Groupware Support in the Windowing System

Peter Hutterer and Bruce H. Thomas

HxI Initiative - Project [braccetto]
Wearable Computer Laboratory
School of Computer and Information Science,
University of South Australia
Mawson Lakes SA 5095

{peter|thomas}@cs.unisa.edu.au

Abstract

In this paper, we discuss the advantages of integrating groupware support for Single Display Groupware (SDG) into the windowing system. For the domain of SDG, a Groupware Windowing System (GWWS) has several advantages over traditional SDG toolkits and applications. A GWWS provides SDG support for legacy applications, custom built SDG applications and supports the execution of multiple applications simultaneously. A GWWS combines the traditional single-user single-input axiom and novel multi-user multi-input desktop environments.

We present the Multi-Pointer X Server (MPX), the first GWWS that supports SDG natively, together with our Multi-Pointer Window Manager (MPWM). MPX and MPWM support an arbitrary number of true systems cursors, sophisticated floor control and per-window annotation overlay. To ease the interaction with such a GWWS, we implemented the DeviceShuffler, a system to couple input devices from any computer. The physical connection point of a device is transparent to both the windowing system and the application. This supports true ad-hoc collaboration on shared screens.

Keywords: Single Display Groupware, CSCW, Windowing Systems, Input Devices, GWWS

1 Introduction

Integrating groupware into the windowing system has been suggested previously (Lauwers 1990, Tse 2004), but no Groupware Windowing System (GWWS) has been reported. This paper presents the Multi-Pointer X Server (MPX), the first implementation of a general-purpose GWWS. MPX currently supports only Single Display Groupware (SDG) and mouse-like input devices, but we are developing mouse and keyboard support for Distributed Groupware and Mixed Presence Groupware. For the scope of this paper, we use GWWS to refer to a windowing system that supports Single Display Groupware.

SDG (Stewart 1999) is a well-investigated research domain. Various SDG applications (Bier 1991, Izadi 2003, Myers 1998) have demonstrated the use of SDG, but those applications are limited in functionality and not as feature-rich as commonly used commercial applications. Several SDG toolkits (Hourcade 1999, Hutterer 2006, Tse 2004) provide APIs to use multiple input devices and more sophisticated features such as floor control or on-screen annotations. However, the adoption of these toolkits has been slow and virtually non-existent. Different toolkits use different ways to access the hardware and provide different APIs to the developers. Many toolkits do not allow the execution of multiple SDG applications simultaneously. Running different applications that rely on different toolkits at the same time is complicated or even impossible.

In this paper we will discuss the advantages of SDG support in the windowing system and present our implementation of a GWWS. We believe that groupware and especially SDG needs to be supported natively in the windowing system to help adoption of real-time groupware. The windowing system is (apart from the operating system) the only environment that is used by every graphical application. SDG in the windowing system avoids many drawbacks of previous SDG applications and toolkits.

A GWWS for SDG provides the following functionality:

- the execution of well-established applications (legacy) in a multi-user context
- support for SDG applications,
- the execution of multiple applications at once,
- the execution of legacy and SDG applications simultaneously,
- SDG features to legacy applications, and
- the utilisation of well-established toolkits rather than being bound to a single toolkit.

If applications that are employed everyday (such as text editors, office applications and web browsers) can be executed in the same environment as feature-rich SDG applications, the threshold to the use of SDG applications will be lowered. In addition, when legacy applications are utilised in a multi-user context without any modifications, developers may get inspired to actively support SDG in future versions of their software. Domain-specific groupware features such as text merging have to be

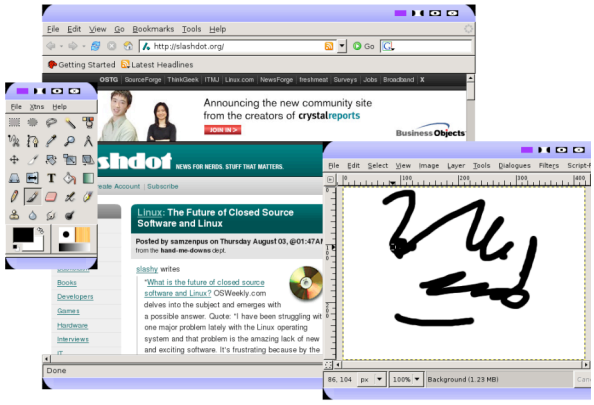

Figure 1. MPX running two legacy applications simultaneously.

implemented in the application and cannot easily be supported by a general purpose GWWS.

