE rendering commands. Users can push application windows to the shared devices.

The windowing toolkit must enable users to quickly push windows to an annexed display. In the top-right of every window is a drop-down button which provides access to the toolkit options for that window. This button takes on different appearances—some of which are shown in Figure 8—depending on the state of the window. Windows on the personal device show the 'Push' option, which, when clicked, starts the process of connecting to a shared device.

After annexing a shared device, all other windows on the personal device change to 'Push: Machine'. All shared windows say 'Pull Back' to provide a single click for removing that window from the shared device. Applications continue to execute on the personal device; only the graphical output is transmitted to the shared display.



Figure 8. Common decorator widgets. The X closes the window, while (a) initiates a connection to a shared device, (b) pushes a window to a currently connected shared device, and (c) pulls back a window shown on a shared device.

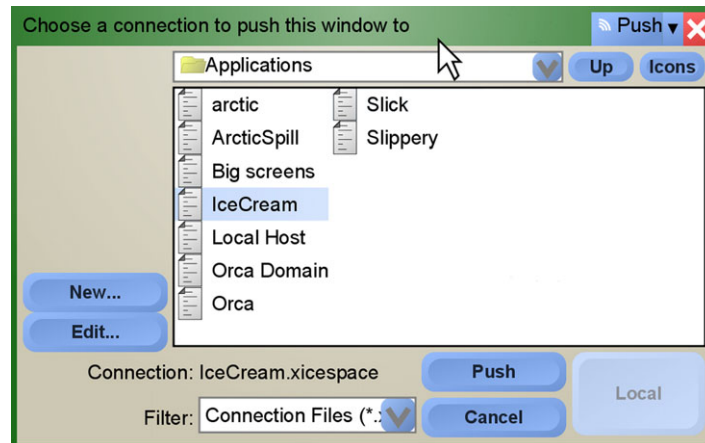


Figure 9. The XICE connection dialog. A user has chosen to push a window to the machine he named IceCream. By default, the configuration files are stored in the 'Applications' folder and the view shows only the configuration files by filtering on the file extension.

After a user clicks the 'Push' button, the connection dialog in Figure 9 is shown listing several configuration files. Each file contains the information necessary to connect to a shared device. A user just picks a previously-used device and the current window is pushed to it.

If the user wants to connect to a new shared device, the user may click the 'new' button. Figure 10 shows the new connection wizard which walks the user through configuring the connection. Using this wizard configures the following values:

- The shared device's connection string (domain name or IP address)
- The connection name (e.g. Orca), created automatically or assigned by the user
- The shared device's trust level (described in section 5.2)
- Where input comes from (defaults to the personal device)

The user only needs to enter the connection string and select the screen's type. The connection name is a unique identifier for the screen generated from the connection string but the user may change it. The three most common types of screens are presented to the user, each with different presets for the display server's trust level and input sources. These presets will be discussed in the next section. Configuration files store all these settings so that regularly used devices, such as desktops or home televisions, do not need to be reconfigured.

The process of selecting a device or entering its connection information could be simplified by using local area network broadcasting technologies such as Bonjour [24]. Currently XICE does not use these technologies, but if it did, these technologies would list shared devices near the personal device so that XICE could show only proximally available devices, highlighting those the user has previously annexed. These broadcasting technologies could prevent clutter from the many devices the user has annexed before.

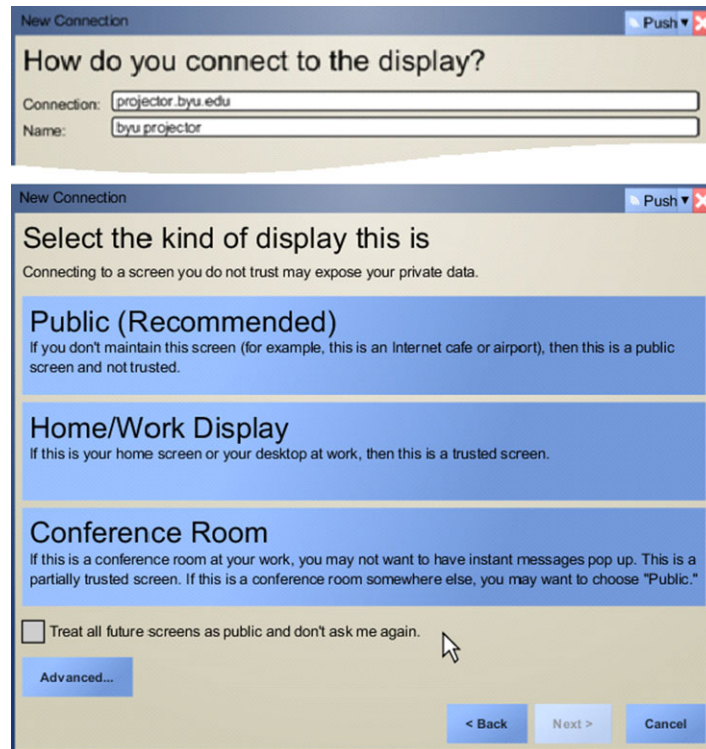


Figure 10. The XICE new/edit connection wizard. The user must supply the connection string and select a trust level for the display.

## 5.2. Mitigating the privacy problems

When a user first connects to a screen he may choose one of the three types of screens presented in Figure 10. This selection results in setting two separate options: the input source for windows shown on that screen and the privacy state for the shared device. Input can either come from the personal device or from the shared device. Input from the personal device is always trusted. However, if the input source is the shared device then the trust level affects that input.

The shared display privacy state may have one of four *trust levels*. When a screen is public, then windows shown on the shared device should only show the user's open data. Private screens should show the user's open data and sensitive data. With respect to shared input, if the input is trusted, then any input from the shared device is accepted and processed normally. But, when the input is distrusted, software may take protective actions. For example, if the user clicks on the 'Show Private Data' option in Figure 3, then that click is confirmed on the personal device. Most of the time, input is allowed without double-checking—like when typing up an e-mail via a public display—but the input is double-checked when it affects sensitive data.

