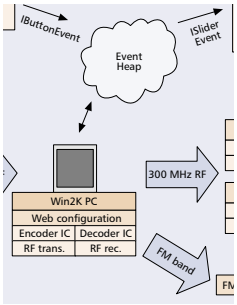
STANFORD INTERACTIVE WORKSPACES: A FRAMEWORK FOR PHYSICAL AND GRAPHICAL USER INTERFACE PROTOTYPING

JAN BORCHERS, MEREDITH RINGEL, JOSHUA TYLER, AND ARMANDO FOX
STANFORD UNIVERSITY



Most smart homes are created evolutionarily. This incremental addition of technology requires a highly flexible infrastructure to accommodate both future extensions and legacy systems without requiring extensive rewiring of hardware or reconfiguration on the software level.

OVERVIEW

Most smart homes are created evolutionarily by adding more and more technologies to an existing home, rather than being developed on a single occasion by building a new home from scratch. This incremental addition of technology requires a highly flexible infrastructure to accommodate both future extensions and legacy systems without requiring extensive rewiring of hardware or extensive reconfiguration on the software level. Stanford's iStuff (Interactive Stuff) provides an example of a hardware interface abstraction technique that enables quick customization and reconfiguration of Smart Home solutions. iStuff gains its power from its combination with the Stanford Interactive Room Operating System (iROS), which creates a flexible and robust software framework that allows custom and legacy applications to communicate with each other and with user interface devices in a dynamically configurable way.

The Stanford Interactive Room (iRoom,Fig. 1), while not a residential environment, has many characteristics of a smart home: a wide array of advanced user interface technologies, abundant computation power, and infrastructure with which to coordinate the use of these resources (for more information on the iRoom or the Interactive Workspaces project, please visit http://iwork.stanford.edu).As a result, many aspects of the iRoom environment have strong implications for, and can be intuitively translated to, smart homes. In particular, the rapid and fluid development of physical user interfaces using iStuff and the iROS, which has been demonstrated in the iRoom, is an equally powerful concept for designing and living in smart homes.

Before focusing on the details of iStuff, we describe the software infrastructure on which it is based and the considerations that went into designing this infrastructure.

iROS: APPLICATION COORDINATION IN UBIQUITOUS COMPUTING ENVIRONMENTS

SOFTWARE REQUIREMENTS FOR RAPID INTEGRATION AND EVOLUTION

The ability to continually integrate new technologies and handle failures in a noncatastrophic manner is be essential to smart homes and related ubiquitous computing environments. Our experience working in the Stanford iRoom enables us to identify four important requirements for a software infrastructure in a ubiquitous computing environment.

Heterogeneity: The software infrastructure must accommodate a tremendous variety of devices with widely ranging capabilities. This implies that it should be lightweight and make few assumptions about client devices so that the effort to "port" any necessary software components to new devices will be small.

Robustness: The software system as a whole must be robust against transient or partial failures of particular components. Failures should not cascade, and failure or unexpected behavior of one component should not be able to infect the rest of the working system.

Evolvability: The application program interface (API) provided must be sufficiently flexible to maintain forward and backward compatibility as technology evolves. For example, it should be possible to integrate a new type of pointing device that provides higher resolution or additional features not found in older devices, without breaking compatibility with those older devices or existing applications.

Compatibility: It should be easy to leverage legacy applications and technologies as building blocks. For example, Web technologies have been used for user interface (UI) prototyping, accessing remote applications, and bringing rich content to small devices; desktop productivity

1070-9916/02/$17.00 © 2002 IEEE

applications such as Microsoft PowerPoint™ contain many elements of a "rich content display server;" and so on. Furthermore, since technology in smart spaces tends to accrete over time, today's new hardware and software will rapidly become tomorrow's legacy hardware and software, so this problem will not go away.

Our prototype meta-operating system, iROS (Interactive Room Operating System), meets the above criteria. We call it a meta-OS since it consists entirely of user-level code running on unmodified commodity operating systems, connecting the various iRoom entities into a "system of systems." We discuss the main principles of iROS here to give the reader an understanding of how it facilitates building new behaviors using iStuff.

iROS and Application Coordination

We will frame our discussion in the context of the Stanford iRoom, a prototype environment we constructed that we believe is representative of an important class of ubiquitous computing installations. The iRoom is intended to be a dedicated technology-enhanced space where people come together for collaborative problem solving (meetings, design reviews, brainstorming, etc.), and applications we prototyped and deployed were driven by such scenarios.