We have enhanced the X Window System (Scheifler 1986) to support up to 255 independent mouse cursors, testing has demonstrated up to 18 operating at the same time. The cursors can operate in multiple applications independently. MPX supports the simultaneous execution of multiple legacy applications (see Figure 1) and/or SDG applications. Users can interact with any number of applications at once. MPX provides a floor control mechanism on a per-window basis and thus each GUI element may have a different access rule.

The MPWM window manager facilitates MPX's SDG features and provides an administration interface for the floor control facilities. The MPWM actively supports multiple input devices for window operations such as moving and scaling windows. Using a window manager as administration interface for floor control ensures that legacy applications can utilise SDG features. The MPWM provides per-device annotation overlays for all top-level application windows.

In MPX, each connected device controls a distinct cursor. However, in many situations we need to employ input devices from a remote user on a different host. Our DeviceShuffler application forwards raw mouse events to any host on the network. The DeviceShuffler is located between the operating system and the device driver. This

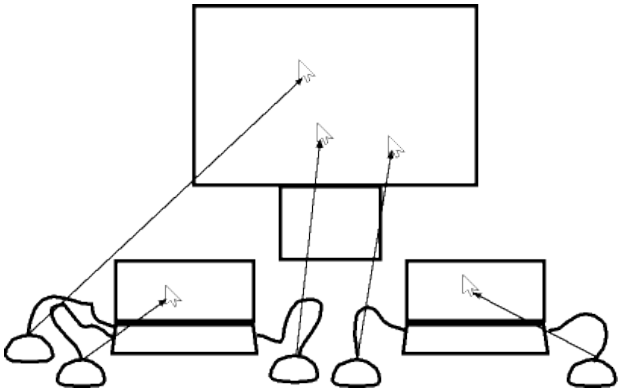

Figure 2. The DeviceShuffler makes the physical connection point insignificant.

makes the physical connection point of a device transparent to the windowing system, and thus to any application. Any mouse connected to any host computer on the network can be used to control any cursor on a shared screen (see Figure 2). MPX is implemented in the X Window System but the model of a GWWS is also applicable for the windowing systems of Microsoft Windows and Apple MacOS X.

2 Related Work

Using two hands is the natural mode people use to interact with the physical world. Stewart et al. (Stewart 1998) showed that users do not like to share input devices when working collaboratively. Collaboration in groups (e.g. during meetings) is common nowadays, but current graphical environments on desktop computers restrict users to only one pair of input devices (a mouse and a keyboard) at a time. In this section we present previous toolkits and applications that enabled collaboration on shared displays.

2.1 SDG toolkits and applications

Single Display Groupware (Stewart 1999) denotes applications that run on a single host computer but allow multiple users to interact with the application simultaneously. SDG applications and toolkits require the support for multiple input devices and often provide additional features.

MMM (Bier 1991) is one implementation of a multi-user editor that provides user dependent menus. MMM implements its own windowing system but cannot execute legacy applications. PEBBLES (Myers 1998) requires users to use PDAs as input devices. The PDAs provide private screens to the user. The PEBBLES environment cannot run arbitrary applications.

Dynamo (Izadi 2003) is a sophisticated content-viewing and data sharing environment. Dynamo supports simultaneous interaction and floor control but needs to communicate to programs using the COM, OLE or ActiveX interfaces. Applications that cannot be controlled by those interfaces, cannot be utilised by Dynamo.

The MIDDesktop (Shoemaker 2001) employs MID (Hourcade 1999) to receive input from different devices. Single-user Java applets can be executed in a multi-user environment. However, the MIDDesktop is limited to Java applets.

The SDGToolkit (Tse 2004) is a C# toolkit that provides high-level functionality such as cursor rotation for tabletop displays and parent classes for SDG widgets. As with all other toolkits, the functionality is limited by the windowing system supporting only one system cursor. The authors had difficulties dealing with the single system cursor and stated that "at some point, our windowing system should have SDG built into them as a fundamental component".

Wallace et al. (Wallace 2004) implemented a multi-cursor window manager where users could connect over a network to a shared display. The window manager would

display a different cursor for each connection. There is no support for multiple input devices on the shared display itself. The system cursor has to be time-sliced and moved to the position of the virtual cursor when an event occurs. This works for single-time events such as mouse moves and button events but does not work for events that require state information of the cursor. When multiple users try to drag a window simultaneously, the behaviour is unpredictable.

The CPNMouse (Westergaard 2002) driver enables multiple mice under Microsoft Windows but is limited to applications that are developed with the CPNMouse API. For all other applications, CPNMouse employs a time-sliced system cursor.