The four trust levels are delineated in Figure 11. The first trust level is *public only*, where the display is public and input is distrusted. The *public with input* trust level is useful when the user trusts the input but the environment is public. A user hosting a discussion in his company-owned conference room can implicitly trust the display server's input to not be malicious, but he may not want sensitive data to appear on the shared display. The public with input trust level protects his sensitive data without needing to confirm input that affects sensitive data. The third trust level is *display only*. The display only option may be used in the situation where the user trusts the audience but not the input. For example, if a lawyer is working with clients he believes are providing malicious input he may treat the display as public and the input as distrusted. Finally, *full trust* is assigned by the user when the display is private and the input is trusted, such as at his desktop.



		Input	
		Distrusted	Trusted
Display	Public	<i>Public Only</i> Show open data and treat input as distrusted	<i>Public with Input</i> Show open data and treat all input as trusted
	Private	<i>Display Only</i> Show everything and treat input as distrusted	<i>Full Trust</i> Show everything and treat all input as trusted

Figure 11. The four different trust levels. The user can trust or distrust input from public or private displays.

Room Type	Trust Level	Input Source
Public	Public Only	Personal Device
Home/Work	Full Trust	Display Server
Conference Room	Public with Input	Personal Device

Figure 12. Settings used for each of the options presented in Figure 10.

An important feature of XICE is the ability to redirect input—input from the personal device may be used to interact with a UI shown on an annexed display. For example, a user may use the input hardware on his personal device to control a mouse cursor on the annexed display. The display is informed of the cursor movement but none of the mouse clicks. The clicks are dispatched to the widget the cursor is over. After selecting a widget the user can use the input hardware on the personal device to enter text into that widget. This input redirection adds an extra level of security for users selecting the public only and display only trust levels.

When the user selects the type of screen in the step of the connection wizard in Figure 10, that selection chooses both the trust level and input source. While each trust level can accept a display server's input (trusted or distrusted), the personal device does not necessarily accept that input by default. The combination of trust level and input source results in eight possible options. Rather than presenting the user with all eight options, the authors opted to provide users with what they believed to be the options best-paired with the three most common situations, hence only three options in Figure 10. Figure 12 enumerates each option with its trust level and input source settings. The presented options are statically defined and are the most common options the user is expected to encounter. The user may, at his discretion, click the 'Advanced' button and adjust the trust level and input source directly, in which case all four trust levels are presented to the user along with both input sources.

Because the dialog in Figure 10 is not actually tied to the XICE protocol but is instead part of the UI presented to the users, other handheld providers could implement these options differently. For example, home consumers could be presented with a dialog that is different from the dialog shown to corporate consumers. Or the dialog's options could be dynamically provided. For example, when the user connects via his employer's wireless network the widget could show him only the work and conference room options with the conference room option listed first. Alternatively, if the user connects via a wireless network he labeled public (e.g. through the Microsoft Windows Network Configuration dialog) then the XICE configuration dialog could present him with just the public only option.

**5.2.1. Customizing display privacy settings.** The trust level is attached to the connection information for the shared device, and is not configured by the shared device but is configured by the user. Different users may apply different privacy states to the same shared device. For example, a user may set his desktop device to full trust, while a visiting coworker might consider the same shared device to be public with input.

A user may want to change the trust level of a shared display. Consider the *visiting student situation*. Suppose a teacher is preparing student grades at her desktop when a student stops by to

review his grades. The teacher would want the full trust desktop display to change to public with input so it will block her sensitive data (like others' grades) from the visiting student.

*5.2.2. Customizing window privacy settings.* Each window shown on a shared device can have a privacy setting that is independent of the device's privacy state. On private screens, the window's privacy setting is ignored and sensitive data is presented. On public devices, *protected windows* protect their sensitive data while *exposed windows* show their sensitive data. By default all windows on public displays are protected windows but the user may change the windows to exposed windows.

This mixed-state view of windows on the public device has some useful situations. Suppose the student in the visiting student situation has submitted an assignment containing sensitive data; the teacher needs to review that assignment with the student. The teacher can open the assignment and change that window to an exposed window using the 'Show Private Data' menu option in Figure 3. The teacher and the student can review the assignment as though that window is on a private display.

*5.2.3. Customizing widget privacy settings.* In addition, individual widgets may have their own privacy settings. For example, in the visiting student situation, the teacher may want to review the student's grades with him. If each row in the grading spreadsheet has an independent privacy setting, the teacher may change the setting on the row containing that student's grades. Only his grades are presented; other students' grades remain hidden.

### 5.3. Blocking data on public devices

Building a custom toolkit affords opportunities for quick and easy experimentation, solidification of solutions built using the toolkit, and development of custom UI responses to the privacy state of a device [25]. XICE is constructed to accomplish these goals. The rest of Section 5 focuses on how XICE is implemented to support privacy for both users and developers.

*5.3.1. Blocking windows.* To show a UI on a display, an application must first build a scene-graph representation of the UI. When the application creates a window the application supplies a scene-graph which XICE renders on the created window. Because a scene-graph may be rendered on one window and a window may render one scene-graph, the two terms are interchangeable.

An application may show five types of windows (scene-graphs). *Plain* windows do not carry sensitive data and can be shown on public or private screens. *Sensitive-data* windows (like e-mails) may initially be shown on public devices, but the application will protect the sensitive data. *Private-first* windows (like instant messages) might contain sensitive data, so these windows are initially shown on personal devices and users must explicitly choose to show them on public screens. *Private-notify* windows (like file dialogs) function similarly to private-first windows, except a notification is shown on the public screen when the private-notify window is redirected to the personal device. *Private-only* windows (like login dialogs) contain only sensitive data and will never be shown on shared devices. Users and software are not allowed to push private-only windows to public displays.

To create a consistent experience for users XICE uses Java reflection to identify several of these different windows. XICE detects a specific annotation, `@PrivateDialog`, on the root class of a dialog's scene-graph and inspects its properties. Then XICE can consistently redirect those windows as necessary. XICE provides the `PrivacyHandling` enumeration which can be passed to the `@PrivateDialog` annotation constructor for private-first (First), private-notify (Notify), and private-only (Only) windows. For example, the XICE-supplied `FileDialog` type is tagged with the `@PrivateDialog` annotation (circled in Figure 13), which defaults to being a private-notify. If the `@PrivateDialog` annotation is detected then XICE will show that scene-graph on the personal device instead. Plain windows and sensitive-data windows are untagged—the salient difference is in how the application treats the data within the window.

```

@PrivateDialog
public class FileDialog extends ModelView
{
    ...
}

```

Figure 13. How to annotate a dialog as private-notify.

