

Enabling Co-located Ad-hoc Collaboration on Shared Displays

Peter Hutterer and Bruce H. Thomas

HxI Initiative – Project [braccetto]
Wearable Computer Labs
School of Computer and Information Science
University of South Australia
Mawson Lakes 5095, South Australia
{peter|thomas}@cs.unisa.edu.au

Abstract

All major desktop environments are designed around the assumption of having a single system cursor and a single keyboard. Co-located multi-user interaction on a standard desktop requires users to physically hand over the devices. Existing collaboration applications require complicated and limiting setups and no collaboration application or toolkit supports ad-hoc transition from a traditional single-user desktop to a multi-user collaboration environment without restarting applications.

Our Multi-Pointer X server (MPX) allows easy transition between a single-user desktop and a multi-user collaboration environment. Pointer devices and keyboards can be added and removed at any time. Independent cursors and keyboard foci for these devices allow users to interact with and type into multiple applications simultaneously. MPX is compatible with any legacy X application and resolves ambiguity in legacy APIs using the novel “ClientPointer” principle. MPX also provides new APIs for multi-user applications and thus enables fluid integration of single-user and multi-user environments.

Keywords: CSCW, GWWS, windowing systems.

1 Introduction

Collaboration is often spontaneous and unstructured, and this is particularly true when colleagues in the work place ask for impromptu help. When the colleagues then focus on a task that is represented on a traditional workstation, they need to collaborate on a single screen with a single set of mouse and keyboard devices. This restriction requires the users to pass the physical devices between themselves.

Several collaborative applications and toolkits have been introduced in the past (Hourcade 1999, Hutterer 2006, Izadi 2003, Tse 2004), focusing on planned and structured collaboration. Our Multi-Pointer X (MPX)

turns the desktop into a single display groupware environment, with the option of adding devices on the fly. This groupware environment can run traditional single-user applications and multi-user aware applications simultaneously (Hutterer 2007b).

MPX is not a collaboration application. The collaboration features are integrated into the windowing system, allowing MPX to provide ad-hoc collaboration across the whole desktop as well as collaboration within a single application. This low-level integration of these features makes the technology available to any graphical application, regardless of application support. Multi-user functionality can be utilised in traditional single-user applications as well as novel multi-user software.

Imagine a scenario where a user, Bob, is required to collect information about a specific topic. Bob uses MPX as his windowing system. After a few hours of browsing the web he asks Alice for help. At this point in time, Bob has two browser windows and a word processor open on his desktop. To help Bob, Alice needs to interact with Bob’s desktop. She connects a mouse to Bob’s host computer and a second cursor appears. Alice browses the web in one of the browser windows using her mouse. Bob continues to use his mouse and keyboard to write the summary. Bob can also interact with the same browser window or any other window at the same time. When Bob’s question has been answered, Alice disconnects her mouse and Bob goes back to working in single-user mode.

This scenario highlights the benefits of support for multiple independent devices across multiple legacy applications. It also shows how a smooth transition between a traditional single-user desktop and a collaboration environment enables ad-hoc collaboration.

MPX is currently the only technology that makes the above scenario possible. Traditional collaboration applications (Bier 1991, Izadi 2003) and toolkits (Hourcade 1999, Hutterer 2006, Tse 2004) provide users with features such as multiple input devices, floor control and tools to increase awareness. At the same time, they restrict the users’ ability to interact with traditional desktop applications. Only a limited set of applications at a time can provide groupware features. Outside of these applications, multi-user interaction is not possible. A simple task such as reading emails can require a shutdown of the collaboration environment.

MPX is a modification of the X.org X server¹ to support multiple cursors and multiple keyboard foci. MPX is compatible with any application that runs on a current X server. In fact, if there is only one pair of input devices connected, MPX is identical to a standard X server. We can transition between a traditional single-user desktop to a multi-user environment just by adding or removing input devices. Users can continue to use well-established applications and even their standard desktop environments (e.g. GNOME). MPX is an important step in the transition from the single-pointer single-keyboard paradigm that dominates current user interfaces to true multi-user environments.

A detailed description of MPX has been presented elsewhere (Hutterer 2007b). This paper presents recent enhancements to MPX to provide multiple independent keyboard foci, support the dynamic addition of cursors and keyboard foci for hotplugged devices, and resolve ambiguities in single-user APIs. These enhancements allow fluid transition from single-user to multi-user mode without restarting any applications. We will also present two prototype applications that demonstrate how the multi-user features in MPX can be utilised. In Hutterer and Thomas (Hutterer 2007a), a brief overview of these concepts was presented, whereas this paper provides a detailed descriptions of the concepts and implementations issues.

