iStuff: A Physical User Interface Toolkit for Ubiquitous Computing Environments

Rafael Ballagas, Meredith Ringel, Maureen Stone[1], Jan Borchers
Computer Science Department
Stanford University
Stanford, CA 94305, USA
ballagas@stanford.edu, {merrie, borchers}@cs.stanford.edu, stone@stonesc.com

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 design space of ubiquitous computing interaction components. The main contribution is a physical toolkit for distributed, heterogeneous environments with run-time retargetable device data flow. 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, prototyping, programming environments, intermediation.

INTRODUCTION
While the mouse and keyboard have emerged as the predominant input devices for desktop computers, user input in ubiquitous computing (ubicomp) environments [19] 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, co-located users to fluidly interact with any of the displays and applications in augmented environments such as the Stanford iRoom, using for input and output any devices conveniently at hand.

The toolkit was designed on top of *iROS*, a TCP- and Java-based middleware that allows multiple machines and applications to exchange information [11]. 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 [6]. 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 using it without having to run wires, solder up components, or write yet another serial device driver. Event communication takes only a few lines of platform-independent Java code, making it easy for applications to become iStuff-enabled.

It should be emphasized that this paper discusses a physical toolkit, aimed at rapidly prototyping and building physical devices. This is fundamentally different from a GUI toolkit, which is aimed at rapidly prototyping and building graphical applications. For example, GUI toolkits have a

[1] StoneSoup Consulting, 191 Pine Lane, Los Altos, CA 94022

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or redistribute to lists, requires prior specific permission and/or a fee.
CHI 2003, April 5-10, 2003, Ft. Lauderdale, Florida, USA.
Copyright 2003 ACM 1-58113-630-7/03/0004…$5.00.

well-defined display space and 2-D position, but these characteristics are not necessarily true for a physical domain. Early work on graphical user interfaces by Foley et al. [7] that has been widely accepted proposed that the set of interaction tasks sufficient for describing interaction with graphical interfaces are {select, position, orient, path, quantify, and text}. However, this classification only applies to graphical UI's, not physical UI's, or UI's in general.

This paper describes the iStuff toolkit, and several examples of iStuff organized into a design space of ubicomp interaction components. We conclude with some examples that illustrate how using iStuff facilitates prototyping experimental user interfaces in the iRoom, and how it prompts us to think about new modes and issues in post-desktop user interfaces.

RELATED WORK

Ishii and Ullmer's innovative work on Tangible Interfaces [9] used physical props to interact with computers. Their *phicons* (physical icons) were often specialized for particular applications, and did not provide a generic model or toolkit for building these physical interfaces. Greenberg and Fitchett's *Phidgets* (physical widgets) [8] represent a 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, and they each provide 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 where computers of heterogeneous sizes and types are both plentiful and subtle, allowing computation to blend invisibly into daily activities [19]. 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. [17] 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 towards exploring explicit interaction in cooperative, multi-device settings.

Myers [13] demonstrated that abstracting device input away from application level code is useful in the Garnet project using Interactors. However, he did not provide solutions for prototyping devices, distributed environments, or handling output. Olsen et al. [15] pointed out the need to decouple user interfaces from services in interactive rooms and similar environments, and proposed XWeb, a web-based architecture to interact with services using a wide variety of input modalities. Myers and Kosbie [14] show how hierarchical event structures can provide higher-grain code reuse, and how to abstract low-level events to application-level events. These concepts are reapplied in iStuff with the addition of run-time retargetable event flow.

Taylor et al. [18] developed C2, a message-based software architecture , and applied it to GUI software for larger-grain reuse and flexible system composition. Taylor introduced the concept of domain translation in which messages from components are translated to address mismatches in message names and parameters. iStuff differs from the C2 architecture by providing a generic software model for I/O including a centralized domain translator known as the PatchPanel. The PatchPanel adds a higher level of flexibility than C2 through run-time translation specification.

Bleser's Toto [3], a GUI toolkit, realized the importance of the type of flexibility our PatchPanel intermediary provides—it included several "candidate technique actions" for a variety of tasks. Beaudouin-Lafon'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 to address dimension mismatches.

Classification schemes for input devices have been proposed earlier by Buxton [4] and Card et al. [5], 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 provide for the classification of output devices.

ISTUFF ARCHITECTURE

To create a convenient toolkit for physical UI prototyping in the iRoom, we developed the following requirements:

- Flexible, lightweight devices.
- Platform independence and cross-platform capabilities.
- Wireless protocol independence.
- Ease of integration with existing applications.
- 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.