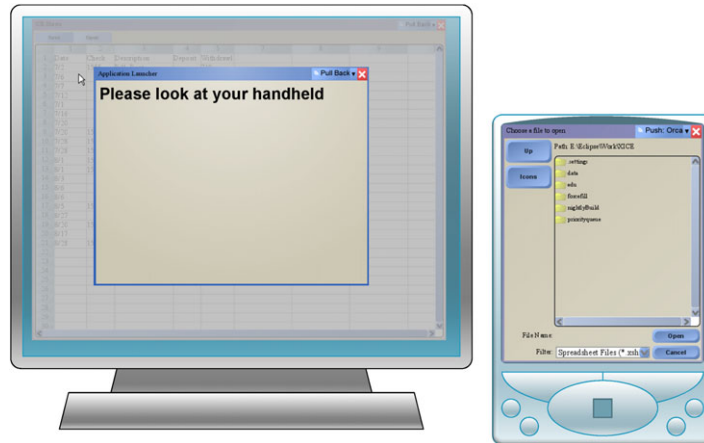


Figure 14. When showing a file dialog on a public display, the user is given a heads-up dialog instructing him to look at his portable device.

If the user expects an application to show a window on the shared device, he might not realize that the window has been redirected to the personal device. Hence, when an application launches a private-notify or private-only window and XICE redirects it to the personal device, the toolkit automatically shows the heads-up window in Figure 14 to cue the user to look at his personal device.

Because the window type may be attached to the window's scene-graph, XICE can enforce the privacy setting for that window wherever that window is used. Otherwise, developers must implement the redirection code everywhere they use those dialogs which may lead to occasional leaks. Instead, by using Java annotations, developers do not have to consider privacy when showing private-first, private-notify, or private-only dialogs. For example, XICE-supplied file dialogs and any subclasses are tagged as private-notify; hence, the embarrassing situation illustrated in Figure 6 will not happen with any file dialog even if the application using the dialog is not privacy-aware.

All five types of windows are created by applications. To show a window, the application must have a reference to the destination display. This reference is represented as a *Space* object within XICE. One *Space* represents the personal device. Any annexed device is represented with an additional *Space*.

The *Space* object has a property that contains the privacy state for the device. Applications can query this value and make decisions independently of XICE. For example, a word processing application may provide an option for always opening documents on private screens. If the option is set, then the software uses code similar to the listing in Figure 15 to redirect documents to the personal device's display when the open command is issued via a public screen.

**5.3.2. Blocking widgets.** In order for a window to appear on a shared device, the scene-graph for that window must be serialized and sent to the shared device. Widgets are embedded within scene-graphs. XICE only serializes the graphical output of the widgets and does not serialize any of the code that controls those widgets. XICE uses a custom text-based serialization framework so that software in other programming languages and frameworks may benefit from XICE.

```

public void openFile(String fileName)
{
    Space space = locateSheet().getSpace();
    if(space.getPrivacyState() == PrivacyState.Private)
    {
        //Display is private: show on shared device
    }
    else
    {
        //Display is public: show on the personal display
    }
}
    
```

Figure 15. Using this code, an application chooses where to show all newly opened documents.

```

private void makePrivate(ResizableContainer widget)
{
    FullPrivacy container = new FullPrivacy();
    container.setPrivateView(widget);
    this.add(container);
}
    
```

Figure 16. When getting automatic privacy protection, a developer would add the first two lines in this method.

For sensitive-data windows, developers may protect sensitive data by hiding widgets within the window. For instance, a developer may choose to hide only the username and password boxes on a public device, but the developer wants these boxes to show normally on private devices. XICE provides the FullPrivacy widget to address this situation. The developer uses code similar to the listing in Figure 16 to wrap each widget he wants protected with a FullPrivacy widget, and then widgets containing sensitive data are blocked on public devices, but available on the user's personal device. The FullPrivacy widget ensures that the widget is only seen on private screens; the widget is replaced with a gray rectangle on public screens so that none of the scene-graph produced by the widget is transmitted to the shared device. This coding technique allows the interactive behavior of any application to be simply wrapped within standard privacy controls.

Suppose a developer has created a simple application for organizing notes. A user can create notes anywhere within the application window, color the notes any way he chooses, and reposition the notes. This application is illustrated in part (a) of Figure 17. Further, suppose this developer believes that users may want to designate certain notes as sensitive. A user would select a note then instruct the application to tag that note as sensitive. The developer can then use the FullPrivacy widget to protect the sensitive notes. The note widgets need not consider privacy issues and the FullPrivacy widget knows nothing about notes but protects all interaction inside of it.

An alternate option to graying out a sensitive widget would be to hide the widget entirely. For some users hiding such widgets may be useful. For example, the act of graying out widgets informs other people in the room that the user does not trust them to see the contents of those widgets. The user may want to have the widget completely hidden instead. On the other hand, hiding such widgets makes it difficult for a user to know of the existence of the hidden widget. If he is familiar with the application's interface and data then hiding the widget may be acceptable. But if he is not familiar with the interface and data then he may have trouble addressing the reviewing situation or the field override situation. The relative benefits of both approaches must be weighed by the application developer.

