Architecture and Implementation of a Java Package for Multiple Input Devices (MID)

Juan Pablo Hourcade, Benjamin B. Bederson
Human-Computer Interaction Lab, Institute for Advanced Computer Studies
Computer Science Department
University of Maryland, College Park, MD 20742 USA
+1 301 405-2764
{jpablo, bederson}@cs.umd.edu

ABSTRACT
A major difficulty in writing Single Display Groupware (co-present collaborative) applications is getting input from multiple devices. We introduce MID, a Java package that addresses this problem and offers an architecture to access advanced events through Java. In this paper, we describe the features, architecture and limitations of MID. We also briefly describe an application that uses MID to get input from multiple mice: KidPad.

Keywords
Single Display Groupware (SDG), Computer-Supported Cooperative Work (CSCW), Multiple Input Devices (MID), Multi-Modal Input, Java, DirectInput, Windows 98, Universal Serial Bus (USB), KidPad, Jazz, Pad++.

INTRODUCTION

Communication, collaboration, and coordination are brought to many people's desktops thanks to groupware applications such as Lotus Notes and Microsoft Exchange, some of the leading commercial products in the field of Computer-Supported Cooperative Work (CSCW). They help people collaborate when they are not in the same place at the same time. We believe that computers should also help people collaborate when they are in the same place at the same time.

Single Display Groupware (SDG) is a model for supporting co-present collaborative work. An SDG application makes it possible for co-present users to collaborate using a single computer and display through the use of multiple input devices [16].

SDG applications must deal with problems particular to their domain, and rules of interaction need to be considered. There are a range of issues that come up for builders of SDG applications. Providing interaction metaphors that make sense for SDG is a challenge. What should an application do if users issue incompatible commands? How should an SDG application deal with global states and focus issues? Should each user have his/her own set of tools and menus? The use of screen space is also critical. These questions have been addressed elsewhere [4, 13, 16]. In this paper, we focus on the technical issue of getting input from multiple devices on one computer with a consistent mechanism.

Figure 1: Two children using two mice with one computer. They are running KidPad, an application we wrote that uses MID.

We introduce Multiple Input Devices (MID), a Java package that supports input from multiple devices on a single computer. It provides developers using Java the ability to write SDG applications with a powerful yet straightforward and cross-platform way for getting input from multiple devices. It is a general purpose architecture for various kinds of input devices.

We chose Java because of platform independence. We want our SDG applications to run on multiple platforms.

MID consists of a cross-platform Java layer, and a native platform-specific layer. The cross-platform layer has to be extended for each new device type. A native class must be implemented for each device on each platform.

The Java layer extends the standard Java event classes. Applications using MID use just this layer, and do not have to be changed for use on different platforms.

MID currently supports just Universal Serial Bus (USB) mice on Windows 98. We picked this hardware/platform combination to start because of its wide availability and install base, and because of the generality of USB devices. While Windows NT does not yet support USB devices, Microsoft has informed us that the next version of NT will, and we should be able to get input from multiple mice there as with Windows 98. The only issue with the next version of NT is that a registry key will have to be set so that NT will not merge the input from multiple mice before it gets to MID.

We chose mice as the first device to support because they are not very expensive, and because a good percentage of Personal Computer users are familiar with how to use them. We chose the USB standard because up to 256 devices can be connected through a single USB port (via hubs), and USB is a growing standard which we expect will dominate the protocols for input devices in the near future.

While MID is written largely in Java and is designed to be as cross-platform as possible, there is a fundamental trade-off to be made in the degree of portability vs. the feature set MID offers. Since Java already supports a truly cross-platform event model, we decided that our goal for MID is to enable applications to access all the input device capability possible while working in as cross-platform a manner as possible.

This trade-off has worked out to mean that while the applications that use MID do not have to be changed to run on different platforms, the underlying implementation of MID must be updated for different devices and different platforms. Furthermore, the very nature of accessing specialized input devices means that if those input devices are not physically available, then it is the application's responsibility to deal with that, and accommodate the user with whatever input devices are available. The good news, though, is that applications that use MID will work even if the MID native code is unavailable. In this case, MID reverts to offering just standard Java events.

RELATED WORK

Other people have also looked at the problem of getting input from multiple devices. Stewart solved the problem on X Windows using XInputExtension [15]. His solution supported serial mice and tablets, and was designed for Tcl/Tk [Ousterhout]. MMM (Multi-Device, Multi-User, Multi-Editor) was an early SDG environment that also supported input from up to three mice [4].

