ABSTRACT
The iStuff toolkit of physical devices, and the flexible software infrastructure to support it, were designed to simplify the exploration of novel interaction techniques in the post-desktop era of multiple users, devices, systems and applications collaborating in an interactive environment. The toolkit leverages an existing interactive workspace infrastructure, making it lightweight and platform independent. The supporting software framework includes a dynamically configurable intermediary to simplify the mapping of devices to applications. We describe the iStuff architecture and provide several examples of iStuff, organized into a taxonomy of ubiquitous computing interaction components. We conclude with some insights and experiences derived from using this toolkit and framework to prototype experimental interaction techniques for ubiquitous computing environments.

Keywords
User interface toolkits, ubiquitous computing, tangible user interfaces, input and interaction technologies, wireless devices, development tools, interactive room, prototyping, programming environments, intermediation, post-desktop user interfaces.

INTRODUCTION
While the mouse and keyboard have emerged as the predominant input devices for desktop computers, user input in ubiquitous computing (ubicomp) environments [14] presents a different set of challenges. A desktop environment is targeted for one user, one set of hardware, and a single point of focus. In a post-desktop, ubicomp environment, complexity is added in every direction; there are multiple displays, multiple input devices, multiple systems, multiple applications, and multiple concurrent users.

The iStuff toolkit was designed to support user interface prototyping in ubiquitous computing environments. Our domain is explicit interaction [1] with a room-sized environment consisting of displays of many sizes, plus support for wireless technology of various types, integrated using a common middleware. Our goal is to allow multiple, colocated users to fluidly interact with any of the displays and applications in the room, using for input any devices conveniently at hand.

The iStuff toolkit was implemented as part of the iRoom, which combines wall-sized displays with portable devices of many types to create a shared, interactive workspace. The toolkit was designed on top of iROS, a TCP- and Java-based middleware that allows multiple machines and applications to exchange information [9]. iROS supports communication through the Event Heap, a central server process that receives events from client applications in the room and redistributes them to the appropriate recipients.

The machines in the iRoom run standard operating systems and applications, rather than custom systems designed exclusively for the environment. Applications developed for the iRoom typically consist of suites of programs that combine their own UIs with interaction linked through the iROS. This approach allows for incremental deployment of complex systems, such as those developed for construction management [5]. However, it exposes a fundamental assumption of such operating systems—that each display comes with its own dedicated pointing device and keyboard.

The iStuff toolkit combines lightweight wireless input and output devices, such as buttons, sliders, wands, speakers, buzzers, microphones, etc., with their respective software proxies running on a machine in the iRoom in order to create iStuff components. Each component can be dynamically mapped to different applications running in the iRoom through a software intermediary called the PatchPanel.

This framework allows HCI researchers to quickly prototype a non-standard physical user interface and run experiments with it without running wires, soldering up components, or writing yet another serial device driver. Event communication takes only a few lines of Java code,
This paper describes the iStuff toolkit, and several examples of iStuff organized into a taxonomy of ubicomp interaction components. We conclude with some examples that illustrate how using iStuff facilitated prototype experimental user interfaces in the iRoom.

RELATED WORK
Ishii and Ullmer’s innovative work on Tangible Interfaces [6] used physical props to interact with computers. Their phicons (physical icons) were often specialized for particular applications, using a building-shaped object to manipulate a computer-generated map, or a car-shaped phicon to handle information from a web page about toy cars.

Greenberg and Fitchett’s Phidgets (physical widgets) [6], presented a more general toolkit of physical user interface components. Both iStuff and Phidgets provide a set of physical components that can be used to build more complex physical interfaces, as well as each providing a software interface that allows developers to integrate the components into their applications. However, the iStuff toolkit and accompanying software interface are designed to be particularly suitable for a ubiquitous computing environment. Mark Weiser’s landmark article [14] envisioned these environments as settings where computers of heterogeneous sizes and types were both plentiful and subtle, allowing computation to blend invisibly into daily activities. iStuff implicitly includes a software infrastructure, a programming model, and software engineering concepts that maximize flexibility for a toolkit deployed in multi-user, multi-application, multi-computer scenarios.

