ABSTRACT
We describe the design of and experience with PointRight, a peer-to-peer pointer and keyboard redirection system that operates in multi-machine, multi-user environments. PointRight employs a geometric model for redirecting input across screens driven by multiple independent machines and operating systems. It was created for interactive workspaces that include large, shared displays and individual laptops, but is a general tool that supports many different configurations and modes of use. Although previous systems have provided for re-routing pointer and keyboard control, in this paper we present a more general and flexible system, along with an analysis of the types of re-binding that must be handled by any pointer re-direction system. This paper describes the system, the ways in which it has been used, and the lessons that have been learned from its use over the last two years.

KEYWORDS: Input redirection, ubiquitous computing, multi-display environments.

INTRODUCTION
PointRight is a peer-to-peer pointer and keyboard redirection system that operates in multi-machine, multi-user environments. It employs a geometric model for pointer motion across screens that is similar to conventional multi-headed displays, but redirects input across multiple independent machines and operating systems. We have developed it in conjunction with interactive workspaces that include large, shared displays and individual laptops, but it can be applied to any networked collection of machines and input devices.

Figure 1. The interactive room (iRoom)

BACKGROUND
The vision of ubiquitous computing [18], has come to fruition in the growing diversity of widely used computer hardware, including PDAs, large displays, wireless networks, and mobile devices of all kinds. But most of the software has borrowed from standard desktop operating systems and applications. More specifically, in most ubiquitous computing environments, the software is based on a one-to-one linking of a user with an input device (mouse and keyboard or stylus) and a single display. To create a more effective environment we need to go beyond this legacy and allow multiple users to share and control multiple devices in a more flexible way.
For many computer applications the display space provided by a single monitor or projected image is not adequate. This has been dealt with in all of the widely used windowing systems (MacOS, Windows, X-Windows) by allowing the user’s pointing device to operate in a geometric space that is tiled across the multiple monitors. Keyboard input follows window focus across these displays, which are managed as a single desktop.

Although this extension is valuable for individual work, it does not generalize to the case of multiple independent devices and users in an interactive workspace [9], a space where people come together to work collaboratively with assistance from computational resources, some of which are built into the environment and others of which are brought into the space in the form of laptops and PDAs [4]. One current example is the iRoom [9], shown in Figure 1, which we have used for the past 2 years. It has three 6’ diagonal back-projected touch-sensitive SMART Boards [15], a bottom-projected 5’ diagonal table-top display, a custom 7’ diagonal high-resolution 12-projector tiled back-projected display and a wireless LAN. Other examples include ILand [16], the MIT Intelligent Room [3] and Easy Living [2].

The iRoom software is based on a middleware layer called iROS [9], which enables the communication of events and information across all of the machines. The key iROS component is the Event Heap [8], which provides a blackboard-like communication mechanism. A primary design criterion was to support interaction across a collection of machines each running its own OS and applications. There is a dynamic binding between machines and displays. Each display can be switched to one of several of the room machines, or be connected to any laptop through a VGA cable.

In this more general situation, the multiple display surfaces are not all controlled by a single machine and there is no master input device to operate across them. Application suites in the iRoom, such as those used for construction management [11], make use of a collection of independent applications running on both shared and individual computers in the room. Their unification into a working environment depends on a general cross-device, cross-platform, multi-user facility that enables each user to control all of the devices with the conceptual simplicity of using multiple monitors tiled in a single geometric space.

To solve this problem we designed PointRight, a flexible input re-direction system. It allows pointer control from any input device to be re-directed from screen to screen as if all of the screens were a large virtual desktop, despite their being driven by different machines. The result is a cursor that the user moves seamlessly across the space of displays as though they were a single surface – when it reaches the edge of a screen it continues its motion onto the connecting edge of the adjacent screen in that direction. If there is no adjacent screen, it simply stops moving and remains at the edge of the screen. The input of any keyboard associated with the pointing device is directed to whatever machine is currently displaying the cursor.

PointRight allows for arbitrary topologies of rectangular screens, flexible mappings between machines and screens, and multiple simultaneously controlled cursors in a workspace. It keeps a map of the room’s topology, updating it dynamically to account for changes of mapping (which computer to which projector) and the status of each computer (running or not). Versions of the system have been in use for two years in our workspace, and have been indispensable. We have deployed the system in several other environments and are currently distributing a version as Open Source [7].