The Pebbles project has investigated the use of Personal Digital Assistants (PDAs) as input devices for SDG applications [13]. While PDAs as input devices are a fine choice for people that already have them or for some special situations, they are currently a prohibitive expense for most people that just want an input device to control a mouse cursor, being approximately 10 times the cost of a standard mouse. MID gets input from USB mice, which are considerably less expensive than PDAs. Mice may also have an advantage over PDAs when users, such as young children, don't have a lot of precision in their control of input devices.

Inkpen studied the effects of turn-taking protocols on children's learning in mouse-driven collaborative environments [10]. In her study, two mice were connected to a computer, but they were not used simultaneously. Bricker used Windows-based gaming input devices to investigate how multiple users collaborated on a single computer when each controlled a different aspect of the input [5].

Rekimoto's pick-and-drop [14] allows for drag-and-drop between computers or PDAs through the use of special "pens". While his architecture supports a computer receiving input from multiple devices, the focus of his research was not on SDG. Pick-and-drop requires special hardware, computers that are connected by a network, and provides only two types of interaction: picking and dropping.

Some video game systems can get input from multiple devices and would be able to support limited SDG applications. These systems for the most part support only joysticks and are not particularly easy to program.

Finally, Buxton and others have investigated how one person can use both hands simultaneously to interact with a computer. This area also has all been implemented with custom code to receive input from one extra device, typically a tablet using a serial port [6, 11, 12].

MID

MID extends the standard Java event mechanism. In order for applications to use MID, they must replace all uses of the standard Java events with the extended MID events.

With the current support of Windows 98 USB mice, an application using MID can access multiple mice provided the MID dynamic link library (MID DLL) is present in the system. We will refer to a system running Windows 98, with the MID DLL and USB mice as a MID-ready system.

Applications that use MID will always work. If an application that uses MID runs on a non-MID-ready system (i.e., the native DLL file is not installed and/or USB mice are not present and/or the operating system is not Windows 98), MID reverts to using Java mouse events. Therefore, an application that uses MID will have the advantage of getting input from multiple mice when running on MID-ready systems, and will still work with one mouse when running on non-MID-ready systems.

USE OF MID EVENTS VERSUS JAVA EVENTS

To a programmer, using MID is very similar to using Java mouse events. MID extends the three Java classes associated with events that applications typically use: event listeners, event sources, and the events themselves.

Use of Java Mouse Events

A class that receives Java mouse events has to implement two interfaces in order to get all possible mouse events: `MouseListener` and `MouseMotionListener`. It also has to register with a component (usually the visible window where the events will occur) to get those events. It does so by calling the `addMouseListener` and `addMouseMotionListener` methods on the component. Finally, the class has to implement the methods specified in the interfaces. These methods will be called when mouse events occur, and a `MouseEvent` object describing the event will be passed to them. The top of Figure 2 shows sample code of a class that uses Java mouse events.

Use of MID Mouse Events

The bottom of Figure 2 shows sample code of a class that uses MID mouse events. The most noticeable difference between this code and the code that uses Java mouse events is one extra line of code that gets a MID source (`MIDMouseEventSource`) object. A component is passed to the MID source object because the native code needs a handle to a visible window in order to work.

Instead of registering with a component, the class has to register with the MID source object to listen to MID events. This is because the MID source object generates the MID events. MID offers the flexibility of registering to listen to all devices or to a particular device. This supports applications written in two styles. The first style dispatches events from all devices to each listener. It is up to the listener to determine which device generated the event by calling `MIDEvent.getMouseID()`, which returns the ID of the mouse that generated the event.

An alternative application style is to register a listener to receive events from a specific device. Then, an application would write one listener for each device, which would support decoupling of the devices.

In addition, there is a call to `MIDMouseEventSource.getNumberOfMice()` that returns the number of mice currently available. This turns out to be necessary for most applications so they can build internal data structures and mechanisms to support the available input devices. The only other difference in the code is that the names of the interfaces and the registration methods, and the name of the event have "MID" prepended.

On MID-ready systems, the MID source object supports extra features. It allows developers to set the location of mice, motion constraints, and the time span used for counting mouse clicks. Setting the location of mice is particularly useful for an application that has to define its own cursors and needs to give them an appropriate initial location on the screen. Constraining the motion of all mice and/or the motion of particular mice can be used to keep the cursor within the visible area of the window. Another application would be to divide the screen into regions where only one mouse would operate in each region.