*5.3.3. Customizing the public view.* Application developers may want to customize sensitive-data handling according to the shared device's privacy state. For instance, a spreadsheet developer might build an application that acts like a normal spreadsheet on a private screen, but hides

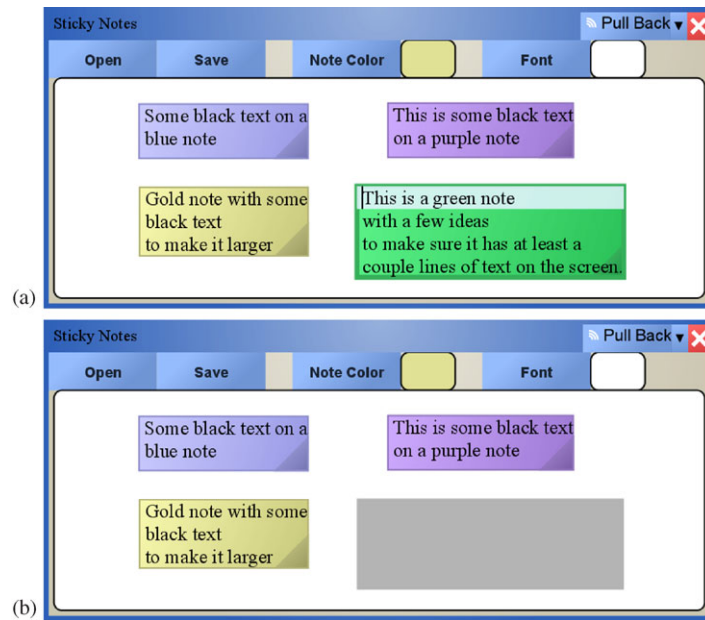


Figure 17. Augmenting an existing application to protect sensitive data. Image (a) is the original application. The bottom-right widget is wrapped with a FullPrivacy widget. Image (b) shows the same window on a public device.

```
public void visualSetupMethod()
{
    Sheet window = this.locateSheet();
    if(window.getImplicitPrivacyState() ==
        PrivacyState.Private)
    {
        //Setup Private View (show everything)
    }
    else
    {
        //Setup Public View (block sensitive data)
    }
}
```

Figure 18. Template code for a custom visual setup. This code checks on the privacy state of the widget's window.

rows and columns containing financial information when shown on a public screen. Or, an e-mail application developer might show e-mails in their entirety on a private screen and then block all e-mail addresses embedded within an e-mail when the e-mail is shown on a public screen.

Similar to how the Space supplies its privacy state each window has an independent privacy setting property. Applications can query this information to change their appearance. Code for getting the privacy setting is shown in Figure 18.

Any time the privacy setting of a window changes, embedded widgets are notified of that change. For example, when a window is pushed from a private screen to a public screen, a *recontext event* occurs. A recontext event is somewhat similar in purpose to repaint or damage events in more traditional damage/redraw architectures. When the user changes the privacy setting of a window, the recontext event also occurs. The recontext event notifies each widget of the change, allowing the widget to query the window's changed privacy setting and update the widget's output. Because XICE delays scene-graph serialization until after all widgets have processed the recontext event, the private view of the widget will never be inadvertently serialized to a public screen.

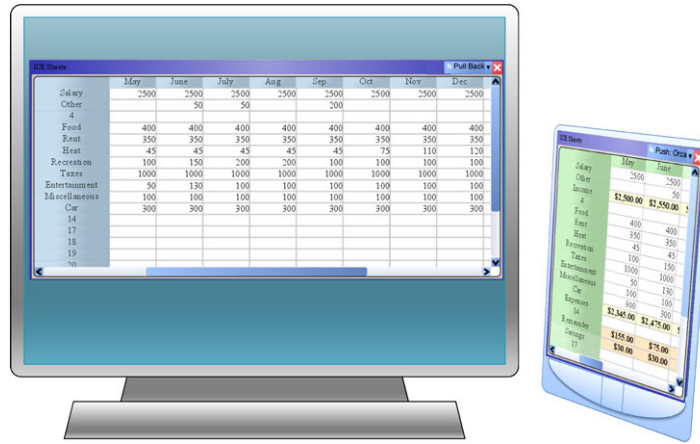


Figure 19. A possible synchronized custom view. The public display (left) blocks sensitive data. The portable device (right) simultaneously shows the same spreadsheet with the sensitive data exposed (the highlighted rows at the bottom and near the top).

#### 5.4. Reviewing sensitive data

Blocking sensitive data on applications can become frustrating to users who need frequent access to that data. If a user can access his sensitive data without exposing it on a public screen, as in the reviewing situation, his privacy remains protected.

XICE facilitates reviewing sensitive data on a personal device without exposing that data on shared devices. Reviewing is supported for windows, widgets, and custom reviewing situations.

**5.4.1. Reviewing windows.** Consider the spreadsheet application mentioned in Section 5.3.3. If a user opens this application on a kiosk in a public environment, he may want to see broad overviews of the data without exposing individual values. The user could interact with a spreadsheet on a public screen, and sensitive data in the corresponding cells will be highlighted on the personal device. Then the user can glance at his personal device to see the actual values without exposing the sensitive data on the shared device. Synchronizing the public screen and personal device spreadsheet views is particularly helpful if the personal device is a handheld which can only show a few columns and rows of data at once. The synchronized view is illustrated in Figure 19.

To synchronize the two spreadsheet views, one solution would be to put the widget in both scene-graphs. However, if widgets were in multiple scene-graphs, a single widget instance would have to support a potentially unlimited number of contexts (i.e. privacy settings for each window), states, and sub-graphs. To minimize the demand on the widget, simplify the scene-graph representation, and streamline the serialization process, XICE does not allow a widget to be embedded in multiple locations. Instead, widgets built using model-view-controller design facilitate the same visual result—two views which share the same model.

To create synchronized views across devices, XICE ensures that all view-controllers (scene-graphs and widgets) can be cloned and that cloned view-controllers share the same model. As XICE clones each widget, only its view-controller is copied; then the copy is pointed at the original widget's model. The cloned scene-graph is rendered on a new window on the personal device. The original view remains on its window on the shared device and retains its privacy settings.

From the user's perspective cloning windows is straightforward. To create a clone of a protected window on the personal device, the user clicks the 'Copy To Handheld' menu option in the title bar of the protected window (Figure 20). XICE then clones the scene-graph and renders the clone on a new window on the personal device.