2 Related Work

The benefit of multiple input devices is well researched. Stewart et al. (Stewart 1998) found a preference towards individual input devices when users collaborate using Single Display Groupware (SDG). Other research showed that domination of one user is less prevalent when users have equal access to a user interface (Inkpen 1997, Stanton 2003). Pawar et al. found that one mouse per user can improve children's ability to memorise words (Pawar 2007), while at the same time reducing boredom and distraction. Grudin stated that support for everyday applications is a requirement for the adoption of groupware (Grudin 1994), yet few groupware toolkits allow collaboration in legacy applications on a single display.

Several applications and toolkits focus on enabling multiple input devices. MMM (Bier 1991) was an early system to support four simultaneous users on a single display. SDGToolkit (Tse 2004) and MID (Hourcade 1999) are high-level toolkits in C# and Java for groupware applications. They are targeted towards new applications that need to support multiple users. The SDGToolkit has been successfully used at a university course to create novel applications (Greenberg 2007). However, the SDGToolkit does not support the execution of multiple collaborative applications simultaneously. The MIDDesktop (Shoemaker 2001) utilised MID to provide a desktop-like environment to execute several Java applets simultaneously. TIDL (Hutterer 2006) requires legacy Java applications to be started by a

custom application loader to provide them with a multi-user context. TIDL cannot provide multi-user functionality to already running applications. Both MIDDesktop and TIDL cannot utilise applications not written in Java. Dynamo (Izadi 2003) provides a multi-user environment focusing on media sharing. Legacy applications are supported through inter-process communication (IPC) mechanisms. The Dynamo desktop is focused around multiple users interacting with media assets, but the dependency on IPC restricts the use of those applications that do not support those interfaces.

Focusing only on multi-user interaction is a common caveat of groupware applications and toolkits. Instant transition between a single-user and multi-user environment without the need to restart any applications is not supported by any current technology other than MPX.

On the other hand, multi-touch hardware is becoming more popular. The e-Beam pen devices (<http://www.e-beam.com>) allow two pens to be used simultaneously. Han (Han 2005) presented a multi-touch display technology based on frustrated internal reflection. This technology supports a theoretically unlimited number of touch points at a very low cost. Wilson (Wilson 2004) used stereo cameras to detect touch-points on a transparent touch screen. The absence of a diffuser allows for high-resolution images and the possibility of user identification through face recognition. The DiamondTouch (Dietz 2001) technology supports user identification using the body's electrical resistance. The DiamondSpin toolkit (Shen 2004) was developed for use with the DiamondTouch and allows applications to freely rotate their windows. DiamondSpin was for example used by the UbiTable (Shen 2003) to allow ad-hoc sharing and viewing of documents stored on users' laptops. When they walk up to the DiamondTouch table, their laptop connects to the software running on the tabletop display and provides them with areas to view and exchange their private data. DiamondSpin provides Java APIs for novel applications and can support legacy applications if they are started from within a specific context. Applications not written in Java are not supported.

The Soap input device (Baudisch 2006) is a modified wireless mouse that can be held and controlled in one hand. The wireless technology and indirect nature of the device easily allows for multiple users to interact simultaneously with a shared whiteboard. Yet these users would be restricted by the windowing system to a single system cursor. The everyday desktop remains restricted to a single user.

MPX is the first GroupWare Windowing System (GWWS) (Hutterer 2007b). Traditional groupware toolkits and applications had to access the hardware directly, causing race conditions between different toolkits. A GWWS provides a unified way for toolkits and applications to access multi-user functionality. Previously, we described how MPX supported multiple independent system cursors, an extensive floor control mechanism and annotation overlays in connection with our Multi-Pointer Window Manager (MPWM). (Hutterer 2007b)

¹ <http://www.x.org>

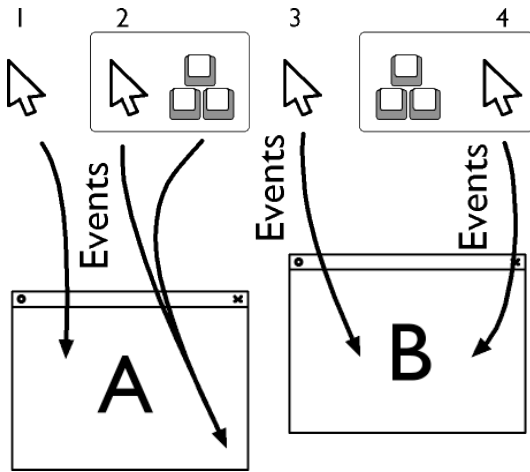


Figure 1. MPX delivers the event to the applications that has the device's focus.