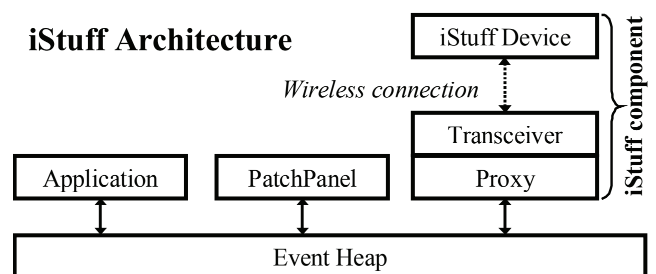

Figure 1: iStuff architecture diagram.

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 simple and lightweight. 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 http://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 that make sense in their application, 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 some 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. A layer of abstraction is necessary for toolkit flexibility and reuse as demonstrated in [13,14,18]. 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 touch panel. 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 rewritten to listen for a "MovePaddle" event. The PatchPanel is then used to map suitable iStuff events to MovePaddle events. The abstraction allows for devices and applications to evolve independently, allowing new devices to be seamlessly introduced.

A more critical component than abstraction for physical UI's is dynamic flow control or run-time retargetability that the PatchPanel provides. This allows for users to change the "focus" of input or output. In a modern desktop environment, the floating cursor is used to bring windows and objects into focus to direct user keyboard input. This model does not extend to a distributed physical space where a GUI doesn't necessarily exist. The PatchPanel enables the same redirected I/O functionality.

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.

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 to translate 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 scenario in the "Examples Of Use" section later.

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. Note that this dynamic reconfiguration of the PatchPanel requires an architecture that allows rerouting events at runtime, something the iStuff framework supports in contrast to toolkits such as [13,14,18].

ISTUFF COMPONENTS AND DESIGN SPACE

Several different iStuff components have been implemented in our lab using both homemade devices and off-the-shelf devices, as shown in Figure 2. For interested parties, our designs are freely available from our website (http://istuff.stanford.edu/). In this section, we will briefly describe a subset of iStuff components, then arrange them into a general design space for ubicomp interaction devices. While preliminary, this design space provides a better understanding of the breadth of iStuff and indicates areas left unexplored so far.

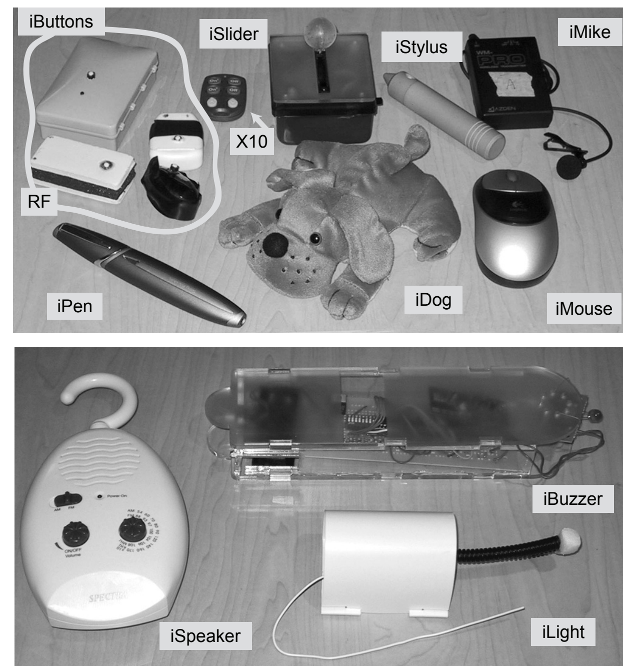

Figure 2: Examples of iStuff input/output components

iButton: This is the most basic binary input component and 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 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 [12]. 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 de-

vices" that is engrained in desktop computing hardware, operating systems, and applications.

iWand: This is an input component that reports absolute position over a fixed 2-D space. The iWand is implemented using off-the-shelf infrared MIDI wands.

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

iMike: This is a voice input component, implemented using a wireless microphone coupled with a proxy containing the IBM WebSphere Voice Server [20] 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 homemade circuitry and an RF transmitter. They provide visual (iLight), audio (iBuzzer) and haptic (iVibe) output.

iSpeaker: This is a continuous audio output component. The PC proxy runs a daemon that accepts both text strings for text-to-speech translation, and links to audio files to play. The daemon then sends the audio signal to the sound card of the proxy, which is connected to a commercially available FM transmitter. The wireless speaker itself is simply an off-the-shelf portable FM radio tuned to the appropriate frequency. Despite this low-tech construction, the iSpeaker appears to applications as a mobile speaker.