**5.4.2. Reviewing widgets.** Section 5.3.2 mentions blocking a username and password using the FullPrivacy widget, but a user must still be able to enter input to those widgets via his personal



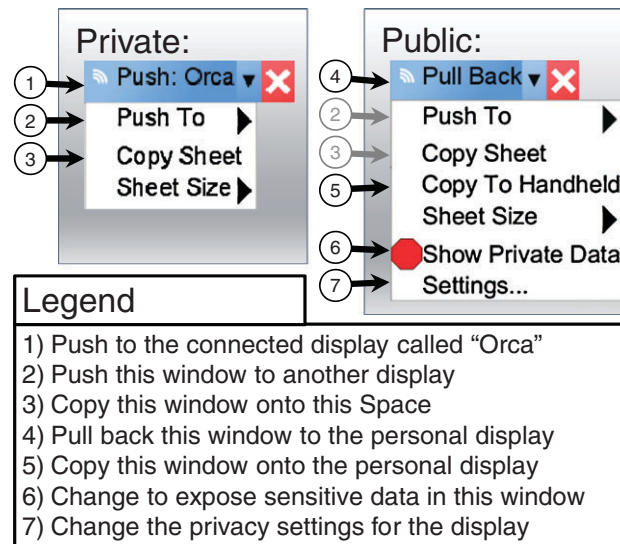


Figure 20. Privacy-related menu options on the title bar of every window. Some options are specific to public displays so users can properly protect their data.

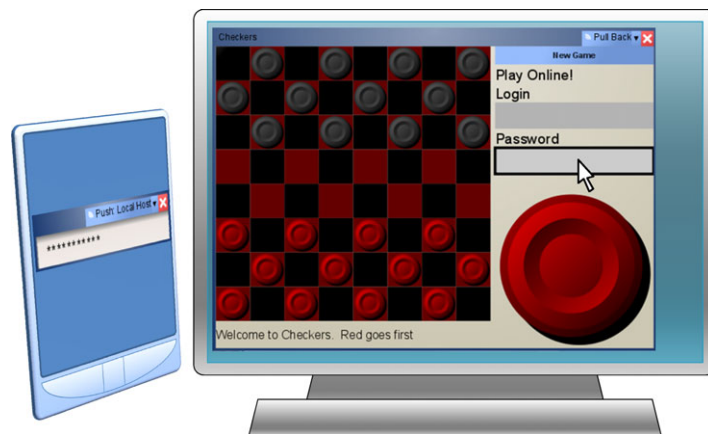


Figure 21. Hovering over a FullPrivacy widget on a public display (right) shows the blocked widget on the personal display (left).

device. When annexing a private screen, he can access the username and password normally. On a public screen, he must take a few extra steps. First, he clicks on (or hovers the cursor over) the gray rectangle where the username or password textbox should be. The FullPrivacy widget then shows the textbox on the personal device (this is illustrated in Figure 21). He then clicks the textbox on the personal device and enters his credentials using the personal device's input. This interaction redirection is completely invisible to the application.

*5.4.3. Custom reviewing.* If a developer chooses to customize data blocking, XICE provides window and device privacy states, but does not offer much other assistance. Any custom reviewing will require more programming time and effort. The developer may instead decide to rely on XICE's cloning infrastructure (Section 5.4.1) to resolve the reviewing situation.

### 5.5. Privacy control

In addition to reviewing sensitive data on the personal device, users may want to show that data on a public screen. Perhaps the user does not consider the data sensitive for a particular audience.



XICE facilitates users showing windows and widgets on public screens, and supplies applications with enough information to implement custom data exposure options.

*5.5.1. Showing windows.* Whenever a shared device is annexed, a connection management window is shown on the personal device. On this window, a button click causes the privacy state of the Space to change from public to private or vice versa. This functionality addresses the portion of the visiting student situation when the teacher must change her desktop to the public state.

Each window on a public display has the ‘Show Private Data’ menu option, which allows the user to expose all sensitive data on that window. This menu option resolves the e-mail issue raised in Section 3 by allowing users to show individual e-mails and all the sensitive data in them on public screens, if they so desire.

*5.5.2. Showing widgets.* If a developer uses the FullPrivacy widget to block another widget, the user can expose the blocked widget on a public screen. On public screens, when the user clicks on the gray rectangle that replaces the private widget, a ‘Show Data’ option is provided which exposes the blocked widget. When the input is tagged as unsafe, the user must confirm that widget exposure on the personal device. This option remedies the field override situation (mentioned in Section 3).

*5.5.3. Custom showing.* If a developer chooses to customize sensitive data blocking, he should also create code for exposing that data. While the developer could rely on the ‘Show Private Data’ menu option to show the sensitive data, this solution is discouraged, because all sensitive data would show on that window and the user may only want to show a portion of the sensitive data. Instead, the developer should provide custom tools for exposing individual pieces of sensitive data.

For instance, in the visiting student situation, the teacher wants to review only one student’s grades. On a public screen, the spreadsheet application could provide a custom context menu to allow the teacher to expose only a single row of data—the row containing that student’s grades. The spreadsheet would track exposed cells and protect all other cells containing sensitive data.

## 5.6. Developer summary

An important feature of XICE is the simplicity it affords developers when creating privacy-sensitive applications. This simplicity surfaces at the toolkit, window, recontext, and widget levels.

First, because overall privacy settings are tracked by the toolkit, developers do not need to add an overall privacy-tracking solution to their application and users have a consistent, central location for specifying the privacy state of a shared display.

Second, if a scene-graph is tagged with a @PrivateDialog annotation, then that scene-graph’s privacy behavior will be interpreted consistently wherever the scene-graph is used. Consequently a developer’s decision about privacy may be made once when the scene-graph for that dialog is designed, but enforced everywhere that scene-graph is used without rethinking about the privacy decision. Even developers who do not consider privacy when developing their applications can benefit. An application that uses the FileDialog scene-graph, or any subclass thereof, will automatically use that dialog in privacy-sensitive ways.

Third, because of the recontext event and independent privacy states for each window, developers can depend on getting and processing the recontext event before the scene-graph is serialized to the shared device. This ensures that no application-known sensitive data is transmitted to a public screen. If privacy is implemented externally to the UI distribution software then this guarantee is more difficult to ensure.

Finally, developers can wrap widgets with a FullPrivacy widget to protect that widget’s contents. If the developer so desires, he may integrate the FullPrivacy widget into the custom widget so that everywhere the custom widget is used its sensitive data is protected.