MPX is not an implementation of a new windowing system. MPX changes the input subsystem and event delivery of the X.org X server, the de-facto standard for X server implementations. As a result, it is compatible to any X application and can be used as a drop-in replacement for the standard X server. X is the standard windowing system for virtually every graphical desktop under Unix, BSD and Linux. A replication of MPX's functionality in the windowing systems of Microsoft Windows and Apple Mac OS X will require internal device-based state tracking for all input events and a revision and modification of the windowing system APIs.

3 Ad-hoc Collaboration on the Desktop

A GWWS needs to support the hotplugging of input devices to support ad-hoc collaboration. By adding and removing input devices users can transition fluidly between a single-user desktop and a groupware environment. Multiple input devices provide the ability for multiple users to interact simultaneously with any number of applications. With such a GWWS, users can set up an impromptu collaboration environment on any workstation.

MPX can support up to 128 (7 bit) independent input devices. Devices can be pointing devices (mice, trackballs, pens, touch screens, etc.) or keyboard devices. For the scope of this paper, we will refer to any type of pointing device as a *pointer*. The visible representation on

the screen of such a pointer will be referred to as a *cursor*.

In traditional windowing systems, the keyboard input focus is usually set when the user clicks into a window. All keyboard events are then delivered to the application that obtained the input focus. With the availability of multiple cursors and multiple keyboard foci, MPX can *pair* each keyboard's focus with one distinct cursor and thus provide one focus per keyboard. A pair of input devices behaves exactly like the traditional mouse-keyboard combination, with the cursor controlling the paired keyboard's focus. MPX delivers the events based on a device's focus. In Figure 1, pointer and keyboard 2 have their focus set to application A. At the same time, pointer and keyboard 4 have their focus set to B. If one of these devices emits events, these events are only sent to the respective application. By keeping the event streams separate, MPX allows simultaneous interaction with multiple applications, even with legacy applications. This gives users the ability to work independently. The legacy applications do not know that there are multiple keyboard foci or multiple cursors.

All major operating systems support input device hotplugging of pointers and keyboards, but windowing systems do not utilise this functionality. Additional pointers are always merged into the single system cursor, and additional keyboards will share a single input focus. As a result, adding additional input devices does not provide any collaboration features in traditional windowing systems.

MPX creates a new additional cursors and keyboard foci for devices hotplugged at runtime. When a new keyboard is plugged in, MPX automatically pairs it with the first available unpaired pointer (see Figure 2a). This automatic pairing for hot-plugged devices directly benefits users. A user only needs to plug in a mouse and a keyboard, a new cursor and a new keyboard focus become available and the user can start working immediately, without the need for further configuration. If all available pointers are already paired, the first pointer is chosen (see Figure 2b). If no physical pointer is connected to the host computer, the keyboard is paired with a virtual pointer to ensure data consistency (see Figure 2c). MPX also exposes an API to change the device pairing at any time. Each keyboard device can be paired with one pointer only but a pointer may be associated with multiple keyboards (1:N

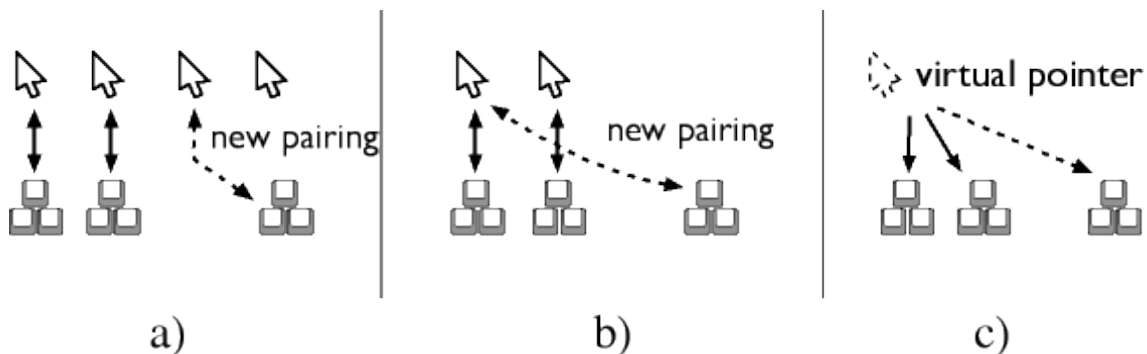


Figure 2. Automatic pairing of hot-plugged devices. a) new keyboard with available unpaired pointer, b) new keyboard with no unpaired pointer available, c) keyboard with no physical pointer connected.

pairing). Whenever a device pairing is changed through the API or by physically adding or removing devices, applications are notified about the new pairing. The list of pointer-keyboard pairings can be queried at any time, allowing applications to adjust their interfaces or interaction methods based on the current device pairing.