TIDL (Hutterer 2006) specifically targets legacy applications written in Java. TIDL's functionality operates across window and application boundaries. TIDL's main focus is support for distributed groups; it supports multiple input devices per host, giving each mouse device a cursor and each keyboard device its own keyboard focus. TIDL however requires applications to be developed with the Java Swing API. Finally, legacy applications have to be started up within TIDL to enable multiple user support.

2.2 Distributed Groupware

In the domain of Distributed Groupware (DG) several networked hosts computers share an application. DG requires the synchronisation and sharing of applications over the network and has to handle race conditions caused by network delays. Although multiple users interact with DG applications simultaneously, each host computer only has one pair of input devices, a keyboard and a mouse. Numerous systems and toolkits have been proposed that allow users to share an application amongst multiple host computers. However, those systems only allow one user to interact with an application at a time or require the applications to be built against the toolkits API. XTV (Abdel-Wahab 1991) or XMX[1] suffer from the X Window System's single cursor axiom. MMConf (Crowley 1990), SharedX (Garfinkel 1994) and Rapport (Ahuja 1988) only allow input from one user at a time. GroupKIT (Roseman 1992) and Rendezvous (Patterson 1990) rely on applications being built against the toolkit's specific API. Virtual Network Computing (VNC) (Richardson 1998) uses the underlying windowing system to generate events and is thus limited to a single pair of input devices.

2.3 Remote Input Devices

The x2x[2] tool sends high-level events using the X Protocol to control the mouse and keyboard on a remote computer running the X Window System. It is limited to forwarding a single cursor's movements and it is not possible to distribute input events from only one specific device.

Synergy[3] is a common Open Source program to control different host computers with only one keyboard and one mouse. Synergy works across different platforms but relies on the windowing system to receive and distribute the cursor coordinates. A mouse can only control one cursor on one display at a time, and there is no distinction between different devices. Synergy only supports a single system cursor and two mice connected to the same display move the same system cursor.

PointRight (Johanson 2000) connects to several displays at once and lets users define a complex display geometry that resembles the physical layout of the displays. A cursor can then be moved from one display to another display. PointRight supports multiple cursors on different displays, but it is "restricted by the OS to single cursor control per machine" (Johanson 2000). If multiple users connect to the same display simultaneously, PointRight averages the cursor movements from all devices.

None of these tools support a fully ad-hoc per-device configuration to assign a specific device to a specific cursor at any point in time.

3 Native support for SDG

Up to date, no detailed discussion or implementation of a GWWS has been presented. We propose that a GWWS has to provide support for the following domain-independent groupware features:

1) an arbitrary number of independent input devices,
2) simultaneous interaction of multiple input devices,
3) fine-grained floor control,
4) differentiation of legacy and SDG applications, and
5) general-purpose annotation overlays.

The following sections explain why we believe that a GWWS is the correct place to integrate SDG functionality.

3.1 SDG in the Windowing System

A GWWS has several advantages over SDG toolkits and SDG applications. All graphical applications rely on the windowing system to display the GUI and receive user input events. Applications and toolkits rarely know about physical input devices. They depend on the event stream from the windowing system to inform them about state changes. For an application there is no cursor moving across the screen, as a user would perceive. Instead, this movement is expressed as a set of events with the cursor's current coordinates as it changes its position. In a similar manner, there is no keyboard device but rather a series of events containing information about the pressed or released keys. If the windowing system is modified to support multiple input devices, applications and toolkits are unaware of this support if the semantics of the event

[1] http://www.cs.brown.edu/software/xmx/

[2] http://x2x.dottedmag.net/

[3] http://synergy2.sourceforge.net/

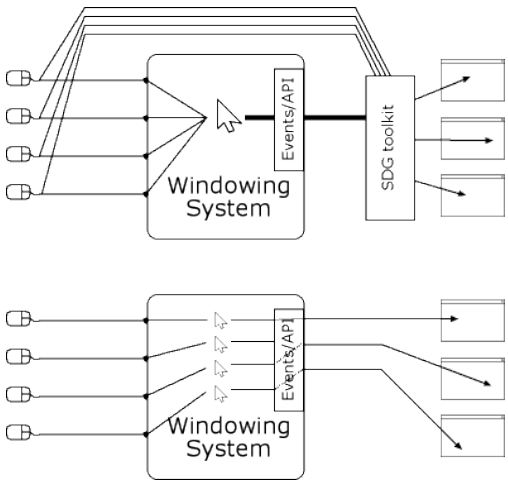

Figure 3. A toolkit has to intercept the windowing system's event and emulate them. This can be avoided by integrating SDG into the windowing system.

streams are maintained. This provides compatibility with legacy applications.