THE SYSTEM MODEL

Our initial experiences using the room revealed a number of desirable user interaction modes that were not supported by the basic software and hardware:

**Controlling multiple screens:** The SMART Board touch screens provide pointing input through direct touch (corresponding to left-button-down mouse). They have mechanisms for entering keystrokes with a virtual keyboard on the screen and right-mouse clicks through a physical switch on the pen tray below the board. There is a tray to hold an ordinary keyboard plugged into the computer driving the projector. In our room configuration, people at one screen or sitting at the table often want to be able to control what is on other screens. We initially had 3 separate wireless keyboards and mice corresponding to the three screens. This led to confusion as to what device controlled what, and required passing keyboards around as people moved from screen to screen or relinquished control to someone else.

**Controlling a device being projected on a touch screen:** A common way to present material that is on a laptop is to map the laptop’s VGA output onto one of the rear-projected touch screens. The presenter stands at the screen, and wants to be able to control the laptop by interacting directly with it through the touch screen.

**Controlling a shared screen seamlessly with a private one.** A key iROS facility, Multibrowse [10], makes it easy for a participant in a meeting to bring up any web page or application on any of the shared screens. Through a simple drag and drop action, a file or URL from the local machine appears on the shared display. Often the next step is to take further actions such as following links on the displayed page. This could be done by acquiring control of a wireless keyboard and mouse for that display, but it is much more natural to be able to use the laptop’s input devices in the same way you would if the shared screen were another monitor attached to the laptop.
In designing to satisfy these and other specific needs, we developed a general model that supports arbitrary topologies, multiple simultaneously active pointers, and flexible mappings between machines and screens.

**Redirection and multiple pointing**

We can divide the problem of many-to-many input into two relatively independent parts: one-to-many (directing input to one of a collection of computers), and many-to-one (handling multiple inputs coming in to a single computer). We have chosen to focus on the problem of one-to-many redirection, rather than on the handling of multiple concurrent input streams. Other projects have developed specialized software that handles concurrent multi-input operation \([1, 12]\). Our environment is based on supporting a wide variety of existing applications on standard operating systems, which in general do not have this kind of support. We allow the underlying OS to handle interleaved streams of mouse and keyboard events in its native way without augmentation. We have also focused on pointing as the key problem, adopting a simple strategy for keyboards: keystrokes are always directed to the current pointer redirection target.

**Flexible and Dynamic Topology**

Interactive workspaces will have a variety of topologies. Not only can screens tile a plane in arbitrary ways, but there may be folds and corners. The system needs to support arbitrary 3D manifolds composed of rectangular screens of different sizes and aspect ratios, providing smooth motion across all of them. For example, in our current iRoom setup, as shown in Figure 2, the left edge of the table display connects to the bottom edge of our front display while the top edge connects to the bottom of the middle SMART Board. Other mappings are possible, such as having each of the three SMART Boards connected to the corresponding third of the adjacent edge of the table screen.

**Types of Input**

There are some key distinctions that constrain the uses of different devices in an input redirection system. We identified three primary cases:

**Screen-bound:** (e.g., SMART Board, tablet computer, eBeam pen) When the pointing device operates directly on the display surface, there is a natural mapping from the pointing position to the corresponding pixel on the screen. For redirection to be successful, either the display needs to be redirected to match (e.g., through switching the projector input), or a segment of the screen displays an image of the redirection target (as with VNC [14]).

**Machine-bound:** (e.g., ordinary laptop and desktop). Most common pointing devices are associated with a specific screen without a fixed point-to-point mapping. Motion of the mouse, rather than its absolute position, controls the cursor location.

**Free-space:** (e.g., our room mouse and wireless keyboard). An input device can be provided in a workspace that has no intrinsic screen mapping. Of course a specific computer is receiving the inputs from such a device, but the display from that computer may not be visible in the workspace. A free-space device operates similarly to a machine-bound device, except that it is associated with the workspace as a whole. It also needs to have an initial location for the cursor on one of the displays.