Design Space

By classifying iStuff into several categories (Figure 3), our goal is to define 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. This format builds upon earlier work in device design spaces and classification mentioned in [4,5].

We propose a five-part space for ubicomp interaction components such as iStuff: direction, modality used/sense addressed, resolution, dimensions, and relative vs. absolute. We describe each of these dimensions in more detail in the remainder of this section.

Direction

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

Sense Addressed / Modality Used

For input devices, this attribute describes the modalities used to operate an input device—manual (e.g., mouse or stylus), visual (e.g., eye-tracking input), acoustic (e.g., 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

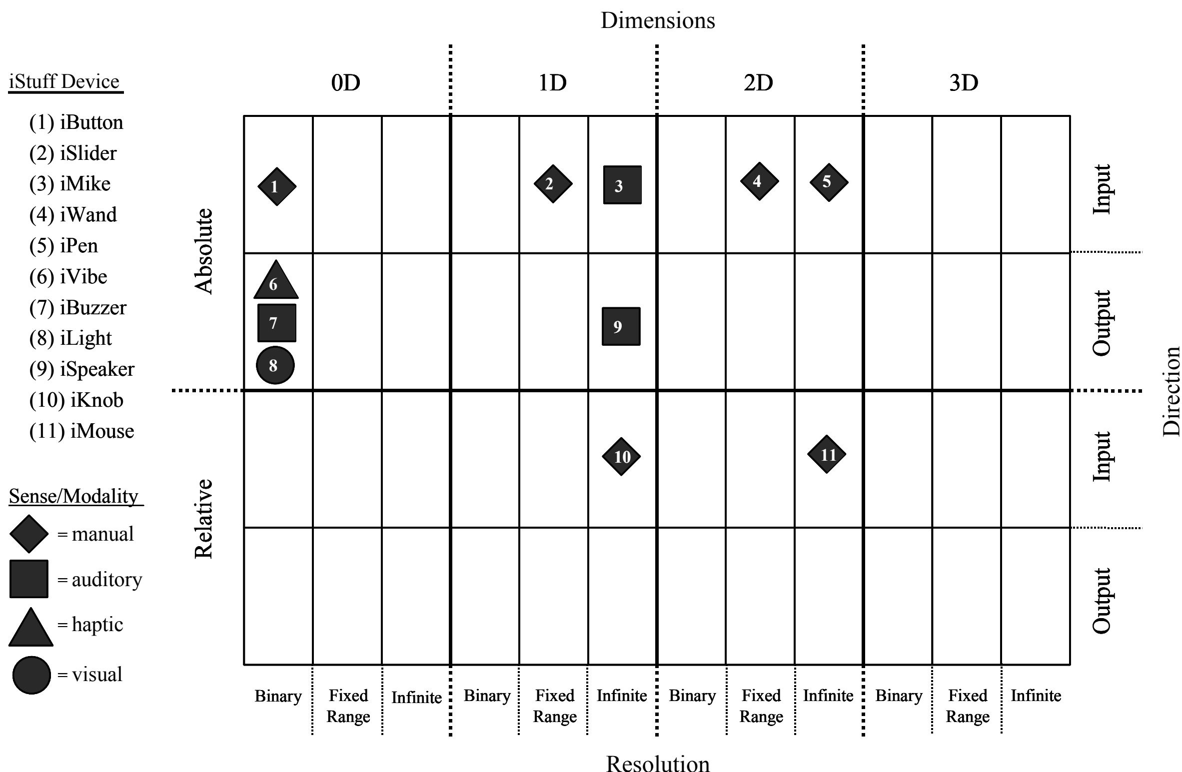

Figure 3 classifies the iStuff devices we have implemented thus far. Inspection of the diagram yields directions for future work—for instance, developing more non-manual input devices and higher-resolution output components.

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, resolution means 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, resolution describes whether a binary value (presence/absence of feedback) or a range of values can be 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.

Other Attributes of Ubicomp Interaction Devices

In addition to the five aforementioned dimensions, we have found "mount-time" and "directionality" to be additional important traits of ubicomp interaction devices.

Mount-time refers to the effort necessary to use an interaction device. Camera-based gesture recognition has no mount-time because the user does not need to engage any special equipment besides their hands. A mouse has a very small mount-time since it must be grasped, and a glove requires a much higher mount-time since it must be put on. High mount-time is not restricted to input devices—a head-mounted stereo display would be an example of a cumbersome output component.