The windowing system generates events for each device as it changes states. If SDG is provided by a toolkit, these events have to be filtered and discarded (see Figure 3) and events from virtual devices have to be emulated. The emulation of such events is complex, time-consuming and often prone to errors. To emulate events, the toolkits have to access the hardware directly. This can lead to race conditions if multiple toolkits are running simultaneously.

A windowing system can filter events more effectively than a toolkit. Event filters applied directly in the windowing system will result in no event being sent to the application at all. Such filters are useful for floor control.

Integrating SDG into the windowing system is attractive to developers of future SDG applications. Rather than having to employ a new toolkit they can test existing applications with a new set of features. Only if they feel the need to integrate support for SDG features directly in the application they are required to alter the code. Doing so unveils a larger set of functionality and the prospect of developing innovative user interfaces.

3.2 Administering SDG in the window manager

Contemporary windowing systems consist of a framework for the display of a GUI, a framework to gather user input events, and a window manager. The window manager is responsible for the placement of application windows, the window decoration, and window visibility. The window manager can be tightly integrated into the windowing system (Microsoft Windows, Mac OS) or run as a separate application (X Window System). An application can suggest window settings to the window manager, but ultimately every application relies on the window manager to position the GUI on the screen.

We believe that the window manager is the ideal place to administer the functionalities provided by a GWWS. The window manager can enforce SDG features better than a toolkit could. Applications are able to communicate with the window manager and can direct the window manager to leave SDG functionality to the application. The window manager can activate certain SDG functionality for legacy applications. Ultimately, the window manager controls which SDG features to enable or disable per application window.

A window manager can provide a desktop environment that provides users with icons and/or menus to start applications or alter global settings. Such menus can be used to introduce hierarchies to input devices (for example, a project manager could be able to override a employees floor control during a meeting) or to pair input devices if the number of mouse input devices differs to the number of keyboard input devices. By using the edges and corners of the screen, the window manager could impose quick token passing or activate screen-wide overlays.

3.3 Distributed input devices

Connecting a multitude of input devices to a single host computer may not be adequate. In a meeting attendees may want to access a shared whiteboard from their laptops, in a teaching situation a teacher may want to access a student's computer without having to connect an additional input device first or in a office environment a task may require short ad-hoc collaboration. In all those settings, a multitude of computers are readily available.

Previous projects (Foster 1986, Wallace 2004) allowed access to shared whiteboards but without great flexibility. The reliance on the windowing system's API imposed the restriction of a single cursor per display. An ad-hoc configuration to forward specific input devices was not possible. We believe that this ad-hoc configuration is necessary for the efficient use of a GWWS. Users and thus input devices can be very transient. Especially host computers that drive shared display surfaces in meeting rooms are used by several different users per day.

We propose that the physical connection point of devices has to be insignificant. Any device that is connected to a host computer should be able to control a specific cursor and/or keyboard in a GWWS. Furthermore, the device routing should be reconfigurable at any time, making it possible to control several cursors with only one physical device.

Such a dynamic routing of devices can introduce another level of floor control in addition to the windowing system's floor control mechanism. Some devices may not be allowed to connect to a machine at all for a given time. Per-device authentication allows cursors to be assigned to specific devices. A user could then bring an input device to a meeting and always control one specific cursor.

4 Implementation

Our GWWS implementation consists of three parts: the Multi-Pointer X Server (MPX), the Multi-Pointer Window Manager (MPWM) and the DeviceShuffler. This

section will discuss the features of MPX, MPWM and the DeviceShuffler from a high-level perspective.

4.1 MPX

MPX is a modification of the existing X Server release X11R7.1[4]. The standard implementation of the X Server supports multiple input devices but merges all input from pointing devices into one single system cursor (the "core pointer"). All mouse events originate from this single cursor only.

MPX modifies this behaviour and provides each device with an independent core pointer. Mouse events originate from each cursor as it changes state. This is significantly different to previously published systems that time-slice the single system cursor, warping the single system cursor to the position of a virtual cursor before an event could be generated. Each cursor in MPX is a true system cursor.

MPX supports two parts of the X Protocol: the core X Protocol and the X Input Extension. A core protocol mouse event does not provide any methods to identify the device that caused the event. Most legacy applications employ the core protocol to deal with user input. MPX cursors also generate X Input Extension events. These events feature a device ID field to uniquely identify the device. The X Input Extension API is well established and known to X developers. Future SDG applications need to employ the X Input Extension API to explicitly support multiple input devices.