The basis of iROS is *application coordination*. In the original formulation of Gelernter and Carriero [1], coordination languages express the interaction between autonomous processes, and computation languages express how calculations of those processes proceed. For example, procedure calls are a special case in which the caller process suspends itself pending a response from the callee. Gelertner and Carriero argue that computation and coordination are orthogonal and that there are benefits to expressing coordination in a separate general-purpose coordination language; our problem constraint of integrating existing diverse components across heterogeneous platforms leads directly to separating computation (the existing applications themselves) from coordination (how their behaviors can be linked).

In iROS, the coordination layer is called the *Event Heap* [2].The name was chosen to reflect that its functionality could be viewed as analogous to the traditional event queue in single-computer operating systems. The Event Heap is an enhanced version of a tuplespace, one of the general-purpose coordination languages identified by Gelernter and Carriero. A tuple is a collection of ordered fields; a tuplespace is a "blackboard" visible to all participants in a particular scope (in our case, all software entities in the iRoom), in which any entity can post a tuple and any entity can retrieve or subscribe for notification of new tuples matching a wildcard-based matching template. We have identified important advantages of this coordination approach over using rendezvous and RMI (as Jini does) or simple client-server techniques (as has been done using HTTP, Tcl/Tk [3], and other approaches); these advantages include improved robustness due to decoupling of communicating entities, rapid integration of new platforms due to the extremely lightweight client API (we support all major programming languages, including HTML, for posting and retrieving tuples), and the ability to accommodate legacy applications (simple "hooks" written in Visual Basic or Java can be used to connect existing productivity, Web, and desktop applications to the iRoom).The only criterion for making a new device or application "iRoom-aware" is its ability to post and/or subscribe to tuples in the Event Heap; since we can create Web pages that do this, any device that enters the room running a Web browser is already minimally iRoom-aware.

Figure 1. *The Stanford iRoom contains a wireless GyroMouse and keyboard (visible on the table), three touch-sensitive SmartBOARDs and one non-touch-sensitive tabletop display, and a custom-built OpenGL hi-res graphic mural. The room is networked using IEEE 802.11b wireless Ethernet. Except for the hi-res mural and the tabletop, all hardware is off-the-shelf, all operating systems are unmodified Windows (various flavors) or Linux, and all software we have written is user-level.*

On-the-Fly User Interface Generation in iROS

The Event Heap is the core of iROS, but we have also built other iROS services that provide higher-level functionality. Most notably, the Interface Crafter (iCrafter) framework [4] can generate UIs dynamically for virtually any iRoom entity and on virtually any iRoom-aware device. Although it extends previous work in several important ways, including integration of service discovery with robustness and the ability to create UIs ranging from fully custom to fully automatic, its main role in the present scenarios is to serve as an abstraction layer between devices and UIs. Briefly, iCrafter is used as follows:

¶Controllable entities beacon their presence by depositing self-expiring advertisements in the Event Heap. These advertisements contain a description of the service's controllable behaviors (i.e., methods and their parameters) expressed in SDL, a simple XML-based markup language we developed.

¶A device capable of displaying a UI (Web browser, handheld, etc.) can make a request for

IMPLEMENTATION DETAILS

Transmitting devices (buttons, sliders) contain a Ming TX-99 V3.0 300 MHz FM radio frequency (RF) transmitter and a Holtek HT-640 encoder to send 8 bits of data to a receiver board, which contains a Ming RE-99 V3.0 RF receiver and a Holtek HT-648L decoder. The receiver board sends its data to a PC using either the parallel or USB port, and a listener program running on the PC then posts an appropriate tuple (based on the ID received) to the iRoom's Event Heap. Receiving devices (buzzers, LEDs) work in the opposite manner: a listener program receives an event intended for the iStuff and sends the target device ID through either the parallel or USB port to an RF transmitter. This data is then received wirelessly by an RF receiver in the device, resulting in the desired behavior. The iSpeaker has a different architecture, since the RF technology we employed is not sufficient for handling streaming media. Instead, a listener program on the PC waits for speaker-targeted events, and in response streams sound files over an FM transmitter, which our iSpeaker (a small portable FM radio) then broadcasts.

the UI of a specific service, or can query the iCrafter's *Interface Manager* (which tracks all advertisements) to request a list of available services. This initial request is made via whatever request-response technology is available on the client: visiting a well-known dynamically generated Web page is one possibility.