In addition, developers can create individual widgets which perform custom privacy protections, typically with little effort. For example, a spreadsheet application is available that uses XICE as its rendering framework. This application represents well over 200h of development

time, most of which involved writing spreadsheet functions. Privacy-aware shared UI features were added to this application after it was fully developed. The new privacy features required less than 8 h of another developer's time. The second developer was not involved in coding the spreadsheet application; hence, the 8 h include his time discussing the existing design with the original programmers. However, the second developer was very familiar with XICE's design (he designed XICE); hence, overhead related to the overall windowing toolkit was low. This time also included adding application-specific storage of 'private' tags to the spreadsheet's data file. The incorporated privacy-aware shared UI features allow rows, columns, or cells to be marked sensitive, and then sensitive rows and columns are hidden on public screens while sensitive cells are grayed out. The synchronized view in Figure 19 demonstrates this spreadsheet application in use.

## 6. USABILITY WALKTHROUGH

Evaluating the usability of toolkits is difficult. Part of this difficulty is because the toolkit is a large development effort. User evaluations provide tight data for controlled experiments with few variables, but toolkits have so many variables that exactly quantifying the effect of each feature on the overall experience is difficult [26]. Therefore, usability experiments are either trivial—collect conclusive data on a small feature—or cannot be generalized—too complex or require too many caveats to be useful. Because a usability experiment on XICE would be too complex to provide useful data, this paper does not include a user study but includes a usability walkthrough.

By doing a walkthrough, one can perceive the burden on a user and characterize the decisions that he must make. In particular, this walkthrough answers questions about when the user encounters the shared UI privacy features and how the user can choose to change privacy settings.

### 6.1. Home/work displays

Consider a user who just purchased a new XICE-enabled cell phone and who is connecting it to his home desktop screen for the first time. To connect to his home display he must configure the display using the wizard in Figure 10.

He could correctly click the 'Home/Work' button, or he could click the 'Public' or the 'Conference Room' button. Regardless of the choice the user makes, the new connection wizard in Figure 10 will not reshow when he connects to that display again. If the user correctly chooses the 'Home/Work' button, he will receive no further privacy settings or warnings, which is appropriate because he is using his home screen.

Let us assume that the user picks the 'Public' option. He then attempts to open a word processing document via his home display. Because the software is informed that the display server is distrusted, the XICE toolkit redirects the file dialog to the cell phone, similar to Figure 14.

It is likely that either now or at some future point (because sensitive dialogs are continually redirected to the personal device) the user will become annoyed and desire to change this setting. On the redirected dialog, as shown in the upper-right of Figure 22, XICE provides the 'Settings...' button which the user may click to change the privacy settings.

Clicking the 'Settings...' button shows the configuration wizard in Figure 10, again. At this point the user may choose to reconfigure the display as a 'Home/Work' display which causes the software to treat the display as full trust. From that time forward, the user will not see any protections for his sensitive data whenever connected to that display.

### 6.2. Corporate conference room

When the user first connects to the conference room at his work, he is presented with the configuration wizard. If the user chooses the 'Conference Room' option, he can show output on the display server and use the input from the display server. In addition, his sensitive information such as instant messages will not show on the shared display.

As the user discusses data within various applications, he can choose to push windows containing sensitive data to the shared display. He may also choose to show any embedded sensitive data

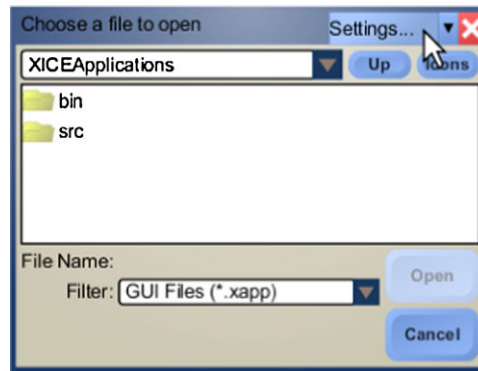


Figure 22. A redirected dialog presents the ‘Settings...’ button which the user may use to access and adjust the display’s privacy settings.

using the ‘Show Private Data’ menu option, or whatever other means the software may provide for overriding privacy settings. For example, if the user has a spreadsheet with sensitive data in it, he may share that window on the conference room screen. The spreadsheet will be shown and the sensitive data blocked from appearing. The user can then selectively expose sensitive data within rows, columns, or cells.

If, when configuring the connection, the user instead chooses the ‘Home/Work’ option, then he may be embarrassed by an e-mail or instant message, or may accidentally share sensitive information (e.g. in a spreadsheet). To change the settings so that he is better protected, the user may click the ‘Settings...’ option on any window he shared on display server (option 7 in Figure 20).

### 6.3. Foreign conference room

The user will likely visit other institutions and may be asked to give a presentation in their conference room. When the user annexes the display server in that conference room, he is shown the configuration wizard.

In the foreign conference room, the correct privacy selection for display server is ‘Public.’ Choosing ‘Public’ reduces the chances for the user’s device to be exploited if the display server is malicious (e.g. the maintenance staff is unaware that the display server is infected with malware). The user could choose to accept input from the display server, and the user is still protected from potentially malicious input.

If, on the other hand, the user chooses ‘Conference Room’ and the institution he is at is trustworthy, then the user is unlikely to be exploited. However, he could be exploited if the display server harbors malicious software. For this reason, the ‘Conference Room’ option includes a description discouraging its use when annexing foreign display servers.

Unfortunately, if the user chooses ‘Home/Work’ for the foreign display, then he may experience privacy violations. Instant messages, file dialogs, or other sensitive data could alert him to his poor choice, and he can subsequently change the privacy settings for that display. After presenting the configuration wizard, the windowing toolkit has attempted to inform a user and help him make good choices about how to connect to a display server. Further explanations may be made available (e.g. through hyperlinks), or the user may be provided with the opportunity to engage in further training to help him understand the privacy system, potential violations, etc. Such a training program may be a help manual, a game, or some other useful, but non-intrusive option. Raja *et al.* have done research showing promising ways to help inform users of the current firewall state and such research should be incorporated in final production dialog designs [27].

Currently people can sense better what kind of environment they are in, but they must be aware of the potential threats as well as opportunities. Some sort of training is necessary to help humans understand how to make appropriate privacy-related choices. When sensing devices improve to the point that the handheld can sense as well as, if not better than, a human then the portable

device may be able to assume the responsibility of making these choices. But with current sensing technologies, the toolkit can only present users with options and attempt to quickly inform users of the implications so that users may make informed decisions.