Another important extra feature of MID is that it enables applications to access all of the input buttons and movement axes on the particular mouse in use. Thus, a mouse with a wheel generates a third button event and motion in the z axis.

MID ARCHITECTURE

In this section we describe the design of MID. Figure 3 shows MID’s class structure.

MID with and without native code

MID is works on both MID-ready and non-MID-ready systems. We provide developers with a MID source class. It is a singleton [9] and implements a `getInstance()` method that returns one of its subclasses: MID multi source (`MIDMultiMouseEventSource`), or MID single source (`MIDSingleMouseEventSource`). A MID multi source object is returned when running on MID-ready systems. It uses native code to get input from multiple mice. A MID single source object is returned when running on non-MID-ready systems. It listens to Java mouse events and then turns them into MID mouse events.

MID on MID-ready systems

The MID multi source class is the heart of MID. On MID-ready systems it takes care of registering listeners, checking for mouse events, creating MID mouse events, and broadcasting them to the listeners. It also takes care of providing all the extra functionality developers get from MID source objects on MID-ready systems. And, in order to keep track of the listeners it registers, MID has its own multicaster.

The MID native code generates events by using the Microsoft DirectInput API to receive events from USB mice. However, it currently assumes there is only one keyboard in order to support the mouse event's access to the keyboard control, shift, and alt modifier keys. This access to the keyboard through the mouse events demonstrates the low level at which the assumption of a single mouse and keyboard have been made. When MID supports multiple keyboards, we will have to deal with this differently.

```
// Java code using standard Java events to access mouse and mouse motion events
//
class MyClass implements MouseListener, MouseMotionListener {
        // Constructor adds mouse and mouse motion listeners
    public MyClass(Component myComponent) {
        . . .
        myComponent.addMouseListener(this);
        myComponent.addMouseMotionListener(this);
        . . .
    }

        // Mouse listener events
    public void mousePressed(MouseEvent e) {
        . . .
    }
    public void mouseReleased(MouseEvent e) {
        . . .
    }
    . . .

        // Mouse motion listener events
    public void mouseMoved(MouseEvent e) {
        . . .
    }
    public void mouseDragged(MouseEvent e) {
        . . .
    }
}
```

```
// Java code using MID events to access mouse and mouse motion events
// from multiple USB mice.
//
class MyClass implements MIDMouseListener, MIDMouseMotionListener {
    int numberOfMice;         // Number of USB mice connected to system
    public MyClass(Component myComponent) {
        . . .
        MIDMouseEventSource midSource = MIDMouseEventSource.getInstance(myComponent);
        numberOfMice = midSource.getNumberOfMice();
        midSource.addMIDMouseListener(this);
        midSource.addMIDMouseMotionListener(this);
        . . .
    }

        // Mouse listener events
    public void mousePressed(MIDMouseEvent e) {
        int device = e.getMouseID();       // Use mouse ID in application-specific manner
        . . .
    }
    public void mouseReleased(MIDMouseEvent e) {
       int device = e.getMouseID();       // Use mouse ID in application-specific manner
        . . .
    }
    . . .

        // Mouse motion listener events
    public void mouseMoved(MIDMouseEvent e) {
       int device = e.getMouseID();       // Use mouse ID in application-specific manner
        . . .
    }
    public void mouseDragged(MIDMouseEvent e) {
       int device = e.getMouseID();       // Use mouse ID in application-specific manner
        . . .
    }
}
```

Figure 2: The top code fragment shows Java code that gets standard Java mouse and mouse motion events. The bottom code fragment shows Java code using MID events to access multiple mice.

The Java event queue uses event coalescing to eliminate redundant events when new events are posted. For example, new mouse motion events replace existing unhandled motion events on the queue. We had to write a custom version of event coalescing for MID mouse events to make sure that events from different mice were not coalesced.

MID on non-MID-ready systems

A MID single source object runs on non-MID-ready systems. It wraps the standard Java events by registering itself as a mouse listener, and then passing the events on the application.

The fact that MID single source uses pure Java code guarantees that it will run on all platforms supported by Java.

To review, Figure 4 summarizes MID's features.

EXTENDING MID

MID can be extended in two ways: adding support for multiple mice on other platforms, and adding support for other devices.