The modified core and the X Input Extension events of MPX differ only slightly to the original X events. MPX events contain the coordinates of the cursor that caused the event. Other cursor-specific values (such as the underlying window) are modified to be valid for the specific cursor. This has been achieved by modifying the event generation inside the X Server to keep track of the different device's individual states. MPX follows the X Protocol semantics and is thus compatible to standard X applications.

MPX features a fine-grained floor control mechanism. Each window maintains Access Control Lists (ACLs) containing the devices that are allowed to access a window. In the X Window System, every visible window component is a window. We can thus assign different permissions to any GUI element, such as an entire application window, buttons, checkboxes, radio buttons

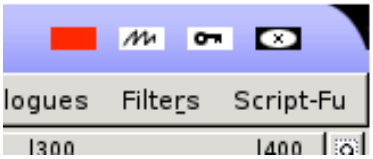

Figure 4. SDG buttons in the window manager allows to easily access overlay (center left) and floor control (center right) for any window.

[4] ftp://ftp.x.org/pub/X11R7.1

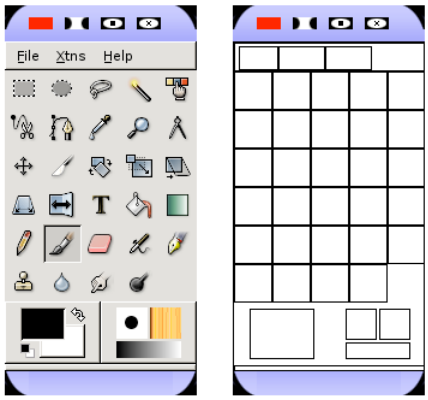

Figure 5. The MPWM rebuilds a wire-mesh model of the GUI to assign ACL to the GUI elements.

or even menus. For example, the "File" menu can be restricted to only allow one user to open a new document, save or quit the application. Each window has two ACLs: DENY and PERM. The former contains devices that are explicitly denied access to the window. The latter contains devices that are explicitly allowed access to the window, thus implicitly denying access for any other device not in the list. Internally, MPX checks each event for the destination window and whether the device that caused the event is permitted to access the window. If a device is denied access, no event is sent to the application. Such a floor control mechanism in the windowing system does not require the application's support. The ACLs are modified by setting X Atoms ("MPG WINDOW PERM" and "MPG WINDOW DENY") on the GUI elements.

In the future, we plan to support sub-window floor control in the windowing system. Rather than supplying ACLs for each GUI element, each device can also be assigned a region within the GUI element that it is allowed to access. A canvas in a drawing program can then be split in multiple areas with different access rights.

4.2 MPWM

The Multi-Pointer Window Manager (MPWM) is the first application to actively use the SDG features of MPX. The MPWM supports interaction with multiple cursors simultaneously, and it supports: 1) the movement of several application windows at once, 2) the scaling of application windows while other windows are moved, and 3) the scaling of application windows with multiple mice at once. The MPWM provides an interface to the MPX floor control mechanisms. Each top-level application window can be locked to a single cursor by left-clicking the floor control button (see Figure 4) in the window decoration. No other cursor can then move, scale or access the application window. Right-clicking on the floor control button brings up a wire-mesh model of the applications user interface (see Figure 5). The user can restrict other users from access to a single GUI element and its sub-elements by clicking on the wire-mesh model. For example, if the eraser button in Figure 5 is restricted to a single cursor, and no other cursor can activate the eraser.

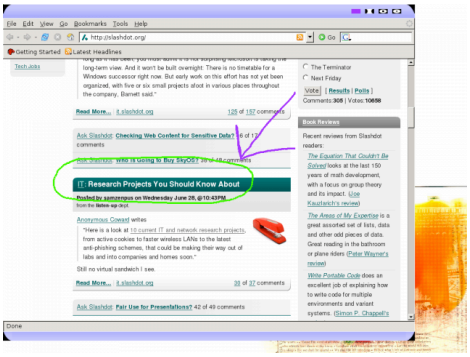

Figure 7. The MPWM provides annotation overlay for top-level application windows.

The MPWM provides annotation overlays for each visible top-level application window (see Figure 7). Annotations are user-specific and independent of the underlying application. The annotations are relative to the position within the window, and they stay properly aligned when the window is moved. Each cursor has a distinctively coloured layer on the annotation. While one user annotates, other users can access the application underneath. An annotation does not affect the application and when the annotation layer is active for one cursor, no events originating from this cursor are sent to the underlying application. Window manager annotations do not know the semantics of the underlying application and thus do not change when the window is resized.