#### 6.4. *Airplane*

When traveling via airplane to another location, the user may encounter a display server embedded in the screen at his seat. Annexing that display server shows the connection wizard yet again. Although the airline is likely to have a maintenance staff which maintains the display server and keeps it free of malware, the environment is public; other passengers could see what the user is working on.

If the user chooses 'Public' or 'Conference Room' he will be properly protected. But if he chooses 'Home/Work' he will suffer from the same problems he could encounter if he chose 'Home/Work' at the foreign conference room.

#### 6.5. *Foreign displays*

The user will continue to encounter display servers in a variety of environments. Many, if not most, of these display servers will be owned by other institutions or in public environments. If the user, out of convenience, chooses to make each display a 'Home/Work' display, his user experience will be smooth in most cases. Unfortunately, he may encounter a malicious display server at some point and the toolkit will not prevent the shared-UI-specific exploits. Thankfully, if the user correctly chooses to make each subsequent display 'Public' he will be properly protected by the toolkit and his applications. The configuration wizard in Figure 10 provides a checkbox titled 'Treat all future screens as public and don't ask me again' which may be used to smooth the configuration process while keeping the user safe.

### 7. ALTERNATE IMPLEMENTATIONS

The XICE Framework is not the only way to implement a consistent privacy-aware shared UI framework that resolves the five key problems. This section will explore the requirements for implementing a privacy-aware framework in another windowing toolkit, such as Microsoft Windows, Java Swing, and Magic Lenses [28] or Attachments [29].

For the user, the solution must have two parts. He must be able to specify the privacy state of a shared device and whether he is comfortable showing a particular window. In order for the developer to support these two features, he must also have access to the privacy state and receive notifications when the privacy state changes. With these four parts in place, developers can create custom privacy-aware presentations for their software.

Personal devices that can annex at least one additional display can support a privacy-aware windowing toolkit. The toolkit must include a suitable UI distribution framework. This distribution could be provided by VNC, X11, RDP, etc. Ideally, the distribution framework would be simple and easy to implement for stand-alone display servers, allowing any personal device from any manufacturer to annex the display server and distribute UIs to it. XICE fills this requirement.

If the UI distribution framework is just a VGA cable, a privacy-aware framework is possible, but properly protecting windows with sensitive data may necessitate some constraints. Imagine a window that opens across two screens—one on the personal screen (private) and the other on the annexed screen (public). If widgets change size or position based on the display's privacy state, providing a consistent and understandable layout for windows that contain sensitive data becomes difficult. Constraining the window to either the public device or personal device is much easier. The windowing toolkit should ensure the window always has a single privacy setting, even if the window is partially on each display.

Once users can specify trust levels for devices and windows, the rendered output of applications needs to be augmented to protect privacy. If applications are privacy-aware, they may augment

themselves. However, if an application is not privacy-aware, a well-designed windowing toolkit can aid in protecting user privacy by blocking file dialogs.

Magic Lenses [28] offer a means for augmenting an application's graphical output without the application's direct knowledge. A lens is a virtual peephole or viewport over the application. Within that viewport, an augmented view of the application is exposed. This augmented view may make lines dotted, squiggly, or a different color, potentially to obscure sensitive data. Any rendering command can be augmented to yield a different appearance for an application.

Attachments [29] allow one application to attach data to the visual output of another application. The application constructing the attachment accepts the original rendering commands of the source application and then adds rendering commands around detected widgets, similar to Magic Lenses. During the attachment process original rendering commands may also be augmented.

Magic Lenses or Attachments could become means for implementing privacy awareness. Either of the tools would be used where a public screen has a full-screen lens that affects all applications shown on that screen. Applications would render themselves normally, and then the privacy lens would obscure potentially sensitive data (e.g. e-mail addresses, proper names, file paths, etc.).

One significant benefit of a lens-based system is that developers would not need to consider privacy when writing applications. As a result, however, developers may be discouraged or prevented from developing novel privacy solutions, in effect, hampering progress in this field. In addition, a side-effect of Magic Lenses and Attachments is that a widget's graphical output is basically the same size regardless of the lens—the lens has a difficult time effectively resizing widgets and drastically changing graphical output within the viewport while maintaining normal user interaction.

Of course, in each of the mentioned alternate systems, input from annexed devices to the personal device must also be treated according to the trust level of the device. Input would need to be tagged with the input source's trust level, so that applications can make appropriate decisions, like allowing, blocking, or confirming input on the personal device when input affects sensitive data.

## 8. CONCLUDING REMARKS

The proposed XICE privacy framework resolves the major issues involved in allowing users to annex shared devices. XICE demonstrably protects users from the five key privacy problems of stolen output, stolen input, false input, false output, and embarrassment. Vital to the success of the proposed solution is a simple content control system and information about whether a screen is private or public and its input is trusted or distrusted. Using this privacy information, the windowing toolkit and any applications running on it can prevent sensitive data from showing on public displays (by altering their appearance), discourage users from entering private information on public devices, and mitigate the effects of actively malicious devices that falsify user input or graphical output.

Users can manage their privacy needs using the privacy-aware framework, XICE. In particular, a foreign device is treated as public and distrusted by default. The user can change, as needed, the privacy state for that device, for windows shown on that device, and for privacy-aware widgets within those windows.

XICE provides a flexible, powerful privacy framework to developers. This framework requires less overhead for creating privacy-aware applications than building such applications via existing UI distribution frameworks. In particular, using annotations and the FullPrivacy widget allow developers to protect sensitive dialogs and widgets by designing that protection into the widget instead of implementing the protection code everywhere the dialogs or widgets are used.

Implementing privacy awareness does not need to be constrained to XICE, but should be part of any windowing toolkit that annexes shares devices. As people become increasingly mobile, the need to annex devices will grow, privacy will become an even greater concern, and a privacy-aware toolkit will become a necessity.