Adding support for multiple mice on other platforms consists of creating a native library that gets input from multiple mice and interfaces correctly with the MID multi source class. The library must implement all the native methods declared in the MID multi-source class, and call the MID multi-source when a mouse event occurs.

Adding support for other devices is more complex. Each new device type needs its own event class, and at least one listener interface with its corresponding adapter. The MID multicaster, and MID source classes would have to be modified as well.

LIMITATIONS

The most limiting aspect of MID is that it currently gets input from multiple mice only under Windows 98, when the MID DLL is in the system, and when the mice are USB mice.

The mice have to be USB mice because of a limitation of Windows. Windows merges the input from the non-USB mouse with the input from USB mice into one channel.

Another limitation of MID is that, when MID multi source is the source of MID mouse events, MID takes over the system cursor and users can't send mouse input to other

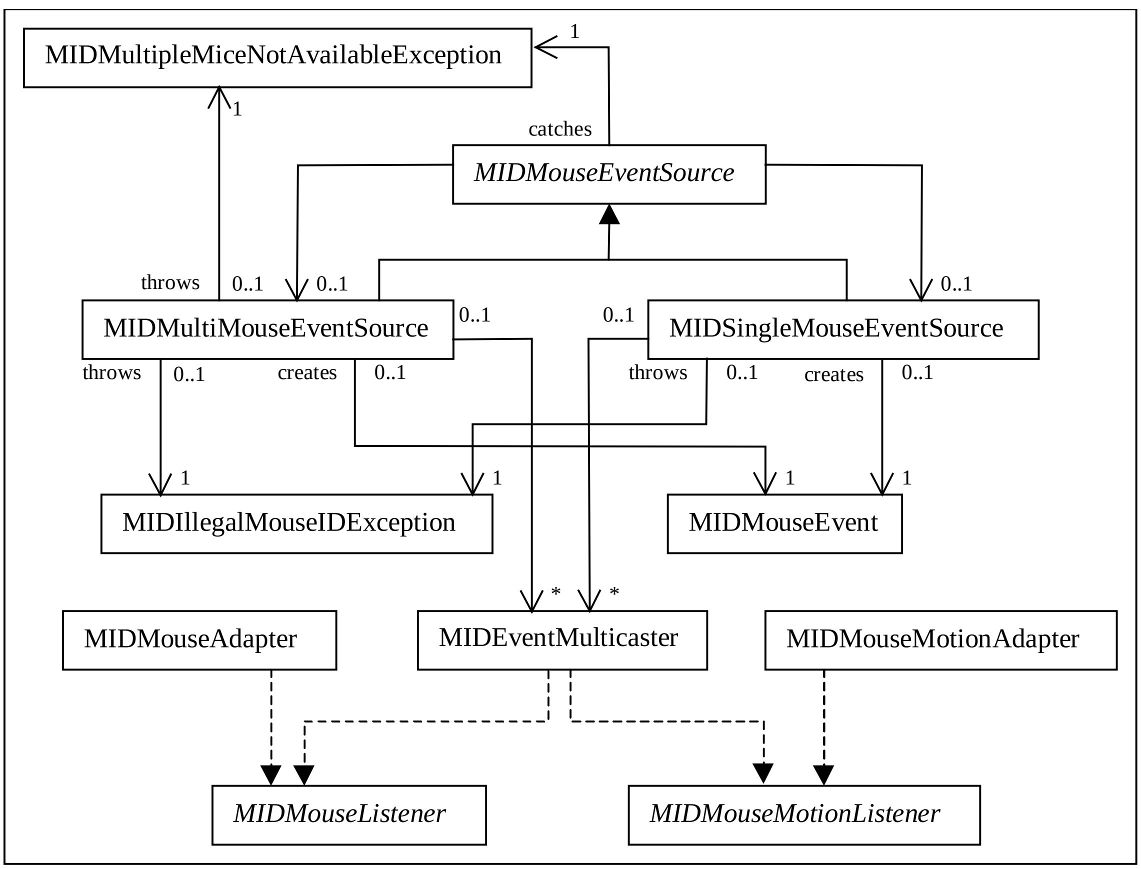

Figure 3: Class structure of MID in UML. Solid lines with solid arrows represent inheritance. Dashed lines with solid arrows represent implementation. Solid lines with open arrows represent associations. The numbers and text next to an association line represent cardinality and role of a class in the association.

applications unless they switch to other applications through the keyboard. This was done on purpose because the alternative is to have both mice control the single system cursor.