¶The desired UI is created by feeding the SDL contained in a recent service advertisement to one or more interface generators. These may be local or remote (i.e., downloaded on demand over the Web), and may be specialized per service and/or per device. The Interface Manager determines the policy for selecting a generator. Part of this process includes integrating contextual information from a separate context database relevant to each workspace, making a "static" UI description portable across installations. For example, in a workspace such as ours with three large wall-mounted displays, it is preferable for a UI to refer to these as "Left, Center, Right" than to use generic names such as "screen0, screen1, screen2" (Fig. 2).

Note that in the last step the client device and service do not need to establish a direct connection (client-server style).This makes each robust to the failure of the other. They do not even need to be able to name each other using lower-level names such as network addresses because the tuple-matching mechanism can be based on application-level names or attributes of the service ("retrieve advertisements for all devices of type LightSwitch").The same service can be controlled from a variety of different devices without knowing in advance what types of devices are involved, since the same SDL description can be processed into quite different UIs by different interface generators.

The ability to insulate the service and UI from each other in these ways has been critical to the rapid prototyping of new UI behaviors. iStuff builds on this ability, usin this indirection to enable rapid prototyping of *physical* UIs as well.

iSTUFF: PHYSICAL DEVICES FOR UBIQUITOUS COMPUTING

iSTUFF MOTIVATION AND DEFINITION

iStuff is a toolbox of wireless platform-independent physical UI components designed to leverage the iROS infrastructure (which allows our custom-designed physical devices to send and receive arbitrary commands to and from other devices, machines, and applications in the iRoom).The capability to connect a physical actuator to a software service on the fly has appeal to users of a ubiquitous computing environment such as a smart home. Residents would have the ability to flexibly set up sensors and actuators in their home, and designers of such homes would be able to prototype and test various configurations of their technology before final installation.

There are several characteristics that are crucial for our iStuff:

- Completely autonomous packaging, wireless connection to the rest of the room, and battery-powered operation
- Seamless integration of the devices with iROS as an existing, available, cross-platform ubiqui-

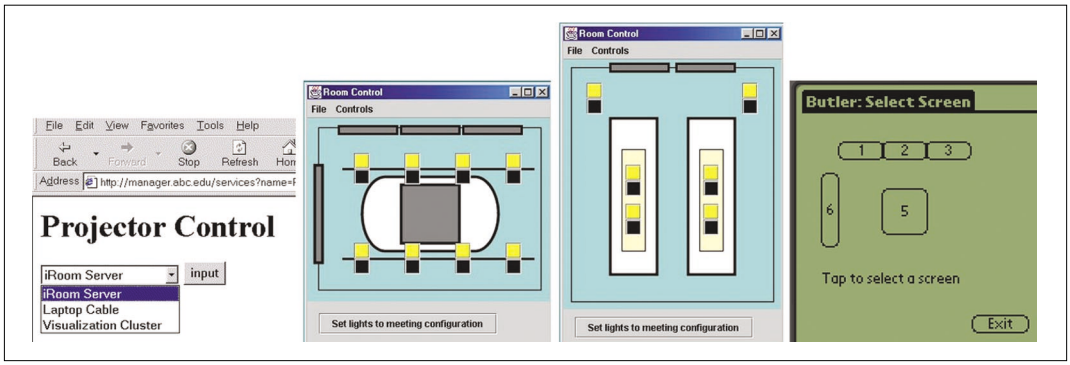


Figure 2. *Screen/projector control UIs customized for various devices and incorporating installation-specific information from the context database: (a) A fragment of a projector HTML interface that can be rendered by any Web browser. This UI was generated by a projector-specific HTML generator. The symbolic names such as iRoom Server are stored in the context database and appear in the SDL markup as "machine0," "machine1," and so on. (b) Room control applet for the same room, generated by a Java Swing-UI generator. The geometry information for drawing the widgets comes from the context database, so the generator itself is not installation-specific. Users can drag and drop Web pages onto the screen widgets to cause those documents to appear on the corresponding room screens. (c) The same UI using different geometry information (for a different room) from the context database. (d) A Palm UI rendered in MoDAL [8] that lacks the drag-and-drop feature.*

tous computing environment to let devices, machines, and services talk to each other and pass information and control around
- Easy configuration of mappings between devices and their application functionality, by customizing application source code, or even just updating function mappings using a Web interface
- Simple and affordable circuitry