The MPWM is designed as a reference implementation of how a window manager could employ MPX. It does not have the sophisticated features (skins, configurable keyboard shortcuts, intelligent window placing) of other contemporary window managers such as metacity, kwin or fluxbox. MPX is compatible with standard window managers, when MPWM is not used.

4.3 The DeviceShuffler

Previous implementations of systems to provide remote input devices used the windowing system's API to deliver events to a host computer. Due to ambiguities in the API (see Section 5.2), a lower-level approach is required.

The DeviceShuffler is situated below the device driver level and effectively inserts a TCP/IP layer between the operating system's representation of the device and the X Window System's device drivers. The windowing system usually accesses the physical devices as provided by the operating system. In our configuration, the DeviceShuffler provides virtual devices in the form of named pipes that are accessed by the X Window System. The DeviceShuffler then forwards the events from the operating system's devices to a virtual device, thus providing a transparent layer (see Figure 6). Such a virtual device can be connected to any physical device on any host computer separated on the network. A cursor can thus be controlled by a device on a remote machine. The DeviceShuffler supports N:1, 1:N, N:N and N:M mouse to cursor configurations. Several mice can be merged into a single virtual mouse device and control a single cursor (N:1 configuration). A 1:N mapping allows one mouse to be multiplexed into several cursors; for example this could be used in a teaching environment to replicate the teacher's actions on each student's computer. The DeviceShuffler allows for a N:N configuration where each mouse controls exactly one cursor. Finally, the DeviceShuffler allows for a N:M configuration, where each mouse is merged and/or multiplexed into any number of virtual devices.

The DeviceShuffler distinguishes between the devices, thus allowing each device to be forwarded separately and to a different host computer. For example, users could decide to forward the touchpad of their laptop to control a cursor on a shared display while a mouse connected via USB would continue to control the local cursor on the laptop. Figure 2 shows an example of how mice could control multiple screens.

Situating the DeviceShuffler below the driver level also allows for dynamic reassignment of a cursor without the windowing system noticing. A virtual device can be disconnected from a physical device and reconnected to another physical device at any time without disrupting the windowing system's event flow. Moreover, multiple physical devices can be merged into one virtual device. This ensures that the DeviceShuffler is compatible to X servers without MPX functionality.

5 Challenges

Introducing SDG into a windowing system unveils several challenges. There is no hardware support for multiple cursors in current graphics cards, current windowing system's APIs become ambiguous with the introduction of multiple cursors, and some toolkits and applications cause race conditions when operated with multiple mice. The challenges described here are not restricted to our MPX implementation, but universal to any windowing system that is to include support for multiple cursors.

5.1 Hardware support

Contemporary graphics cards render the system cursor in hardware to speed up the display. The windowing system writes the cursor's coordinates directly to the graphics hardware, the graphics card then writes it into the output stream. However, an investigation of graphics card drivers and driver developers suggests that graphics cards hardware is limited to a single system cursor and cannot render an arbitrary number of mouse cursors to the screen. The windowing system thus has to use software

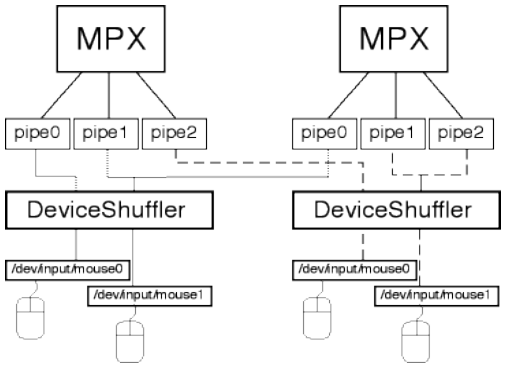

Figure 6. Each physical device can be connected to a local and/or remote pipe.

rendering to display the cursors. Software rendering however is less efficient than hardware based rendering and can cause the cursors to visually flicker.

5.2 Ambiguous APIs

Current windowing system APIs are limited to single cursors and some functions become ambiguous with the introduction of multiple cursors. Some applications modify or query the cursor. When multiple cursors are provided by the windowing system, such a query is ambiguous. To maintain compatibility with legacy applications, a multi-cursor windowing system needs to decide which cursor should be selected. There are three options:

1) to always select a specific cursor (e.g. the first cursor),
2) to select the cursor closest to or inside of the application window, or
3) to assign ownership of the application to a particular cursor and select this cursor.

The first option introduces a hierarchy of cursors and is thus inadequate. This begs the question, "*Who determines the hierarchy?*" In previous research prototypes (e.g. (Tse 2004, Wallace 2004)) the system cursor was a first-class object and the virtual cursors provided by the toolkit were second-class ones.