Modifying the keyboard-pointer pairing has wide-reaching consequences on user interaction. For example, if the pairing is changed at an inappropriate time, a user typing a password may suddenly end up typing it into a text field visible to other users. For this reason, only one application at a time can be authorized to change device pairing. The window manager is arguably the application with the biggest influence on the user interface. It is responsible for window placement, focus changes and the minimizing and maximizing of the windows. We believe that the window manager is the ideal application to administer device pairing. Our experimental window manager MPWM provides a default device pairing mechanism where users can click on the icon of a keyboard to pair it with their pointer device. However, we acknowledge that further research and experimentation is needed to develop a less interruptive pairing interface. A simple method to explicitly pair devices is to unplug both devices, and immediately plug both devices back into the computer.

Unplugging a pointer that is paired to a keyboard will release the keyboard's pairing. As a result, keyboard events will not be sent to any application and the keyboard needs to be manually paired with a different pointer. MPX does not automatically re-pair a device for security reasons. Automatic re-pairing may also result in two keyboards being paired with a single pointer, which leads to interference when both users type simultaneously.

The support for hotplugged input devices directly in the windowing system leads to a new perception in the user interface. Instead of a single-user environment, it is now a true multi-user environment with the number of input devices varying over time. This number can even be zero, when all physical devices are unplugged. Future applications must be developed with this in mind. However, legacy applications assume the existence of one pointer and one keyboard. MPX internally keeps a virtual

pointer and a virtual keyboard. These virtual devices are only utilised when legacy application request information that requires data from a pointer and/or keyboard and there is no physical device connected to the host computer. The virtual devices never send events to the applications and are ignored whenever MPX needs to traverse the internal device lists for processing.

Collaboration is not limited to desktop computers. For example, tabletops and shared whiteboards are commonly used for collaboration on a single display (Ishii 1992, Kruger 2002, Wu 2003), yet they require custom setups and depend on prototype toolkits (Esenther 2002, Shen 2004). With a GWS like MPX, these displays can provide the same environment as a desktop computer. This allows executing both collaborative applications as well as traditional desktop single-user applications, thus blurring the difference between a desktop computer and a shared collaborative surface. With hotplugging support in the GWS, users can just naturally walk up to a table or whiteboard and start interacting.

Ad-hoc collaboration on a desktop can suffer from a lack of screen real estate. MPX supports both multi-monitor graphic cards and also multiple graphics cards in one host computer, allowing for setups with several display devices. Grudin found in a survey that multiple monitors were often used as secondary displays and not necessarily as extension of the available screen size (Grudin 2001). He noted that windows were rarely maximised across two monitors. Such behaviour is beneficial for ad-hoc collaboration. Two users could utilise one monitor each, and collaborate with minimal interference, while at the same time being able to access the peer's screen estate.

4 Resolving API Ambiguity

All traditional single-user applications are designed for an infrastructure that provides a single system cursor and a single keyboard focus. Applications communicate with the X server by sending requests over a reliable socket. Most of the communication is part of the core X protocol, but several protocol extensions are available. About one quarter of the requests in the core X protocol becomes ambiguous if more than one cursor and/or more than one keyboard focus are available. A typical example for such an ambiguity is an application querying the position of the cursor. In MPX, there may be multiple cursors on the screen.

Our novel *ClientPointer* principle resolves these ambiguities. Upon application start-up, the windowing system assigns one distinct pointer to the application. Whenever an application issues an ambiguous request, this ClientPointer is chosen as the default device. If a request requires keyboard data, the keyboard that is paired with the ClientPointer is chosen. Figure 3 demonstrates the ClientPointer principle. Application A has pointer 1 set as ClientPointer, application B pointer 4. When A issues an ambiguous request (say get cursor position), MPX uses pointer 1 to provide the necessary data (dashed line). When B issues an ambiguous request (say warp cursor), MPX selects pointer 4 (dotted line). If B's request requires keyboard data, MPX selects the