PointRight allows screen-bound devices, such as the SMART Board touch screens, to control any machine whose display output is currently mapped to the corresponding display, maintaining a direct correspondence from the touch location to the display. Any other pointing device can be used as a universal pointer, regardless of whether it is bound to a specific machine or free-space. In the case of a machine-bound device, the user can explicitly choose (using a hot-key command) whether to allow the cursor to be redirected when it reaches the edge of the screen. When control is being redirected, the view on the local display is grayed out, since the redirected cursor no longer appears on the bound display.

**IMPLEMENTATION**

To participate in PointRight, a machine runs a PointRight application and is connected through the local network to the interactive workspace infrastructure. There are two components to the PointRight application: the *sender* redirects mouse and keyboard events from the local input device or devices while the *receiver* accepts remote pointer and keyboard events into the local event stream. Senders use a soft-state local database (see below) to maintain information about the current configuration. Receivers are responsible for receiving events and rescaling cursor motions to fit their particular display. Any machine running the PointRight software can operate as either sender or receiver.
receiver, but not both simultaneously, avoiding problems of recursive redirection, loops, etc.. The interaction is peer-to-peer with no centralized PointRight server.

The PointRight system currently runs on Windows 9x/NT/2000, and implementation is under way for Linux and Mac OSX. The code is available as open source [7], and there are binary installers available for Windows.

**Space Topology Description**

In order to control the movement between displays, senders maintain an internal representation of the current topology, which consists of components representing objects in the interactive workspace. They include:

**Screens:** the dimensions of the physical screen, a set of connections to other screens, a set of machines that can display video to the screen, the state of whether the screen is on or off, and which machine is currently driving the screen.

**Machines:** Machines are computers running the PointRight application and can therefore receive pointer events (they can also send events, but this information does not need to be shared globally). Data includes the rectangular pixel region for which the PointRight application on that machine is responsible. A multi-headed physical machine will therefore have multiple machine objects. There is a special machine called “laptop drop” that is dynamically remapped to the laptop currently connected to the drop-cable. This provides for laptop display to screens in the workspace.

**Connection:** A connection represents a valid transition between screens that pointers can traverse. They are represented by an edge (top, bottom, left, right) for each of the connected screens, and the region of the edge on each screen through which the pointer can transition. The region is defined by offsets that define the active region on the edge. For example in the configuration shown in Figure 2, 0'-4' on the top of table connects to 0'-5’ on the bottom of the middle of the display labeled “Smartboard Two.” Note that a left edge can be connected to a top or bottom, and screens flipped relative to each other are handled by reversing the order of the offsets relative to one another (as is the case with the connection between the table and the front screen in Figure 2). There can be multiple connections to a single screen edge as long as the edge regions for the connections do not overlap.

**Sender Implementation**

When a sender is started, it accesses the database to determine the configuration and status of the available machines. Currently it uses a shared, manually edited configuration file that specifies the basic topology of screens, machines, and connections in a text-based attribute-value format. Dynamic state information is maintained about screen state (on/off), machine state (on/off), and which machine is displaying to which display. We are implementing a more flexible database interface that will enable dynamic changes to other configuration aspects as well.

By monitoring events on the Event Heap, each sender maintains dynamic data on the state of machines and displays. Events are generated when a screen is switched on or off, a new machine becomes connected to a screen, or when the PointRight application is opened or closed on a machine. Events are posted and retrieved on the Event Heap server machine, and persist for a little more than two minutes. All active machines post events refreshing their current state every two minutes. If the senders do not see an event from a previously active machine or screen, the state of that object in the local database is updated to “off.”

This soft-state mechanism allows for the graceful handling of crashes, and also provides a simple mechanism for new senders that come up to acquire the current state of machines and screens in the interactive workspace.

There are two basic modes for the sender, one for screen-bound input, and one for normal (machine-bound or free-space) input. For screen-bound devices, when an event is posted that a new machine is being displayed on the screen, the sender begins forwarding the absolute position of the pointing input to the machine currently displaying to it. With normal input, the sender determines a virtual location for the pointer based on its own screen coordinates (extending beyond its visible screen) and uses the geometrical information to convert the absolute position into an appropriate target screen (based on the current state of connections of machines to screens) and a normalized position on that screen. If a machine is not currently running a PointRight receiver, or a display is off, then the sender skips over that display to the next one in the same direction. If no receiver is available in an indicated direction, then the cursor remains at the edge of the screen, just as it does on reaching the edge of the screen in a standard one-display system. Once a target screen is determined, a command to position the cursor on the machine displaying to that screen is sent.