Abowd et al. proposed that interaction in ubiquitous computing settings can be divided into two subsections: implicit and explicit [1]. Work such as that of Salber et al. [12] explored the space of implicit interactions by creating context widgets that aided in the prototyping and development of “context-aware” ubiquitous computing applications. iStuff is targeted to address exploring explicit inter-action in cooperative, multi-device settings.

Olsen et al. [16] pointed out the need to decouple user interfaces from services in interactive rooms and similar environments, and propose XWeb, a web-based architecture to interact with services using a wide variety of input modalities. Taylor et al. [13] developed a software architecture that resembles iStuff and applied it to GUI software for larger-grain reuse and flexible system composition. Bleser’s Toto [3] realized the importance of the type of flexibility our PatchPanel intermediary provides—it included several “candidate technique actions” for a variety of tasks. Beaudouin-Lefon’s concept of “Degree of Integration” [2] discussed the mapping of devices to tasks that require a different number of dimensions than the device offers (for instance, when mapping a 2-D device like a wireless mouse to provide control for a 1-D slider). Our intermediary software allows for dynamic re-mappings of devices to events, and handles the transformations and normalizations this task requires.

iSTUFF ARCHITECTURE
To achieve the goal of a convenient toolkit for physical UI prototyping in the iRoom, we developed the following requirements:

- Flexible, lightweight components. Devices can be simple with minimal computational complexity built into them.
- Platform independence with cross-platform capabilities. This provides maximum flexibility in a typically heterogeneous ubicomp environment.
- Protocol independence. We want to support as broad a range of wireless protocols and input technologies as possible.
- Ease of integration with existing applications. We want to simplify the programmer’s task of using iStuff.
- Support for multiple simultaneous users.

To meet these requirements, we created the iStuff architecture, which consists of iStuff components that provide the physical toolkit of wireless input and output devices, asynchronous communication based on iROS Events, and the PatchPanel intermediary to dynamically re-map events to applications. This architecture is summarized in Figure 1. Each element is described in further detail below.

![iStuff Architecture](image)

**Figure 1:** iStuff architecture diagram.

**iStuff Components**

iStuff components consist of wireless devices paired with a machine connected to the Event Heap that has a transceiver and related software and serves as a proxy to the room. Both device and proxy are required for an iStuff component, although multiple iStuff devices can share a proxy. This design isolates most of the “smarts” in the proxy, allowing the physical devices to be very lightweight, simple components. For example, most of our custom-built iStuff is based on a simple RF transmitter/receiver connected through a USB port to a PC proxy. The iStuff devices contain inexpensive chips that match the transmitter/receiver plus simple input and output hardware such as buttons, sliders, buzzers and lights.
All that is necessary for a physical device to become an iStuff component is a proxy that encapsulates data into an event (or extracts data from it), making iStuff independent of any particular wireless protocol or technology. This architecture also made it easy to assimilate off-the-shelf hardware technologies like X10 (www.x10.com), the Anoto Pen (www.anoto.com), or even a wireless mouse into the iStuff family.

This division into device and proxy makes iStuff easy to construct and reproduce, lightweight, inexpensive, and extensible to a wide variety of protocols and technologies.

**Event Communication**

iStuff components communicate with applications using events, as supported by the iROS infrastructure. Conceptually, an event is a message or a tuple that contains a type and an optional number of fields containing key-value pairs. Producers post events to the Event Heap, and consumers register to receive events, specifying the event type and, optionally, other criteria based on matching the content of specific fields. This creates a communications mechanism that extends the notion of an event queue to an entire interactive room, with multiple machines and users. It is designed specifically to be robust against failure, and to support easy restarting of arbitrary parts of the system (including the central Event Heap itself). The iROS implementation is primarily in Java, to make it platform-independent, and is available in Open Source distribution from iros.sourceforge.net.

An iStuff component is associated with an iStuff event. However, rather than working directly with iStuff component events, application programmers are encouraged to create their own abstracted event types and to use the PatchPanel to translate between iStuff events and application-specific events. For example, instead of expecting input such as “GetMousePosition,” an iStuff application may expect a “NewPositionEvent.” This event can be supplied by a mouse, a touch panel, a slider, or a set of wireless wands, depending upon the current PatchPanel configuration. Similarly, an application can provide feedback with a “FeedbackEvent.” This can be translated into an event that creates a sound, a light or even graphical feedback on another display. iROS and its Event Heap were designed to efficiently support such intermediation, making them an ideal platform for the iStuff toolkit.