The second option may lead to ambiguous choices. Two cursors could be inside an application window or the cursor closest to an application window may be in use by another application. There may not be a correct decision.

The third option requires the implicit or explicit assignment of ownership to an application. An implicit assignment of ownership can be difficult if multiple users interact with the application simultaneously. An explicit ownership requires direct interaction of the users and furthermore knowledge of the users whether this ownership is necessary for a given application.

These ambiguities occur at the function calls to change a cursor shape, to warp a cursor to a given position, to query the cursor's current coordinates or to artificially issue a cursor event.

5.3 Application race conditions

Legacy applications were not designed with multiple users in mind. Interacting with multiple cursors in such an application may lead to race conditions where the application refuses to interpret events or exhibits wrong behaviour.

Some legacy applications will experience strong "cursor jumps". For example an application that listens to mouse motion events will assume that the cursor jumps around on the screen when multiple mice are moved simultaneously. In some applications, this can lead to erratic behaviour. We have limited the effects of this behaviour. For example, if a cursor is dragging inside a window, other cursors do not send motion events to this window. Only mice that also drag will send events. It is thus possible to use drag-and-drop in legacy applications.

Context menus can experience race conditions. If one cursor brings up a right-click context menu, another cursor can "steal" this context menu by right-clicking or just disable it by left-clicking.

Finally, erratic behaviour can be observed with applications listening for mouse enter and leave events. Mouse leave events are event that are sent to notify an application that a cursor has left a particular area. Mouse enter events on the other hand notify an application that a mouse has just entered a specific area on the screen. Naturally, two mice entering an application window consecutively will cause two mouse enter events to be sent. Upon the first mouse leaving, a mouse leave event is issued and a legacy application will assume that no mouse is inside the application's window. In some cases, the second cursor will then not be able to interact with the application. This behaviour has been observed with the xterm terminal application but also with the GTK toolkit where a button cannot be clicked after one mouse has left the button's window.

6 Future work

MPX currently modifies the X Input Extension's behaviour. In the future we plan to add a MPX extension to the X Window System to make SDG features easier accessible from client applications. We currently investigate improved user interfaces to administer the powerful floor control functionality built into MPX. We plan to support multiple independent keyboards in addition to the currently supported multiple pointing devices. In this paper, we focused on GWWS for the domain of SDG. We are working on MPX view distribution to convert MPX to a GWWS that supports not only SDG but also Distributed Groupware and Mixed Presence Groupware. Finally, further investigations will be made to resolve the challenges described in Section 5.

7 Conclusion

In this paper we outlined why SDG needs to be supported in a Groupware Windowing System rather than in toolkits or in SDG applications themselves. The windowing system is used by virtually all graphical applications and can thus provide SDG functionality to all those applications. A GWWS allows for the simultaneous execution of multiple SDG applications but also the usage of legacy applications in an SDG manner. SDG for legacy applications may be administered by the window manager. The window manager can enforce floor control or annotation overlays on any top-level application window.

We presented our implementation of the Multi-Pointer X server (MPX), a modification of the X Window System to support an arbitrary number of system cursors and elaborate floor control to restrict devices from using the GUI. This floor control can be applied to any GUI element such as buttons, checkboxes and windows. For legacy applications, our Multi-Pointer Window Manager provides an administration interface to configure the Access Control Lists for the GUI elements. Finally, we presented the DeviceShuffler, an application that allows

connecting any mouse on the network to a particular cursor in an MPX setup. This allows for ad-hoc collaboration with shared displays. The DeviceShuffler hides the physical connection point of an input device from the windowing system.

8 Acknowledgements

We would like to give our thanks to the ViCAT project team, in particular Benjamin Close, Julien Epps and Masahiro Takatsuka. We also acknowledge DSTO and NICTA for funding the project.

9 References

Abdel-Wahab, H. M. and Feit, M. A. (1991): XTV: a framework for sharing X Window clients in remote synchronous collaboration. In *Proceedings of the TRICOMM 91,* pp 159-167, April 1991.

Ahuja, S. R., Ensor, J. R., and Horn, D. N. (1988): *The rapport multimedia conferencing system.* SIGOIS Bull., Vol. 9, No. 2-3, pp 1-8, 1988.

Bier, E. A. and Freeman, S. (1991): MMM: a user interface architecture for shared editors on a single screen. In *Proc's of the 4th annual ACM symposium on User interface software and technology,* pp 79-86, Hilton Head, South Carolina, United States, 1991.