**Receiver Implementation**

The receiver is configured with the area of the screen for which it is responsible. The sender passes an absolute value in a normalized range for x and y. These normalized values are scaled to the x and y range managed by the receiver, and the pointer on the receiver machine is set to this position. Keyboard events from the sender are inserted into the event queue on the receiving machine. Events coming from multiple senders are put into the queue in whatever order they arrive. Currently they are not tagged with information as to the sender, but that is a planned extension.
Communication
A set of communicating PointRight senders and receivers are on an IP wired or wireless network with access to a shared Event Heap server. The latency of the Event Heap is too variable to track cursor motion at guaranteed adequate rates, so a simple socket-based protocol was implemented for these events, which are sent directly from a sender to the current receiver. The Event Heap is used to communicate the changes of pointer location to a new screen/machine that lead to opening and closing these sockets (sockets are only opened if there is no existing connection to the new target machine). This means that as the system scales to more senders and receivers, the Event Heap is not a bottleneck since new socket connections rarely need to be created and the continuous stream of pointer updates goes over independent sockets. Keystrokes are communicated using the socket as well.

CONFIGURATIONS
The value of PointRight is its generality that enables it to be used in a variety of hardware and user configurations. It has been deployed in a number of different settings, which are now discussed.

iRoom
The primary development of PointRight has been in the iRoom [9], which incorporates large back-projected touch screens, a bottom-projected table and a wireless LAN. It is used for project meetings of 2 to 20 people, and participants often bring laptops with wireless communication that are used as part of the environment. The creation of PointRight was prompted by user problems in managing the multiple input devices in this environment. A wireless mouse and keyboard are now provided as free-space input devices, and are commonly used when keyboard input is needed for the large displays. PointRight is used on individual laptops to control applications on the large shared displays. Laptops are often projected onto one of the touch screen displays through the VGA cable and controlled from the touch screen.

PointRight has been in daily use for two years, and has become one of the key features of the iRoom. It has been used in all of the ways that were described in the earlier description of needs, and meetings are disrupted when it is not available. Fortunately, it has been relatively robust, and on the rare occasions when it is down, restarting takes only a minute or two. We were initially concerned with whether
the non-planar mapping of the left side of the table to the bottom of the front screen (as shown in Figure 2) would confuse users. We found, however, that they used it without hesitation or confusion.

Alternate iRooms
In addition to conducting our own project and course meetings in the iRoom, applications have been developed for other uses, such as construction management [11] and project-based education. We have installed several iRoom-like facilities for these projects with different configurations of devices. One uses three front-projected touch panels. Users have the choice of using the touch screen directly, thereby shadowing the screen, or using PointRight with a free-space device, so they don’t have to stand in front of the screen. The overall experience is that the bulk of “driving” the display is done from a PointRight sender away from the screen, but participants (not just the presenter) will often step to the screen for a moment to perform a simple action such as scrolling or following a link. Another room, the prototype for a project learning setting, has two front projection displays and no touch panels. PointRight on the wireless mouse and keyboard or on laptops is the primary input mode for that room.

iRoom2Go
Few environments provide the dedicated high-capacity facilities of an iRoom. We have found that PointRight is equally valuable in “low tech” environments using small amounts of standard equipment. The smallest configuration is a pair of computers (laptops or desktops) one of which runs the Event Heap server (which supports a variety of iROS capacities in addition to PointRight) and one or both of which run a PointRight sender or receiver. We have found it useful in a development environment, where a developer can use a single mouse and keyboard to interact with several machines without using a monitor switcher.

Writing Laboratory
PointRight was used to create an environment for group critiquing of writing. Laptops are provided for individual students and 5 large plasma panel displays are available for shared use by groups of three laptops. A single projected display is at the front of the room. The PointRight configuration is set up so that students within a group can simply move the cursor off the top of their laptop screens onto the bottom of the shared screen, which displays a document being jointly discussed. Moving off the top of the group screen, the cursor goes onto the projected room screen.