**PatchPanel**

The original version of iStuff did not include a PatchPanel, but we quickly found that this is a critical component for flexible prototyping. For example, we created an application called iPong, modeled after the original arcade game Pong, but designed to span multiple displays and machines. It listened for device-level mouse input so its paddles could be moved with a mouse or a touchpanel. To map an iSlider to a paddle required changing iPong to listen for iSlider events. To make it listen instead to a wand would require another change. To solve this problem, iPong was redesigned to listen for a “MovePaddle” event. The PatchPanel is then used to map suitable iStuff events to MovePaddle events.

The PatchPanel consists of an intermediary application that implements event mapping, and one or more GUIs that provide a user-accessible way to configure events. The PatchPanel architecture is shown in Figure 2.

**PatchPanel Architecture**

![PatchPanel Architecture](image)

**PatchPanel Intermediary**

The PatchPanel intermediary exists as an Event Heap client that non-destructively translates events from one type to another. It listens for all events, translating those that match its event mapping configuration. The configuration itself is updated by sending Event Heap events to the intermediary, which allows any program to dynamically reconfigure event mappings. To create a mapping through the intermediary, the user must generate an event of type IntermediaryConfigEvent with the appropriate fields that represent the event and its mapping. When the intermediary receives a new IntermediaryConfigEvent, it updates its internal translation look-up structure.

The simplest event mapping matches only the event type, generating one complete event from another. Another common mapping matches both the EventType and a unique ID field, to discriminate, for example, one iButton from another.

To support coordinate system translation, the intermediary allows the specification of simple arithmetic expressions (affine transformations) to convert fields from an incoming event to values in an outgoing event. For example, the iStuff slider event has a field that specifies its current value, which must be rescaled to map to the correct location in an iPong MovePaddle event.

**PatchPanel GUI**

The PatchPanel GUI presents the user with a graphical tool for creating event mappings. After the user specifies a particular event translation, the PatchPanel GUI posts an IntermediaryConfigEvent to update the Intermediary. This interface allows an experimenter to combine existing iStuff components to prototype a new physical device and to map
that device to an existing application without writing any code.

Because of the intermediary’s event-based API, the PatchPanel GUI is completely independent from the intermediary and may be running on a separate machine connected to the same Event Heap. Often, it is convenient to make the GUI web-based, for easy access. The GUI can be general, or customized for a specific application, as described in the meeting capture example below.

**PatchPanel Example**

The Super Slider is a device that is built from a combination of multiple iStuff components: an iSlider based on RF technology and a pair of iButtons based on X10 technology. We want to configure these to create a slider that alternately drives the left and right paddles of iPong. The buttons are used to dynamically select which paddle is being driven by the iSlider.

The iSlider and iButton both produce events of type “iStuffInputEvent” with some common fields including “DeviceType” and an “ID” field that is a unique device identifier. The iSlider device also has the fields “Value”, “Max”, and “Min” which correspond to the current, maximum, and minimum value respectively for the particular hardware being used.

The iPong application developer has defined a set of events of type “iPongEvent,” one of which has the subtype field “MovePaddle.” This event also contains the fields: “Side,” a string that specifies left or right paddle, and “Yloc,” an integer specifying the location on the Y-axis within a fixed range of 0-700.

The basic translation that maps an iSlider to a MovePaddle event first matches any iStuffInputEvent whose Device is Slider. It then creates an iPongEvent whose subtype is MovePaddle. The Yloc field is defined as an expression, (Value-Min)*700/(Max-Min), where Value, Min and Max are all fields of the iStuffInputEvent.

The translation can be expressed as a string that is included as a field in an event of type “IntermediaryConfigEvent” which is sent to the PatchPanel intermediary to update the configuration.