Various other research projects have looked at physical devices in the past; Ishii's Tangible Bits project [6] introduced the notion of bridging the world between bits and atoms in UIs, and more recently Greenberg's Phidgets [7] represent an advanced and novel project in physical widgets. Phidgets, however, are designed for use in isolation with a single computer, are tethered, and do not work across multiple platforms.

Device Classification and Implementation

The range of potentially useful UI components is almost unlimited, and the really useful devices to go in a standard toolbox will only be identified over time. Ideas for such devices can be categorized according to whether they are input or output devices, and the amount of information they handle, as in the following examples:
- One-bit input devices, such as pushbuttons and toggle buttons, or binary sensors such as light gates
- Multiple-bit discrete input devices, such as rotary switches or digital joysticks as well as packaged complex readers that deliver identification data as a result (e.g., barcode readers or fingerprint scanners)
- Near-continuous input devices, such as sliders, potentiometers, analog joysticks or trackballs, and various sensors (light, heat, force, location, motion, acceleration, etc.)
- Streaming input devices, such as microphones and small cameras
- One-bit output devices, such as a control or status light, beepers/buzzers, solenoids, and power switchers
- Multiple-bit discrete output devices, such as LED arrays or alphanumerical LCD displays
- Near-continuous output devices, such as servos, motors, dials, and dimmers
- Streaming output devices, such as speakers and small screens

Thus far, students in our laboratory have designed and built five types of prototype iStuff devices spanning four of the above categories: iButtons, iSliders, iBuzzers, iLEDs, and iSpeakers (Fig. 3).While our hardware designs have proven surprisingly powerful and proofs of concept, they are simple enough to be reproduced easily (see box).

We have developed several successful setups using iStuff in the Stanford iRoom:

¶New users coming into our iRoom are not familiar with the environment, and need an "entry point" to learn about the room and its features. Using our iStuff configuration Web interface, we programmed one iButton to
- Send events that turn on all the lights in the room
- Switch on all SMARTBoards (large touch-sensitive displays) and our interactive table display

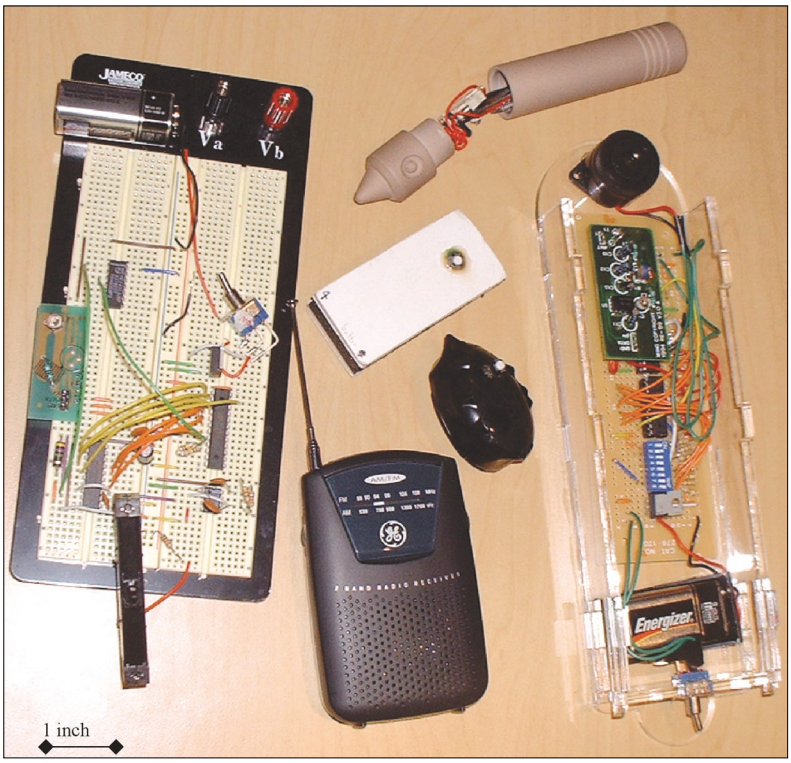


Figure 3. *The various types of iStuff created so far: buttons, potentiometers, speakers, and buzzers.*