Ordinary word processing software is used on the shared screens, with no special treatment for multiple pointers. Since the students are engaged in face-to-face discussion, ordinary social protocols are quite appropriate and have proved adequate to avoid problems of concurrent action. This negotiation is invisible in the same sense that the movement of pointer from screen to screen is invisible. It does not interpose any explicit mechanisms, but simply meets natural expectations.

To create a mode where each student’s cursor moves directly from the laptop to the room screen instead of to a group plasma panel, the plasma panels can be turned off and the dynamic mapping automatically routes across them to the next available target, which is the room screen. When we have a dynamic topology database it will also be possible to accomplish this by changing the connections between screens.

Multi-board integration
Our iRoom includes a machine with multi-display output that can be used to display a single Windows desktop combining the three adjacent SMART Boards. Although it can be controlled through PointRight using a free-space or machine-bound mouse, the natural interaction is to use the touch screens, each of which now has a display that maps to one third of the desktop on the single machine. Although this is a somewhat specialized situation, the fact that the generalized PointRight mechanism can be used for it is indicative of the flexibility of the approach.

LESSONS
In working with these various configurations, we learned a number of lessons about what makes a cross-machine input redirection mechanism effective.

Simple installation. There is a single small application that can be installed with a simple installer on any computer to provide sender and receiver capabilities. When a machine starts up it simply needs to run the application to become a sender. This makes it realistic to use individual laptops as PointRight clients, even when they are not dedicated for use in the workspace. This use has turned out to be one of the most effective. Since all communication is through IP protocols, it should be easy to create versions for different operating systems.

Understanding display modes: Our initial implementations had a single way of treating display mapping. The distinctions between different kinds of displays led to distinguishing the functionality that works for screen-bound and other input devices. While this distinction is invisible to users, it allows them to interact with displays in a manner that is intuitive.

General model of machines, displays, and connections.
Since most previous pointer redirection programs have a simpler, fixed mapping from machines to their corresponding display devices, using a similar model for the flexible configurations available in interactive workspaces does not work. An extended model had to be created which considered a more open object model of the correspondence between computers, displays, devices, and regions. One aspect of this was refining the geometric model so that users can go across gaps and
Robust dynamic data. The soft state mechanism makes it possible for machines to react to system changes without interruption. In particular, if some machine goes down, the system adapts to the change within a couple of minutes without the need for explicit intervention. Unfortunately in today’s computing environment, this needs to be planned for as a routine event.

RELATED WORK
The previous systems that are closest to the user’s experience of PointRight are single-user systems with multiple monitors. PointRight opens up this functionality to the full array of displays, pointing devices, and keyboards associated with all of the computers in an interactive workspace.

When a PointRight sender is used on a visible screen such as a laptop, the system shares some characteristics with redirection systems that redirect pointer input from one machine to another. Remote interaction applications, such as VNC [14] provide an image of the remote screen, on which the pointer is moved. This differs from the PointRight metaphor of moving the pointer off the edge of one screen onto another. Systems more similar to PointRight include x2x [19] and x2vnc [6], which provide for configuration of an X-Windows machine such that mouse and keyboard control are redirected to another X-Windows machine or a machine running a VNC server. These are specific to X-Windows and do not support arbitrary topologies, allow dynamic changes based on machine state, allow multiple machines switched to a single screen, or provide for multiple simultaneous redirections.

The Mouse Anywhere capacity of Easy Living [2] allows a single mouse to control any of a number of devices, using the physical proximity of a person to a screen to provide the binding. To redirect the pointer input from screen to screen, the user physically moves from a location near one screen to a location near the other.

The Pebbles Remote Commander [12] provides for the case of controlling a single shared display from a collection of PDAs. The InfoTable and InfoWall [13] also provide a specialized case of the PointRight architecture. Control of the cursor is only from laptops, and both the laptop and the other displays can run only specialized Java applications, which hand off pointing and dragging events via Java RMI. This enables more sophisticated features such as “hyperdragging”, but is not applicable to the heterogeneous systems and legacy applications in an interactive workspace. No actual user experience with these systems has been reported. The use of dedicated applications is also the key to the iLand system [16] which builds workspace applications on a uniform Smalltalk software platform, and provides cross display pointer control to applications built on the framework.