To have the iButtons dynamically map the iSlider to the left or right paddles, the iButton events are mapped to IntermediaryConfigEvents that express the above mapping and place the desired side in the “Side” field of the translated MovePaddle Event. The end user can then alternately manipulate the left and right paddles with the iSlider by pushing the corresponding iButton.

**iStuff Examples and Taxonomy**

Several different iStuff components have been implemented in our lab using both homemade devices and off-the-shelf devices, as shown in Figure 3. For interested parties, our designs are freely available from our website.

In this section, we will briefly describe these iStuff components, then arrange them into a general taxonomy for ubi-comp interaction devices. While preliminary, this taxonomy provides both a better understanding of the breadth of iStuff, and indicates areas left unexplored.

**Figure 3: Examples of iStuff input and output components**

**iButton:** This is the most basic binary input component that is an essential building block for many different physical user interfaces. One style of iButton has been implemented using homemade circuitry and a garage-door-opener style radio frequency (RF) transmitter. Another style of iButton has been implemented using commercially available X10 keychain remotes.

**iSlider and iKnob:** These are one-dimensional input components that report absolute (iSlider) or relative (iKnob) position over a fixed axis. They have also been implemented using homemade circuitry coupled with an RF transmitter.

**iMouse:** The iMouse is a standard off-the-shelf Logitech wireless mouse. Its iStuff proxy converts mouse motion into iStuff events sent to the Event Heap. Any application connected to the Event Heap can therefore receive input from the mouse. We have extended the system with events to allow passing the mouse cursor between multiple displays, making the iMouse a room-wide pointing device similar to [10]. This also allows single applications to listen to the iMouse in addition to other input device streams, removing the barrier of “one user with one set of input devices” that is engrained in desktop computing hardware, operating systems, and applications.
These are binary output components, implemented using homemade circuitry and an RF transmitter. They provide visual (iLight), audio (iBuzzer) and haptic (iVibe) output.

iPen: The iPen is a component that supports handwriting input. This is implemented using the Anoto pen, which is a commercially available Bluetooth device combined with specialized paper.

iMike: This is a voice based input component, implemented using a wireless microphone coupled with a proxy containing the IBM WebSphere Voice Server [15] speech recognition engine that supports VoiceXML menu definitions. As voice commands are recognized, events are generated and posted to the event heap.

iLight, iBuzzer and iVibe: These are binary output components, implemented using off-the-shelf infrared MIDI wands (Lightning II, www.buchla.com).

iWand: This is a two-dimensional input component that reports absolute position over a fixed 2-D space. The iWand is implemented using off-the-shelf infrared MIDI wands (Lightning II, www.buchla.com).

iPen: The iPen is a component that supports handwriting input. This is implemented using the Anoto pen, which is a commercially available Bluetooth device combined with specialized paper.

As voice commands are recognized, events are generated and posted to the event heap.

iSpeakerm: This is a voice based input component, implemented using a wireless microphone coupled with a proxy containing the IBM WebSphere Voice Server [15] speech recognition engine that supports VoiceXML menu definitions. As voice commands are recognized, events are generated and posted to the event heap.

These components, implemented using homemade circuitry and an RF transmitter, provide visual (iLight), audio (iBuzzer) and haptic (iVibe) output.

iPen: This is a component that supports handwriting input. This is implemented using the Anoto pen, which is a commercially available Bluetooth device combined with specialized paper.

iWand: This is a two-dimensional input component that reports absolute position over a fixed 2-D space. The iWand is implemented using off-the-shelf infrared MIDI wands (Lightning II, www.buchla.com).

Taxonomy

By classifying iStuff into several categories, our goal is to create a design space for ubicomp interfaces. Using this taxonomy, we are able to pinpoint gaps in the breadth of our toolkit and mark them as areas for future development. Based on earlier work by Buxton [17], and Card et al. [4] proposed a classification scheme for input devices, classifying such devices by the axes they moved along (either linear or rotary), whether they reported position/rotation, changes in position/rotation, force/torque, and/or changes in force/torque, and the range of values they provided (from a single value to an infinite range). However, we found this scheme too narrow for describing iStuff, as it does not classify devices of varying modalities (such as speech), nor does it provide for the classification of output devices.