Crowley, T., Milazzo, P., Baker, E., Forsdick, H., and Tomlinson, R. (1990): MMConf: an infrastructure for building shared multimedia applications. In *Proceedings of the 1990 ACM conference on Computer-supported cooperative work,* pp 329-342, Los Angeles, California, United States, 1990.

Foster, G. and Stefik, M. (1986): Cognoter: theory and practice of a colab-orative tool. In *Proceedings of the 1986 ACM conference on Computer-supported cooperative work,* pp 7-15, Austin, Texas, 1986.

Garfinkel, D., Welti, B. C., and Yip, T. W. (1994): *HP SharedX: A Tool for Real-Time Collaboration.* HP Journal, 1994.

Hourcade, J. P. and Bederson, B. B. (1999): *Architecture and Implementation of a Java Package for Multiple Input Devices (MID).* University of Maryland, Report No. CS-TR-4018, 1999.

Hutterer, P., Close, B. S., and Thomas, B. H. (2006): TIDL: Mixed Presence Groupware Support for Legacy and Custom Applications. In *Proceedings of the seventh conference on Australasian user interfaces,* pp 107-114, Hobart, Australia, 2006.

Izadi, S., Brignull, H., Rodden, T., Rogers, Y., and Underwood, M. (2003): Dynamo: a public interactive surface supporting the cooperative sharing and exchange of media. In *Proceedings of the 16th annual ACM symposium on User interface software and technology,* pp 159-168, Vancouver, Canada, 2003.

Johanson, B., Hutchins, G., and Winograd, T. (2000): *PointRight: A System for Pointer/Keyboard Redirection Among Multiple Displays and Machines.* HCI Group, University of Stanford, Report No. CS-2000-03, 2000.

Lauwers, J. C. and Lantz, K. A. (1990): Collaboration awareness in support of collaboration transparency: requirements for the next generation of shared window systems. In *CHI '90: Proceedings of the SIGCHI conference on Human factors in computing systems,* pp 303-311, Seattle, Washington, United States, 1990.

Myers, B. A., Stiel, H., and Gargiulo, R. (1998): Collaboration using multiple PDAs connected to a PC. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work,* pp 285-294, Seattle, Washington, United States, 1998.

Patterson, J. F., Hill, R. D., Rohall, S. L., and Meeks, S. W. (1990): Rendezvous: an architecture for synchronous multi-user applications. In *Proceedings of the 1990 ACM conference on Computer-supported cooperative work,* pp 317-328, Los Angeles, California, United States, 1990.

Richardson, T., Stafford-Fraser, Q., Wood, K. R., and Hopper, A. (1998): *Virtual Network Computing.* IEEE Internet Computing, Vol. 2, No. 1, pp 33-38, 1998.

Roseman, M. and Greenberg, S. (1992): GROUPKIT: a groupware toolkit for building real-time conferencing applications. In *CSCW '92: Proceedings of the 1992 ACM conference on Computer-supported cooperative work,* pp 43-50, Toronto, Ontario, Canada, 1992.

Scheifler, R. W. and Gettys, J. (1986): *The X window system.* ACM Trans. Graph., Vol. 5, No. 2, pp 79-109, 1986.

Shoemaker, G. B. D. and Inkpen, K. M. (2001): *MIDDesktop: An Application Framework for Single Display Groupware Investigations.* School of Computing Science, Simon Fraser University, Report No. TR 20001-01, April 2001.

Stewart, J., Bederson, B. B., and Druin, A. (1999): Single display groupware: a model for co-present collaboration. In *Proc's of the SIGCHI conference on Human factors in computing systems,* pp 286-293, Pittsburgh, Pennsylvania, United States, 1999.

Stewart, J., Raybourn, E. M., Bederson, B., and Druin, A. (1998): When two hands are better than one: enhancing collaboration using single display groupware. In *CHI '98: CHI 98 conference summary on Human factors in computing systems,* pp 287-288, Los Angeles, California, United States, 1998.

Tse, E. and Greenberg, S. (2004): Rapidly prototyping Single Display Groupware through the SDGToolkit. In *Proc's of the fifth conference on Australasian user interfaces,* pp 101-110, Dunedin, New Zealand, 2004.

Wallace, G., Bi, P., Li, K., and Anshus, O. (2004): *A Multi-Cursor X Window Manager Supporting Control Room Collaboration.* Princeton University, Computer Science, Report No. TR-0707-04, July 2004.

Westergaard, M. (2002): Supporting Multiple Pointing Devices in Microsoft Windows. In *Microsoft Summer Workshop for Faculty and PhDs,* Cambridge, England, September 2002.