Directionality, or scope, measures whether a device is targeted to one, many, or all of the users in a room. For instance, an earbud-type speaker targets its output to one user, while a large wall-mounted speaker's output is heard by all occupants inside a room. Input can also exhibit directionality—a keyboard used to authenticate to a system is a one-user device, while camera-based face recognition could capture multiple people simultaneously.

Since we are only beginning to explore the ramifications of these two traits, we did not feel they were developed enough to include in our design space at this time.

EXAMPLES OF USE

We tested the effectiveness of our toolkit by making it available to other researchers in the iRoom. In addition to the intended purpose of quickly combining components to prototype devices, developers and researchers in the room also used iStuff to explore various aspects of physical interaction. This included experimenting with physical form factors, as well as augmenting, and in some cases even replacing, application GUIs.

Encapsulating Events

The iROS infrastructure itself provides several services to the room. One of these services, known as iCrafter [16], 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—this button can be configured in approximately 30 seconds using a web-based patch panel, and requires no further coding.

Meeting Capture Software

iStuff was used to add functionality to another research project in the iRoom. One of the developers in the room had been working on a meeting capture program. During user studies, participants expressed a desire to discreetly 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.

iDog

A developer incorporated an iButton 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. The iDog is an important example because it was created in an attempt to inspire applications—inspiring 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.

The iWall supports multiple users and cursors by associating each cursor with a unique cursor ID. The iWall expects a cursor event that contains a field that specifies this cursor's 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 coexist on a single display.

Light Switch Toggle Button

The PatchPanel can be used to create state machines. A very simple state machine is demonstrated with the light switch toggle. The lights in the iRoom can be turned on by sending events to an X10 controller. iStuff can be used to create a physical light button for the iRoom that maintains as state whether the lights are on or off. The physical state is represented by whether the button is mapped to the event StateOnEvent or StateOffEvent. The iButton event is initialized in the PatchPanel to map to a StateOnEvent. The latter is configured to map both to a LightsOnEvent, and to an event to remap future events from this iButton to StateOffEvents. The StateOffEvent is mapped to a LightsOffEvent, and to an event to remap the iButton event to a StateOnEvent. When the iButton is pressed, the lights turn on, and the toggle state is modified to turn the lights off on the next press. In this example, the current button state is visualized simply by the room light itself.

Composite Event Translation

Composite events can be created using state in the PatchPanel to establish intermediate stages using event translations. This idea is demonstrated in Figure 3. For example, an application would like to be notified when both iButton1 and iButton2 have been pressed in any arbitrary temporal order. Each button can be configured in the PatchPanel to establish a connection between stages as well as an event to test the "circuit". The composite event can then be translated to an event that reinitializes the intermediate stages.

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 gained using iStuff.

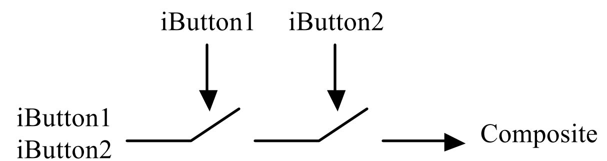

Figure 3. Composite Events via the PatchPanel

A Toolkit Fosters Exploration

iStuff has provided ourselves and others with a set of physical, post-desktop user interface components that can be integrated into research interface prototypes without exceeding the "threshold of indignation." This has led to more adventurous, innovative, and outside-the-box user interfaces in the research projects created by our group and visiting project students than before. By providing a playground to effortlessly gather experience with non-standard interface ideas, the toolkit has also led us to think about possibilities and challenges of distributed collocated interfaces and interactive rooms in a more profound and realistic way, as exemplified by the following items.

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 [11]. Under normal conditions, 5–6 users can operate PointRight simultaneously without noticeable delays, which is sufficient for experimentation. However, simultaneously downloading a movie to a laptop will slow the entire wireless 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. We are now exploring the design of new feedback techniques and modalities to provide the user with a mental model of the action that is both comprehensible and consistent with what they may observe in the room.

Streaming I/O

The Event Heap is not an ideal channel for streaming data. However, the iStuff model includes streaming I/O devices. Streaming I/O can be integrated into iStuff in two different ways. First, the iStuff proxy can process the streaming I/O into higher level abstractions similar to the iMike example above. Secondly, the Event Heap can be used as a control channel for an external stream. For example, a video camera can be controlled by start and stop events, and the came-

ra proxy can return an event with a pointer (i.e. a URL) to the streaming data.