PERFORMANCE

By generating MID mouse events when notified by DirectInput, MID suffers from minimal overhead when MID multi source is the source of MID mouse events.

When MID mouse events are generated by MID single source, the only overhead is that the Java mouse events are passed through MID's event listener on the way to the application listener.

SAMPLE APPLICATION: KIDPAD

We incorporated MID into an application with a user interface tailored to SDG. KidPad [7, 8] is an authoring tool for children. It includes drawing, typing and hyperlinking tools. It also has zooming capabilities. It supports the collaborative creation and telling of non-linear stories. Several children use KidPad on a single computer simultaneously where each child has their own mouse to interact with the software.

There have been several previous versions KidPad that have been written in Tcl, C++, and Perl – all using the Pad++ library to support zooming [2, 8, 17]. We recently rewrote KidPad in Java using Jazz, a Java package that supports 2D object-oriented graphics with zooming [3].

The Perl version of KidPad [16] got input from multiple mice on Linux computers running X. This version was very tightly tied to UNIX and X, making porting it to other platforms difficult. This was a significant problem since our target users (children) have very little access to Linux.

Nevertheless, it was a successful SDG application. A good part of the reason for that success was the design of the user interface. It used local tools [1, 16] instead of using menus or tool palettes. A local tool is a graphical object that sits on the data surface along with the user-created data. Each local tool can be picked up and used with each input device. This approach is naturally suited to SDG because there is no global state (such as pen color or font). Instead, each tool has its own state. In addition, the local tools act as cursors which is crucial since no current operating systems support multiple cursors.

The aim of rewriting KidPad in Java was to make it available on multiple platforms, and to take advantage of Jazz, which has replaced Pad++, the earlier zoomable graphics package. Local tools were also used in the Java version of KidPad to facilitate making it an SDG application. In about a day's work we were able to modify KidPad's standard Java event code so that it would take input from multiple mice. And because of MID's architecture, KidPad runs on both MID-ready and non-MID-ready systems (i.e., 100% Java with one mouse, or a combination of Java and native code with multiple mice).

- 100% Java API with native code per device per platform allows cross-platform applications with no awareness of application-specific details.
- Native code defines event input mechanisms. Currently supports USB mice on Windows 98.
- Applications continue to work when native code not available, but features reduce to standard Java features (i.e., extra mouse and wheel are not recognized).
- Provides access to all input device features such as mouse wheels.
- Extendable to support other input devices on other platforms.

Figure 4: Summary of MID features.

With input from multiple mice available, the KidPad design team can now concentrate on figuring out interaction rules for users sharing the KidPad workspace.

FUTURE WORK

As other input devices, such as tablets, become available for USB, we plan to add support for them. We also plan to make a fully functional MID available to other platforms. Our first focus is likely to be UNIX systems that run X Windows.

In addition, MID provides the possibility of providing more application-specific input events. For instance, an application could have a native voice-recognition system that generates events with the recognized voice data. Then, a cross-platform application could be written that accesses the voice data using MID events with a mechanism consistent with other input devices. Thus, our hope is that MID becomes a uniform architecture to support multi-modal input.

CONCLUSION

We feel MID will make it easier to build SDG applications by removing the painful problem of getting input from multiple devices. The fact that applications that use MID will run on all platforms supported by Java is an added advantage. Since MID's use is very similar to the use of Java mouse events, developers with Java experience should find it easy to incorporate MID into their applications. While we think that there is room for improvement, we believe MID already is a step in the right direction.

ACKNOWLEDGEMENTS

This work is motivated by our earlier work with Jason Stewart at the University of New Mexico and Allison Druin also at the University of New Mexico and now at the University of Maryland, and it was only because of our group experience with application-specific input device code that we finally came upon this notion of a general architecture to support SDG. In addition, Jon Meyer shared his experience writing Java code to get tablet input, and his early experiences were very helpful to us. Rajesh Iyer helped write an early version of the native code that helped us to understand how Microsoft DirectInput works.

The new version of KidPad and MID is supported in part by a wonderful collaborative project with several partners from the Swedish Institute of Computer Science (SICS) and the Royal Institute of Technology (KTH) in Stockholm as well as the University of Nottingham in England all funded by the European Union's *i3 Experimental Schools Initiative*. The current version of single user KidPad has been used in elementary schools in Stockholm and England, and we are grateful to the children there for their feedback and design ideas. We are eager to learn from them about how to better design SDG applications.