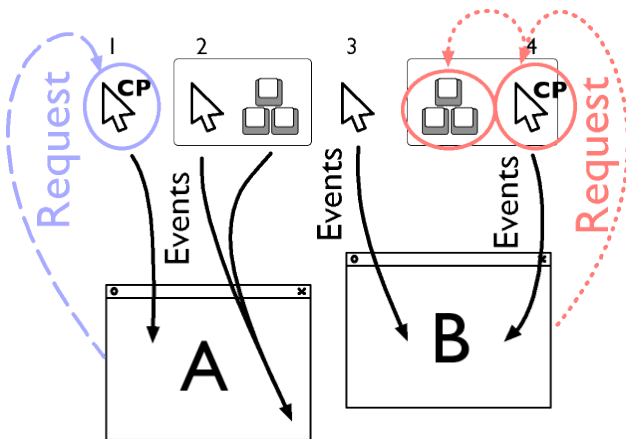


Figure 3. Illustration of the ClientPointer principle.

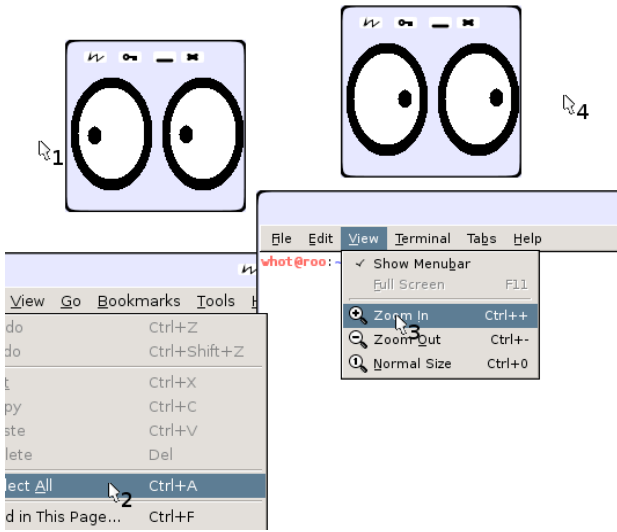


Figure 4. Screenshot of MPX with four legacy applications utilizing the ClientPointer principle.

keyboard paired with pointer 4 (dotted line). No keyboard is paired with pointer 1 and if A issues a request that requires keyboard data, MPX provides data from a virtual keyboard that is in a neutral state. Note that although pointers 1 and 4 are set as ClientPointers, pointers 2 and 3 can nevertheless interact with both applications (solid lines).

The xeyes application shown in Figure 4 queries the cursor coordinates and then adjusts the eyes to look at the cursor. In Figure 4, two xeyes have different ClientPointer settings and thus look at two different cursors. The ClientPointer can be set at any time using Xlib functions. For the same reasons as given above, we believe that the window manager should be responsible for adjusting the ClientPointer. Our window manager changes the ClientPointer upon click into a window. For exotic management of ClientPointer rules, explicit applications may be developed to handle these cases.

Previous research prototypes used the notion of a system cursor and virtual cursors (Hourcade 1999, Hutterer 2006, Tse 2004). The ClientPointer principle is different. In MPX, all cursors are true system cursors. Each application can have a different ClientPointer, and any input device can interact with the application regardless of the ClientPointer setting (solid lines in Figure 2). The ClientPointer only gets preference over the other pointer devices when an application issues an ambiguous request. If an application never issues such a request, all pointer devices are equal.

The ClientPointer principle is designed to resolve ambiguities for legacy applications. Multi-user applications do not need to use ambiguous single-user APIs and thus do not depend on the ClientPointer setting.

The ClientPointer principle ensures valid data is provided to an application, but it depends on the application how to process the data. Some features such as typing into two text fields simultaneously within one application need active support by the application or toolkit. These

applications need to be adapted to comply with the new multi-device paradigm. However, Tse et al. (Tse 2004) noticed that even when users work on the same task they consciously and unconsciously avoid interference. With this in mind and the existing proper floor control in MPX (Hutterer 2007b), we believe that many legacy applications will not need modifications.

5 Grab ownership

The standard X APIs allow an application to “grab” the pointer and/or the keyboard. During such a grab, only the grabbing application will get events from the device, regardless of the device’s focus or the pointer’s location. Grabs are heavily used for popup menus. For example, while a popup menu is visible, a click on the menu itself results in the appropriate action, whereas a click outside of the menu will cause the menu to disappear. A “pointer grab” ensures that the event is delivered to the popup menu, regardless of the click location. Only one application can have a grab on a given device at a time.

Grabs are designed on the assumption that there is only one set of input devices. In MPX, if one pointer were to send an event after another pointer caused a popup menu to appear, the popup menu may disappear again immediately. This interrupts the user’s interaction. To avoid this, we have introduced the notion of “grab ownership” (see Figure 5). While a grab is active (dashed line), no other device can send events to the grabbing application (dotted lines). At the same time, other devices can interact with all other applications (solid lines).