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 prototyping physical user interfaces, including incorporating iStuff into an HCI design course or a Mechanical Engineering design course. We will also submit the Intermediary and PatchPanel 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 move information and control around 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, evolving our taxonomy into a design space. Lastly, we will continue to explore novel interaction techniques in our interactive workspace, using our iStuff toolkit.

CONCLUSIONS

In summary, the iStuff toolkit has proved to be a flexible prototyping platform for post-desktop ubiquitous computing interaction. This paper described a number of iStuff components and their application. An important aspect of the iStuff framework is the flexibility its 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'd like to give special thanks to Joyce Ho for her work on the iMike, Ya'ir Aizenman for his work on iWall and general iStuff improvements, Michael Champlin for his work on iStuff hardware, Jeff Raymakers for his work on the Event Heap based PointRight, and Robert Brydon for his work on iWall, iWands, the Event Heap based PointRight, and for feedback on revisions of this paper.

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

1. Abowd, G., Mynatt, E., and Rodden, T. The Human Experience. *IEEE Pervasive Computing Magazine,* 1(1), January–March 2002.
2. Beaudouin-Lafon, M. Instrumental Interaction: An Interaction Model for Designing Post-WIMP User Interfaces. *Proc. CHI 2000*, 446–453.
3. Bleser, T. and Sibert, J. Toto: A Tool for Selecting Interaction Techniques. *Proc. UIST 1990*, 135–142.
4. Buxton, W. Lexical and Pragmatic Considerations of Input Structures. *Computer Graphics*, 17(1), 31–37. 1983.
5. Card, S., Mackinlay, J., and Robertson, G. The Design Space of Input Devices. *Proc. CHI 1990*, 117–124.
6. Fischer, M., Stone, M., Liston, K., Kunz, J., Singhal, V. Multi-stakeholder Collaboration: The CIFE iRoom. *Proc. CIB W78 Conference 2002: Distributing Knowledge in Building*, 6–13.
7. Foley, J. D., Wallace, V. L., Chan, P. The Human Factors of Computer Graphics Interaction Techniques. *IEEE Comput. Gr. Appl.* 4(11), 13–48. 1984.
8. Greenberg, S. and Fitchett, C. Phidgets: Easy Development of Physical Interfaces Through Physical Widgets. *Proc. UIST 2001*, 209–218.
9. Ishii, H. and Ullmer, B. Tangible Bits: Towards Seamless Interfaces Between People, Bits and Atoms. *Proc. CHI 1997,* 234–241.
10. Johanson, B. and Fox, A. The Event Heap: A Coordination Infrastructure for Interactive Workspaces. *Proceedings of the 4th IEEE Workshop on Mobile Computer Systems and Applications (WMCSA-2002),* Callicoon, New York, June 2002.
11. Johanson, B., Fox, A., and Winograd, T. The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms. *IEEE Pervasive Computing Magazine*, 1(2), April–June 2002.
12. Johanson, B., Hutchins, G., Stone, M., and Winograd, T. PointRight: Experience with Flexible Input Redirection in Interactive Workspaces. *Proc. UIST 2002* (to appear).
13. Myers, B. A New Model for Handling Input. *ACM Trans on Info. Sys.*, 8(3), 289–320. 1990.
14. Myers, B. Kosbie, D. Reusable Hierarchical Command Objects. *CHI 1996*.
15. Olsen, D., Jefferies, S., Nielsen, T., Moyes, W., and Fredrickson, P. Cross-modal interaction using XWeb. *Proc. UIST 2000*, 191–200.
16. Ponnekanti, S., Lee, B., Fox, A., Hanrahan, P., and Winograd, T. iCrafter: A Service Framework for Ubiquitous Computing Environments. *Proceedings of Ubiquitous Computing Conference (UBICOMP) 2001*.
17. Salber, D., Dey, A., and Abowd, G. The Context Toolkit: Aiding the Development of Context-Enabled Applications. *Proc. CHI 1999,* 434–441.
18. Taylor, R., et al. A Component- and Message-Based Architectural Style for GUI Software. *IEEE Transactions on Software Engineering*, June 1996.
19. Weiser, M. The Computer for the 21st Century. *Scientific American,* 265(3), September 1991, 94–104.
20. "IBM WebSphere Voice Server: An IBM White Paper," IBM, October 2001.