Finally, Bob Hummel at DARPA has been instrumental in supporting Jazz, and KidPad wouldn't exist without Jazz, which wouldn't exist without his support.

REFERENCES

1. Bederson, B. B., Hollan, J. D., Druin, A., Stewart, J., Rogers, D., & Proft, D. (1996). Local Tools: An Alternative to Tool Palettes. *In Proceedings of User Interface and Software Technology (UIST 96)* ACM Press, pp. 169-170.
2. Bederson, B. B., Hollan, J. D., Perlin, K., Meyer, J., Bacon, D., & Furnas, G. W. (1996). Pad++: A Zoomable Graphical Sketchpad for Exploring Alternate Interface Physics. *Journal of Visual Languages and Computing, 7*, 3-31.
3. Bederson, B. B., & McAlister, B. (1999). Jazz: An Extensible 2D+Zooming Graphics Toolkit in Java. *In Proceedings of User Interface and Software Technology (UIST 99)* ACM Press, (submitted).
4. Bier, E. A., & Freeman, S. (1991). MMM: A User Interface Architecture for Shared Editors on a Single Screen. *In Proceedings of User Interface and Software Technology (UIST 91)* ACM Press, pp. 79-86.
5. Bricker, L. J., Baker, M. J., & Tanimoto, S. L. (1997). Support for Cooperatively Controlled Objects in Multimedia Applications. *In Proceedings of Extended Abstracts of Human Factors in Computing Systems (CHI 97)* ACM Press, pp. 313-314.
6. Buxton, W., & Myers, B. A. (1986). A Study in Two-Handed Input. *In Proceedings of Human Factors in Computing Systems (CHI 86)* ACM Press, pp. 321-326.
7. Druin, A. (1999). *The Design of Children's Technology*. San Francisco, CA: Morgan Kaufmann.
8. Druin, A., Stewart, J., Proft, D., Bederson, B. B., & Hollan, J. D. (1997). KidPad: A Design Collaboration Between Children, Technologists, and Educators. *In Proceedings of Human Factors in Computing Systems (CHI 97)* ACM Press, pp. 463-470.
9. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software* . Addison-Wesley.
10. Inkpen, K., Booth, K. S., Klawe, M., & McGrenere, J. (1997). The Effect of Turn-Taking Protocols on Children's Learning in Mouse-Driven Collaborative Environments. *In Proceedings of Graphics Interface (GI 97)* Canadian Information Processing Society, pp. 138-145.
11. Kurtenbach, G., Fitzmaurice, G., Baudel, T., & Buxton, W. (1997). The Design of a GUI Paradigm Based on Tablets, Two-Hands, and Transparency. *In Proceedings of Human Factors in Computing Systems (CHI 97)* ACM Press, pp. 35-42.
12. Lee, S. K., Buxton, W., & Smith, K. C. (1985). A Multi-Touch Three Dimensional Touch-Sensitive Tablet. *In Proceedings of Human Factors in Computing Systems (CHI 85)* ACM Press, pp. 21-25.
13. Myers, B. A., Stiel, H., & Gargiulo, R. (In Press). Collaboration Using Multiple PDAs Connected to a PC. *In Proceedings of Computer Supported Collaborative Work (CSCW 98)* ACM Press,
14. Rekimoto, J. (1997). Pick-and-Drop: A Direct Manipulation Technique for Multiple Computer Environments. *In Proceedings of User Interface and Software Technology (UIST 97)* ACM Press, pp. 31-39.
15. Stewart, J. (1998). *Single Display Groupware*. Doctoral dissertation, University of New Mexico, Albuquerque, NM.
16. Stewart, J., Bederson, B. B., & Druin, A. (1999). Single Display Groupware: A Model for Co-Present Collaboration. *In Proceedings of Human Factors in Computing Systems (CHI 99)* ACM Press, p. (in press).
17. Stewart, J., Raybourn, E., Bederson, B. B., & Druin, A. (1998). When Two Hands Are Better Than One: Enhancing Collaboration Using Single Display Groupware. *In Proceedings of Extended Abstracts of Human Factors in Computing Systems (CHI 98)* ACM Press, pp. 287-288.