## REFERENCES

1. Scheifler RW, Gettys J. The X Window System. *ACM Transactions on Graphics* 1986; **5**(2):79–109. DOI: <http://doi.acm.org/10.1145/22949.24053>.
2. Tritsch B. *Microsoft Windows Server 2003 Terminal Services*. Microsoft Press, 2003.
3. Richardson T, Stafford-Fraser Q, Wood KR, Hopper A. Virtual network computing. *IEEE Internet Computing* 1998; **2**(1). DOI: <http://doi.acm.org/10.1145/22949.24053>.
4. Sharp R, Madhavapeddy A, Want R, Pering T. Enhancing web browsing security on public terminals using mobile composition. *Proceeding of the 6th International Conference on Mobile Systems, Applications, and Services, MobiSys '08*, Breckenridge, CO, U.S.A., 17–20 June 2008. ACM: New York, NY, 94–105. DOI: <http://doi.acm.org/10.1145/1378600.1378612>.
5. Arthur R, Olsen DR. XICE windowing toolkit: Seamless display annexation. *ACM Transactions on Computer–Human Interaction (ToCHI)* 2011.
6. Facebook. Available at: <http://www.facebook.com/> [October 2010].
7. MySpace, MySpace, Inc. Available at: <http://www.myspace.com/> [October 2010].
8. Howard M, LeBlanc D. *Writing Secure Code* (2nd edn). Microsoft Press, 2003.
9. Apple Computer. Macintosh. Apple Computer. Available at: <http://www.apple.com/mac/> [October 2010].
10. Izadi S, Brignull H, Rodden T, Rogers Y, Underwood M. Dynamo: A public interactive surface supporting the cooperative sharing and exchange of media. *User Interface Software and Technology (UIST '03)*. ACM: New York, 2006; 159–168. DOI: <http://doi.acm.org/10.1145/964696.964714>.
11. Want R, Pering T, Danneels G, Kumar M, Sundar M, Light J. The Personal Server: Changing the way we think about ubiquitous computing. *Ubiquitous Computing (UbiComp '02)*. Springer: Berlin, 2002; 223–230. DOI: [http://dx.doi.org/10.1007/3-540-45809-3\\_15](http://dx.doi.org/10.1007/3-540-45809-3_15).
12. Hawkey K, Inkpen KM. PrivateBits: managing visual privacy in web browsers. *Proceedings of Graphics Interface 2007 (GI 2007)*. ACM Press: New York, 2007; 215–223. DOI: <http://doi.acm.org/10.1145/1268517.1268553>.
13. Tarasewich P, Gong J, Conlan R. Protecting private data in public. *CHI '06 Extended Abstracts on Human Factors in Computing Systems (CHI 2006)*. ACM Press: New York, 2006; 1409–1414. DOI: <http://doi.acm.org/10.1145/1125451.1125711>.
14. Berry L, Bartram L, Booth KS. Role-based control of shared application views. *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology (UIST 2005)*. ACM Press: New York, 2005; 23–32. DOI: <http://doi.acm.org/10.1145/1095034.1095039>.
15. Berger S, Kjeldsen R, Narayanaswami C, Pinhanez C, Podlaseck M, Raghunath M. Using symbiotic displays to view sensitive data in public. *Third IEEE International Conference on Pervasive Computing and Communications (PerCom 2005)*. IEEE Computer Society: Silver Spring, MD, 2005; 139–148. DOI: 10.1109/PERCOM.2005.52.
16. Oprea A, Balfanz D, Durfee G, Smetters DK. Securing a remote terminal application with a mobile trusted device. *20th Annual Computer Security Applications Conference (ACSAC '04)*, 6–10 December 2004; 438–447. DOI: <http://doi.ieeecomputersociety.org/10.1109/CSAC.2004.33>.
17. Sharp R, Scott J, Beresford AR. Secure mobile computing via public terminals. *Proceedings of the International Conference on Pervasive Computing (PerCom 2006)*. IEEE Computer Society: Silver Spring, MD, 2006; 238–253. DOI: [http://dx.doi.org/10.1007/11748625\\_15](http://dx.doi.org/10.1007/11748625_15).
18. Yue C, Wang H. SessionMagnifier: A simple approach to secure and convenient kiosk browsing. *Proceedings of the 11th International Conference on Ubiquitous Computing (UbiComp '09)*, Orlando, FL, U.S.A., 30 September–3 October 2009. ACM: New York, NY, 125–134. DOI: <http://doi.acm.org/10.1145/1620545.1620566>.
19. GKS (Graphical Kernel System), ANS X3.124-1985 ANSI, December 1984.
20. Shuey D, Bailey D, Morrissey TP. PHIGS: A standard, dynamic, interactive graphics interface. *Computer Graphics and Applications* 1986; **6**(8):50–57.
21. Microsoft Corporation, Silverlight. Available at: <http://www.microsoft.com/silverlight/> [June 2010].
22. Petzold C. *Applications = Code + Markup: A Guide to the Microsoft Windows Presentation Foundation*. Microsoft Press, 2006.
23. Oracle Corporation. JavaFX. Available at: <http://www.javaafx.com/> [February 2010].
24. Apple Computer. Bonjour. Available at: <http://www.apple.com/support/bonjour/> [October 2010].
25. Bederson BB, Grosjean J, Meyer J. Toolkit design for interactive structured graphics. *IEEE Transactions on Software Engineering* 2004; **30**(8):535–546. DOI: <http://dx.doi.org/10.1109/TSE.2004.44>.
26. Olsen DR. Evaluating user interface systems research. *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology (UIST '07)*, Newport, RI, U.S.A., 7–10 October 2007. ACM: New York, NY, 2007; 251–258. DOI: <http://doi.acm.org/10.1145/1294211.1294256>.
27. Raja F, Hawkey K, Beznosov K. Towards improving mental models of personal firewall users. *Proceedings of the 27th International Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA '09)*. ACM: New York, NY, 2009; 4633–4638. DOI: <http://doi.acm.org/10.1145/1520340.1520712>.
28. Bier EA, Stone MC, Pier K, Buxton W, DeRose TD. Toolglass and magic lenses: the see-through interface. *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 1993)*. ACM Press, 1993; 73–80. DOI: <http://doi.acm.org/10.1145/166117.166126>.
29. Olsen DR, Hudson SE, Verratti T, Heiner JM, Phelps M. Implementing interface attachments based on surface representations. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 1999)*. ACM Press: 191–198. DOI: <http://doi.acm.org/10.1145/302979.303038>.