X allows active grabs and passive grabs. Active grabs are requested by the application and last until the application requests to “ungrab” the pointer/keyboard. A legacy application does not explicitly specify the device to grab, and in MPX these active grabs default to the ClientPointer, or the keyboard paired with the ClientPointer respectively.

Passive grabs on the other hand are requested only once and stored in the X server. They are activated when a particular button or key (as defined by the application) is pressed, and the grab is deactivated again when this button or key is released. Passive grabs are heavily used for drop-down and popup menus, as well as window

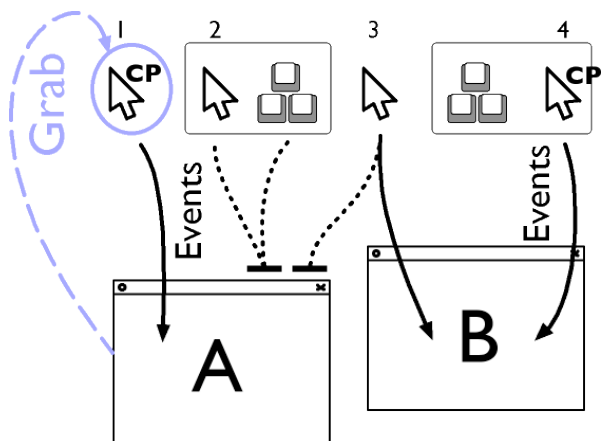


Figure 5. Grab ownership for legacy applications.

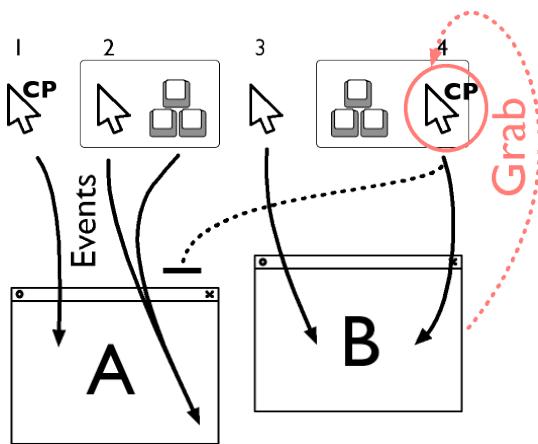


Figure 6. Device grabs as applicable for multi-user applications.

manager interaction. Since passive grabs are activated in response to an interaction with a GUI element, we introduced grab ownership with flexible devices for passive grabs. Instead of selecting the ClientPointer, MPX switches the grab device upon activation to reflect the device causing the passive grab to activate. If the application requests an active grab while a passive grab is active, the active grab is issued on the switched device. This ensures interface consistency.

Grab ownership only applies to legacy applications. Multi-user applications know about the existence of multiple devices and can explicitly specify which device to grab. In this case, the notion of a grab is different (see Figure 6). For legacy applications, activating a grab means that only one device can interact with the grabbing application. For multi-user applications, an active grab on a device (dashed line) ensures that any event from this device being sent only to the grabbing application, and no other application can receive events from this device for the duration of the grab. At the same time, the application (and any other application) can still receive events from a non-grabbed device (solid lines).

Grab ownership allows multiple popup windows simultaneously, albeit only across multiple applications. As mentioned before, it depends on the application to process input events. Multiple simultaneous popup windows within the same application require application support and are thus not possible for legacy applications.

6 The MalaMinya and MM2 drawing applications

We implemented two multi-user drawing editors to demonstrate MPX's support for single-display groupware applications. The first application, "MalaMinya", is a simple bitmap drawing tool that allows several users to interact simultaneously on the canvas (see Figure 7). The drawing canvas is surrounded by a number of colour buttons. One home area is provided for each user, with buttons to select the pen tool, the eraser tool and to clean the whole canvas. The home areas are locked to their respective user using MPX's floor control mechanism. A user's cursor is appended with a small icon, and this icon is also displayed next to the user's home area. The home



Figure 7. Screenshot of the MalaMinya drawing tool.

areas are arranged around the canvas to accommodate for tabletop users and their physical position around the table.

A selection of a tool or a colour only affects this user's state, allowing for different tools and colours for each user. All interaction with MalaMinya can happen simultaneously. For example, while one user may be drawing on the canvas, another user may be erasing a part of the canvas. Other users may simultaneously change their tools and/or colours and start drawing.