- Bring up a Web-based introduction to the room on one SMARTBoard
- Show an overview of document directories for the various user groups on a second SMARTBoard
- Open up our brainstorming and sketching application on the third SMARTBoard

It is worth nothing that setting up this iRoom Starter took less than 15 minutes of configuration using the Web interface.

¶SMARTBoards provide only a rather inconvenient way to issue right clicks when using the touch-sensitive board for input — users have to press a right-click "mode key" on the tray in front of the board to have their next touch at the board be interpreted as a right click. To study whether having the right-click modifier closer to the actual input location at the board would make this interaction more fluid, we built a specialized iButton that was shaped to fit inside a hollow pen prototype made from RenShape plastic by a product design student in our model shop. When the button is pressed it sends an event to the Event Heap that is then received by a listener application running on the computer associated with the SMARTBoard. The listener then tells the SMARTBoard driver to interpret the next tap as a right click. Users can now simply press the button on the pen and then tap to issue a right click.

¶We found our iSlider could conveniently control the paddles for our multiscreen version of the classic video game Pong, described below.

¶The iSpeaker has been extended to provide verbal feedback for user actions (e.g., "Pong game started") by means of a text-to-speech program — applications simply send a SpeakText event to the iSpeaker containing the ASCII text to be spoken.

When residents acquire a new device, or wish to reconfigure existing devices, they can simply use a utility such as our "Patch Panel" to map the event type sent by the new device to the event type expected by the target application.