We propose a six-part taxonomy for ubicomp interaction components such as iStuff: direction, modality used/sense addressed, resolution, dimensions, relative vs. absolute, and invasiveness. Other attributes are possible and were considered such as footprint and a measure of mobility, but these generate an interesting design space.

Direction

This attribute indicates whether a device is used to provide input, output, or both.

Modality Used/Sense Addressed

For input devices, this attribute describes the modality(ies) used to operate an input device—manual (such as a mouse or stylus), visual (such as eye-tracking input), acoustic (such as sound or speech input), thermal (heat sensors), etc. For output components, this describes the sense(s) which perceive the output—visual (LEDs, displays), auditory (noise or speech output), haptic (force, temperature changes), etc.

Resolution

For an input device, resolution is analogous to Card et al.’s property that classifies the domain provided by the device as ranging from a single, binary value to an infinite range of values. For output devices, the interpretation of resolution varies depending on the sense addressed. For visual output, it is sensible to discuss resolution in terms of number of pixels, levels of brightness, and/or number of colors. Resolution of auditory devices can range from one-bit (as in a buzzer) to near-infinite (as in a speaker). For haptic feedback, it makes sense to discuss whether a binary value (presence/absence of feedback) or a range of values are provided.

Dimensions

For manual input and visual output devices, the familiar concepts of 0, 1, 2, and 3D are applicable. Upon inspection, such concepts apply to other modalities as well—for instance, sound output could provide 3D information if high-quality “surround sound” speakers were used to provide a sense of location to the sound. Similarly, vocal input could carry with it dimensional information if triangulation techniques were used to pinpoint the location of the speaker.

Relative vs. Absolute

This concept applies not only in the familiar domain of manual input (with a stylus providing absolute positional information while a mouse provides the relative variety), but to other domains/directions as well. For instance, an audio output device could be absolute, conveying the presence or absence of a sound, or it could be relative, conveying a change in pitch.

Invasiveness

Invasiveness describes how cumbersome some various technologies are to the user. Output devices such as head-mounted displays or input devices such as gloves would be classified as highly invasive, while output devices such as room-based speakers or input devices such as video camera based gesture recognition would be considered minimally invasive, since they do not require the user to don any special equipment. Devices such as a stylus, which require more equipment than the user’s own body, but which are
Table 1 classifies the iStuff devices we have implemented thus far. Inspection of the table yields directions for future work—for instance, developing more non-manual input devices and non-visual output devices.

<table>
<thead>
<tr>
<th>Device</th>
<th>Direction</th>
<th>Modality Used/Sense Addressed</th>
<th>Resolution</th>
<th>Dimensions</th>
<th>Relative vs. Absolute</th>
<th>Invasiveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>iButton</td>
<td>Input</td>
<td>Manual</td>
<td>Binary</td>
<td>0D</td>
<td>Absolute</td>
<td>Low</td>
</tr>
<tr>
<td>iKnob</td>
<td>Input</td>
<td>Manual</td>
<td>Infinite</td>
<td>1D</td>
<td>Relative</td>
<td>Low</td>
</tr>
<tr>
<td>iSlider</td>
<td>Input</td>
<td>Manual</td>
<td>Fixed Range</td>
<td>1D</td>
<td>Absolute</td>
<td>Low</td>
</tr>
<tr>
<td>iMike</td>
<td>Input</td>
<td>Auditory</td>
<td>Infinite</td>
<td>1D</td>
<td>Absolute</td>
<td>Minimal</td>
</tr>
<tr>
<td>iMouse</td>
<td>Input</td>
<td>Manual</td>
<td>Infinite</td>
<td>2D</td>
<td>Relative</td>
<td>Low</td>
</tr>
<tr>
<td>iPen</td>
<td>Input</td>
<td>Manual</td>
<td>Infinite</td>
<td>2D</td>
<td>Absolute</td>
<td>Low</td>
</tr>
<tr>
<td>iWand</td>
<td>Input</td>
<td>Manual</td>
<td>Fixed Range</td>
<td>2D</td>
<td>Absolute</td>
<td>Low</td>
</tr>
<tr>
<td>iBuzzer</td>
<td>Output</td>
<td>Auditory</td>
<td>Tones</td>
<td>0D</td>
<td>Absolute</td>
<td>Minimal</td>
</tr>
<tr>
<td>iLight</td>
<td>Output</td>
<td>Visual</td>
<td>Tones</td>
<td>0D</td>
<td>Absolute</td>
<td>Minimal</td>
</tr>
<tr>
<td>iSpeaker</td>
<td>Output</td>
<td>Auditory</td>
<td>Natural Language</td>
<td>1D</td>
<td>Absolute</td>
<td>Minimal</td>
</tr>
<tr>
<td>iVibe</td>
<td>Output</td>
<td>Haptic</td>
<td>On/Off</td>
<td>0D</td>
<td>Absolute</td>
<td>Low</td>
</tr>
</tbody>
</table>