A number of researchers have developed systems that provide for multiple people simultaneously using a single application [1]. PointRight does not assume the existence of special software such as PebblesDraw [12] or Beach [17] that coordinates the use of pointers by several users within a single application. Since separate sockets are opened for each sender, receiver applications could make use of the PointRight input mechanism as a source for distinct streams.

FUTURE WORK
We are continuing to develop and extend PointRight in several directions:

Dynamic topology update Currently, data about the interactive space is partitioned into two components: static and dynamic. Information about the size and location of displays, projectors, machines, etc. is stored in a file that is accessed by the senders when they start up. Information about the current state of each of these (whether projectors are on or off, what machine currently is feeding the VGA cable, etc.) is maintained dynamically through the soft state mechanism of the Event Heap. We are developing ways to interactively update the static information without having to edit a configuration file. Some of this can be done automatically (e.g., detecting new machines that are running a PointRight receiver). In the long run we can imagine a context-aware environment in which sensors report the location, identity, and orientation of each device, possibly extending the vision-based techniques used to locate laptops on the InfoTable [13] and to track people in Easy Living [2]. But in the meantime some changes require human input (e.g., reporting the new physical location of a screen that has moved).

Integration of the visual space. PointRight provides a unified multi-display space for pointing, but not for moving windows or other objects on the display. Unlike a single-processor multi-monitor desktop, it does not provide for dragging a window or other object from one screen to the next, or allow an object to be located straddling a boundary. Since the screens are actually independent desktops or applications, this would require mechanisms for handoff from one to the other and for managing simultaneous actions from two sources when an object overlaps two screens. This has been done in systems in which the entire screen is taken over by specialized applications in Java [13] and Smalltalk [17], but will require generalization to work in our open environment.
Concurrent Input.  All of the PointRight receivers we have experimented with so far have interleaved input events directed to them. Since events can be associated with the specific senders, it is possible for a multi-input application to handle the streams independently. We see this as best handled in the design of specific applications. In other work in our interactive workspace we have developed a large high-resolution interactive display based on a modeless interaction style that does not require a single input focus [5]. It was developed using a pen-based technology that supports only a single physical input device, but the underlying structure could be used to allow concurrent operation by multiple PointRight senders, and we hope to extend it in that way.

CONCLUSIONS
One of the first things we found when people began to use our prototype iRoom was that they needed an easy way to provide mouse and keyboard control to a large collection of displays. PointRight has evolved to fill that role and has also been deployed in several other settings with configurations substantially different from the iRoom.

Unlike the other systems we know of, it was designed to provide for heterogeneous operating systems and applications, combinations of fixed and mobile devices, and flexible layout topologies. It supports dynamic environments, multiple machines per screen, and multiple screens per machine, and allows for direct interaction input devices such as touch screens. The system requires only that an Event Heap be running in the interactive workspace, and that each machine that is going to participate, either as a source or target of pointer events, install a small application. Users need no special training or explanation, beyond a simple introduction to the idea. They find using PointRight to be intuitive and convenient.

PointRight is a part of the iROS software that we have made publicly available [7], and we expect to develop it further in response to feedback from a larger community of users.

Finally, the fluidity of using the PointRight system is difficult to convey without seeing it in action. A video of the system is available in streaming RealVideo format at http://graphics.stanford.edu/papers/pointright-uist2002/.

ACKNOWLEDGMENTS
The Interactive Workspaces project is the result of efforts by many students. Thanks to Bryn Forbes, and Rito Trevino in particular, who helped figure out how to tap into the Windows and Linux mouse and keyboard event systems, to Susan Shepard for her help making the video and keeping the iRoom stable enough to develop in, and to Aren Sandersen and Brian Luehrs, who have done the most recent implementation and maintenance. The work described here was supported by DoE grant B504665, and by donations of equipment and software from Intel Corp., InFocus, IBM Corp. and Microsoft Corp.

REFERENCES
1. Bier, Eric A., Steve Freeman, Ken Pier, MMM: The multi-device multi-user multi-editor, CHI'92