The second drawing tool, "MM2", provides vector-based shapes (lines, rectangles, ellipses) to be put on the canvas (see Figure 8). Shapes can be manipulated once on the canvas. Each user has a free-floating home area for tool selection. It displays buttons to select a shape, change colour and adjust line thickness. Each home area is only accessible by the respective user and can be moved around freely on the canvas. MM2 accommodates for hot-plugged devices and adds and removes home areas as new pointer devices are plugged in.

MalaMinya and MM2 use two different design choices. MalaMinya places all user interface elements around the canvas. MM2 on the other hand provides all tools in the moveable home areas. We expect different interaction for each of the tools when employed on a tabletop display. MalaMinya requires users to reach across the table to activate a control, potentially interfering with other user's personal space. MM2 on the other hand allows private works. Each user can drag the home area into their personal space and interact with the drawing area, without interfering with others. MalaMinya and MM2 have been tested with up to eight and four pointers respectively.

Both MalaMinya and MM2 are prototype applications and demonstrate how to write groupware applications for MPX. The novelty about our drawing tools is that the integration of MPX into the windowing system allows them to be used simultaneously with any other application on the screen. For example, three users could sketch in MalaMinya, while four others assemble a

drawing in MM2 and two more users use a legacy word processor and a web browser.

7 Conclusion

In this paper we presented how MPX enables fluid transition between a single-user and a multi-user environment by supporting input device hotplugging. The low-level integration of collaboration features allows any X application to be utilised in a multi-user context. MPX can be used as a drop-in replacement for a traditional X Server. Additional cursors and keyboard foci are added and removed as new devices are plugged into the host computer. Automatic device pairing allows users to collaborate instantly after connecting new devices.

The ClientPointer is a novel principle to resolve ambiguity in single-user APIs. Each application is assigned a ClientPointer device, and MPX chooses the ClientPointer to provide data for ambiguous requests. Any device can interact with an application regardless of the ClientPointer setting, and the ClientPointer can be changed at run-time.

The concept of grab ownership enables multiple applications to gain exclusive access to a pointer and/or keyboard device at a time to provide common interfaces such as popup menu. Allowing passive grabs with flexible devices allows for multiple popup or drop-down menus simultaneously.

Both the ClientPointer and grab ownership are designed for legacy applications and do not affect novel multi-user applications. These applications can query the list of input devices at any time and thus adjust their interface to provide true multi-user interaction within the same application.

MPX broadens the single-pointer single-keyboard paradigm that is prevalent current desktop environments. To extend the windowing systems of Microsoft Windows and Apple Mac OS X to become GWWS, they need to replicate the ClientPointer principle to maintain compatibility with legacy applications. Additionally, the windowing systems need to adjust the event delivery to represent each cursor and keyboard focus. Finally, they need to provide new APIs for multi-user applications.

8 Future Work

We are currently working on integrating additional functionality into MPX such as remote controlled input devices for distributive collaborative groupware. A formal evaluation of MPX is pending. We also plan to improve MalaMinya and MM2 and extend the suite of demonstration applications. Finally, we are working on merging MPX into the main X.org source code.

9 Acknowledgements

The authors would like to acknowledge NICTA for funding this project. We would furthermore like to thank the members of the Wearable Computer Lab, especially Benjamin Close, Mark Rebane and Rebecca Witt, and the X developer community, especially Daniel Stone and Keith Packard, for their comments.

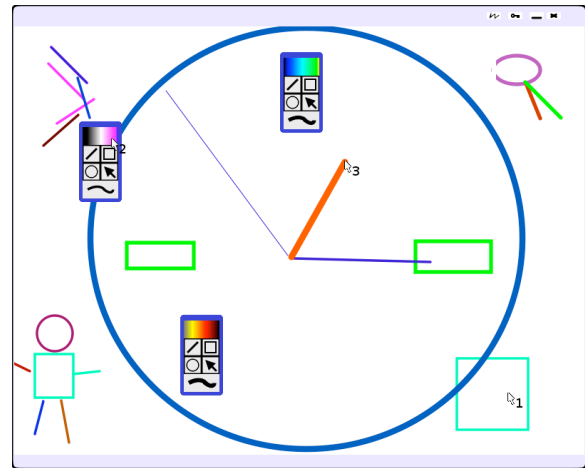


Figure 8. Screenshot of the MM2 drawing tool.