small and do not need to be attached to the user, recalibrated, etc., have a low invasiveness value.

EXAMPLES OF USE
The effectiveness of the toolkit was tested by making the components available to researchers in the iRoom. In addition to the intended purpose of quickly combining components to prototype devices, iStuff was used by developers and researchers in the room to explore several different aspects of physical interaction including physical form factor, augmenting application GUIs, and in some cases replacing application GUIs.

Encapsulating Events
The iROS infrastructure itself provides several services to the room. One of these services, known as iCrafter [11], exposes software interfaces via the Event Heap to objects and applications in the room. iButtons were quickly recognized as a convenient interaction medium to activate these services, such as turning on the lights, launching web pages, or starting applications on the iRoom displays. For example, an iButton was configured to “start the room” which meant turning on all the lights and projectors, launching the standard applications and opening a help page—a good example of how the one-to-many mapping feature of the intermediary can be used. The ease of this configuration task should be emphasized—a start the room button can be configured in approximately 30 seconds using a web-based patch panel, and requires no further coding.

iPong and the Super Slider
iPong and the Super Slider have been described in detail already. iPong was an early iRoom demo, but it was easy to expand it to include iStuff as it was already enabled for Event Heap communication. The Super Slider is one of our most complex iStuff examples, showing a set of iStuff components integrated into a single, dynamic UI.

Meeting Capture Software
iStuff was used to add functionality to another research project in the iRoom. One of the developers in the room has been working on a meeting capture program. During user studies, participants expressed a desire to discretely annotate important moments in the meeting for use during the post-meeting review. They felt a type-in window would make it too obvious they were adding an annotation, which might be disruptive. We were able to integrate iButtons into this application by mapping them to an event implemented by the meeting capture software.

A new PatchPanel GUI was created to specifically customize iButtons for this application. A web-based servlet was created that contained a single user input field for the meeting participants to enter a name. After submitting the name, the web page instructed them to select and press their personal iButton for the meeting. The servlet subscribed for the next iButton event and automatically mapped that particular iButton to the name entered just prior to the button press. This specialized GUI made it very easy for non-technical meeting participants who had no background knowledge of iStuff to map event translations in the intermediary for their particular task.
Right Button for the SMART Boards

SMART Boards (www.smarttech.com) with touch panels suffer from having no convenient right-click facility. We created the iStylus to solve this problem. The iStylus includes an embedded iButton that supplies a right click when pressed.

iDog

A developer was inspired by the size of the iButton circuitry to incorporate it into a small stuffed dog, creating the iDog. The button switch was replaced with a gravity switch so that every time the dog was turned over the switch was activated. The iDog had no intended purpose, but has been creatively configured by other room developers through the PatchPanel to ‘bark’ by playing a sound out the iSpeaker. With a smaller iSpeaker (or a larger dog!) the two could be packaged together to create a self-contained physical device whose ‘smarts’ reside externally in the room that contains it. The iDog is an important example because it was created in an attempt to inspire applications—inpiring the development of novel interfaces was one of the original goals of the iStuff project.

iClub

A rambunctious group of computer science undergraduates decided to use the infrastructure in our interactive room to develop their senior design project, transforming the room into an interactive dance club. The students used iStuff to create physical interaction mechanisms so that “clubbers” could participate in music creation. They chose to use the iSlider to control a high-frequency filtering mechanism for the music playing in the room. iStuff allowed the students to quickly and easily add a physical interface late in the design of the iClub.

iWall

The iWall is a distributed whiteboard application we have created that allows multiple people using different cursors to interact with different images and other graphical objects on multiple machines and displays. It is an experimental application written specifically for exploring multi-user interaction.

One set of experiments will involve different ways to move virtual objects around the iRoom. For example, a group of users may collaboratively solve an interactive jigsaw puzzle.

The iWall supports multiple users and cursors by associating each cursor with a unique cursor ID. The cursor event format the iWall is expecting contains a field that specifies this cursor ID. iStuff is a key enabler of this software interaction model because it provides each user a physical interaction device of their choice that can easily be configured to a different cursor. One user may use a mouse, another user may use an iWand, and yet another user may use a prototyped device that is a combination of two iKnobs. Each device can be mapped to different cursors that can all co-exist on a single display.

DISCUSSION

The iStuff toolkit and its supporting infrastructure have allowed us to implement a wide range of different physical devices, and to start exploring post-desktop input metaphors for ubiquitous computing environments. In this section we will discuss some of the insights we have developed using iStuff.

Intermediation is valuable

The intermediary provides a conceptual layer that completely decouples applications from their physical I/O devices and allows each to evolve independently. In other words, the format of events created (or received) by iStuff components is completely independent of application event format. Changing a device event field structure does not affect the application event format that it is translated to (or from), but only modifies the translation mapping.

The intermediary was designed so that the event translation mappings are dynamically reconfigurable at run time. Not only does this simplify prototyping, it enables real time remappings, and can thus change the “focus” of a device or application.

Latency is inevitable

The iStuff toolkit depends on network communication, as do all ubiquitous computing environments. Latency acceptable for most network communication can be unacceptable for user input.

We have done some stress-testing in our environment to begin to understand the limitations of this issue. The most demanding example is the iStuff implementation of PointRight. Under normal conditions, 5-6 users can operate PointRight simultaneously without noticeable delays, which is sufficient for experimentation. However, simultaneously downloading a movie will slow the entire network significantly, making PointRight unusable.

The issue of latency is inevitable in ubiquitous computing because of its distributed nature. Although latency can be minimized, it must be tolerated at some level in ubiquitous computing environments.

The importance of social protocols

When multiple users each have their own input devices, it is inevitable that they may get in each other’s way. While some UI techniques are inherently more “orderly” than others, we believe that ultimately social protocols will be a key component of UI design in ubicomp environments. For example, using PointRight, it is possible for two people to try and operate on the same content. Rather than designing elaborate locking systems that enforce turn taking, we believe that relying on social protocols is often sufficient.
**iStuff is not just for the iRoom**

While iStuff requires the iROS, the iROS does not require an iRoom to be effective. It will run quite happily on a single machine, or perhaps more interestingly, on a collection of laptops or desktop machines. iStuff can thus be developed outside of the iRoom, and applied to any networked environments willing to run the iROS.

**FUTURE WORK**

We hope to continue aiding third parties in developing physical user interfaces, including incorporating iStuff into an HCI design course or a Mechanical Engineering design course. We will also submit the Intermediary and Patch-Panel GUI to be a part of the open source release of iROS. We intend to use the iWall to perform user studies on how people interact in a multi-user, multi-screen, multi-device environment. In addition we intend to continue expanding the iStuff component family to make the device spectrum more complete, turning the taxonomy into a design space. Lastly, we intend to prototype new novel interaction techniques in our interactive workspace using iStuff.

**CONCLUSIONS**

In summary, the iStuff toolkit has proved to be a flexible prototyping platform for post-desktop ubiquitous computing interaction. This paper describes a number of iStuff components and their application. An important aspect of the iStuff framework is the flexibility the PatchPanel intermediary provides. We hope that the iStuff toolkit and framework will help us and others to further explore and systematically study interaction techniques for ubiquitous computing environments, to help uncover what will be the WIMP interface of the post-desktop era.

**ACKNOWLEDGMENTS**

We have removed some important acknowledgements to maintain anonymity.

This material is based upon work supported under a National Science Foundation Graduate Research Fellowship and the Wallenberg Foundation. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the funding agencies.

**REFERENCES**