10 References

- Baudisch, P., Sinclair, M., and Wilson, A. (2006): Soap: a pointing device that works in mid-air. In *UIST '06: Proceedings of the 19th annual ACM symposium on User interface software and technology*, pp 43-46, Montreux, Switzerland, 2006.
- Bier, E. A. and Freeman, S. (1991): MMM: a user interface architecture for shared editors on a single screen. In *UIST '91: Proceedings of the 4th annual ACM symposium on User interface software and technology*, pp 79-86, Hilton Head, South Carolina, United States, 1991.
- Dietz, P. and Leigh, D. (2001): DiamondTouch: a multi-user touch technology. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, pp 219-226, Orlando, Florida, 2001.
- Esenher, A., Forlines, C., Ryall, K., and Shipman, S. (2002): DiamondTouch SDK: Support for Multi-User, Multi-Touch Applications. Mitsubishi Electronics Research Laboratory, Report No. TF2002-48.
- Greenberg, S. (2007): Toolkits and interface creativity. *Multimedia Tools Appl.*, Vol. 32, No. 2, pp 139-159, 2007.
- Grudin, J. (1994): Groupware and social dynamics: eight challenges for developers. *Commun. ACM*, Vol. 37, No. 1, pp 92-105, 1994.
- Grudin, J. (2001): Partitioning digital worlds: focal and peripheral awareness in multiple monitor use. In *CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems*, pp 458-465, Seattle, Washington, United States, 2001.
- Han, J. Y. (2005): Low-cost multi-touch sensing through frustrated total internal reflection. In *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology*, pp 115-118, Seattle, WA, USA, 2005.

- 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 AUIC '06: Proceedings of the seventh conference on Australasian user interfaces, pp 107-114, Hobart, Australia, 2006.
- Hutterer, P. and Thomas, B. H. (2007a): Bridging the Gap between Desktop Computers and Tabletop Displays. Conference Supplement of Tabletop 2007: IEEE Int'l Workshop on Horizontal Interactive Human-Computer Systems (DVD Proceedings), 2007.
- Hutterer, P. and Thomas, B. H. (2007b): Groupware Support in the Windowing System. In AUIC '07: Proceedings of the eighth conference on Australasian user interfaces, pp 39-46, Ballarat, Australia, 2007.
- Inkpen, K., McGrenere, J., Booth, K. S., and Klawe, M. (1997): The effect of turn-taking protocols on children's learning in mouse-driven collaborative environments. In GI '97: Proceedings of the conference on graphics interface, pp 138-145, Kelowna, British Columbia, Canada, 1997.
- Ishii, H. and Kobayashi, M. (1992): ClearBoard: a seamless medium for shared drawing and conversation with eye contact. In CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems, pp 525-532, Monterey, California, United States, 1992.
- 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 UIST '03: Proceedings of the 16th annual ACM symposium on User interface software and technology, pp 159-168, Vancouver, Canada, 2003.
- Kruger, R., Carpendale, S., and Greenberg, S. (2002): Collaborating over Physical and Electronic Tables. In Extended Abstract of CSCW '02, pp 139-140, November, 2002.
- Pawar, U. S., Pal, J., Gupta, R., and Toyama, K. (2007): Multiple mice for retention tasks in disadvantaged schools. In CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems, pp 1581-1590, San Jose, California, USA, 2007.
- Shen, C., Everitt, K., and Ryall, K. (2003): UbiTable: Impromptu Face-to-Face Collaboration on Horizontal Interactive Surfaces. MERL, Report No. TR-2003-49, 2003.
- Shen, C., Vernier, D., Forlines, C., and Ringel, M. (2004): DiamondSpin: an extensible toolkit for around-the-table interaction. In CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems, pp 167-174, Vienna, Austria, 2004.
- 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.
- Stanton, D. and Neale, H. R. (2003): The effects of multiple mice on children's talk and interaction. *J. Comp. Assisted Learning*, Vol. 19, No. 2, pp 229-238, 2003.
- 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 AUIC '04: Proceedings of the fifth conference on Australasian user interfaces, pp 101-110, Dunedin, New Zealand, 2004.
- Tse, E., Histon, J., Scott, S. D., and Greenberg, S. (2004): Avoiding interference: how people use spatial separation and partitioning in SDG workspaces. In CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work, Chicago, Illinois, USA, 2004.
- Wilson, A. D. (2004): TouchLight: an imaging touch screen and display for gesture-based interaction. In ICMI '04: Proceedings of the 6th international conference on Multimodal interfaces, pp 69-76, State College, PA, USA, 2004.
- Wu, M. and Balakrishnan, R. (2003): Multi-finger and whole hand gestural interaction techniques for multi-user tabletop displays. In UIST '03: Proceedings of the 16th annual ACM symposium on User interface software and technology, pp 193-202, Vancouver, Canada, 2003.