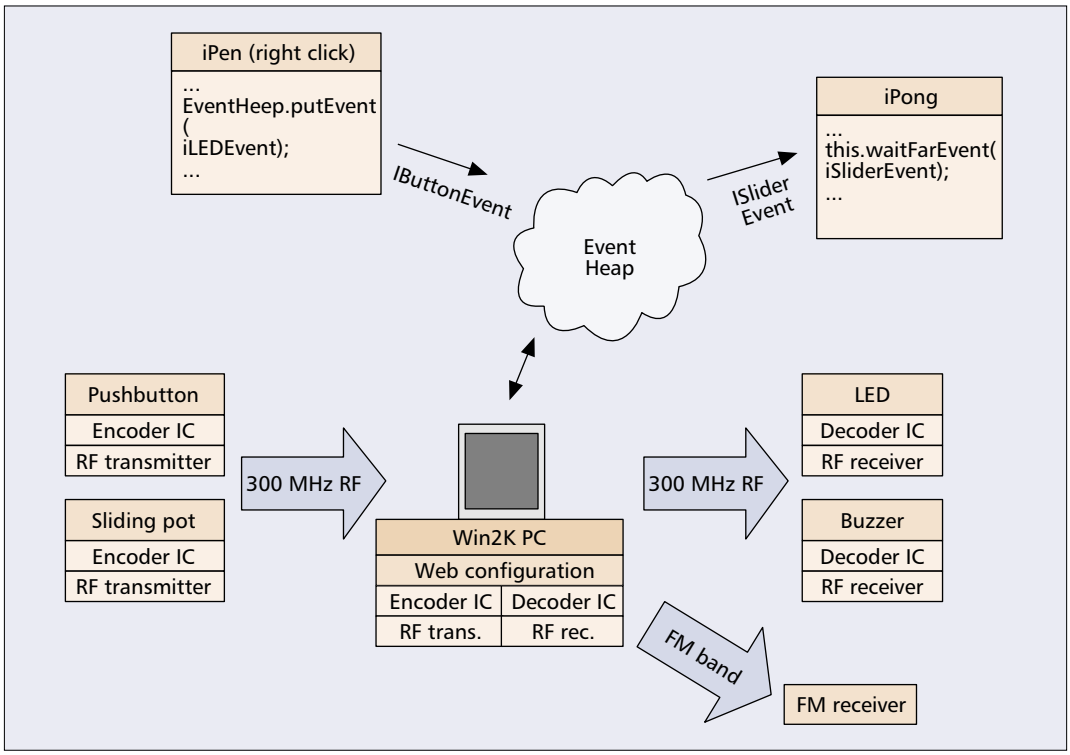


Figure 4. *The overall system architecture for iStuff. In darker boxes are the actual physical devices, and in the lighter ones are a couple of examples of applications using iStuff and the iROS Event Heap. An iStuff server translates the wireless device transmissions into software events, which can be consumed by interested applications. For example, when the potentiometer of the iSlider is moved, it sends a radio signal, which is received by the server and turned into a SliderEvent. The event is posted to the Event Heap, and subsequently received by the iPong application, which is listening for SliderEvents.*

¶We are experimenkting with our iLEDs and iBuzzers to provide feedback about the status of devices in the room.

As discussed before, the Event Heap is a core component of the iROS that makes it possible to decouple the sender and receiver from the contents of the message itself (Fig. 4).This architecture allows great flexibility for the prototyping of interfaces; for instance, an application can be controlled by either a traditional mouse, a graphical slider widget, or an iSlider as long as each of those devices sends the event type (perhaps an event containing a new Y-coordinate) for which the application is listening.

In our iRoom we have demonstrated the utility of the combination of Event Heap software with iStuff hardware by developing iPong, a multimachine version of the video game where players control the vertical position of virtual paddles to make contact with a virtual bouncing ball. The game was written to listen for Paddle Events, which contain information about the new position of the target paddle. Any input method that can generate a Paddle Event can control the paddle position. We have mapped the standard mouse, touch panel input, and an iSlider (a sliding-potentiometer iStuff widget) to drive the paddle. To the application, the physical source of the events is irrelevant. Thus, we have decoupled the link between hardware and software components in a physical UI.

Our iButtons are already reconfigurable dynamically via a Web interface that lets users enter arbitrary events to send when a specific button is pressed. We intend to provide this flexible interactive mechanism for mapping applications and events for all iStuff, using the on-the-fly service discovery tools of iROS (described earlier).The result will be a general virtual Patch Panel that allows even end users to map events to services and map conversions between related event types. Thus, iStuff makers can send and receive their own types of events (e.g., button events or slider events) without concern for the exact names of events desired by end-user applications, and application developers can send and receive their own types of events (e.g., Paddle Event) without prior knowledge of every possible type of device the user might choose to interface with their application.

The iStuff/Event Heap combination has direct applications to the Smart Home that incrementally acquires new technologies. When residents acquire a new device or wish to reconfigure existing devices, they can simply use a utility such as our Patch Panel to map the event type sent by the new device to the event type expected by the target application.

Smart Home Applications

While our iStuff was originally designed with our iRoom (a space used for meetings and brainstorming/design sessions) in mind, our technology and infrastructure could be useful in a dmart home environment. In particular, the ability to create *task-oriented user interfaces* — interfaces reflecting the user's task as opposed to the technical features of an appliance — makes iStuff par-

ticularly compelling for Smart Home applications.

Dynamic, task-based remote controls: Currently, when a user wants to watch a movie on a DVD, they need several remote controls: one to control the DVD player, another to control their home's surround-sound system, and a third to control the television set (and then the user has to get up to dim the lights!).Today's remotes are device-based, but because the Event Heap architecture allows for the decoupling of devices from messages we are able to use iStuff to construct task-based remote controls. By gathering appropriate iStuff components and using the Patch Panel application to ensure that the appropriate iStuff events are converted to the events appropriate to the target devices (DVD player, speakers, TV set, lights), the user can construct a task-oriented controller-one device that controls all appliances relevant to viewing a DVD movie, regardless of their physical connectivity. iCrafter could be used in an analogous manner to dynamically create GUI controllers for household appliances, thus transforming a PDA into a task-based universal remote control.

Monitoring house state: A user is on her way out the door of her smart home, about to head off to work. The display near her door shows her the status of several devices in her home that have been instrumented with iSensors: did she leave the stove on? The lights in her bedroom? Is the thermostat too high? Is the burglar alarm on?

Setting house state: A user can create an iButton or similar device to set the house's "state" as she leaves for work every day, and mount this button by her door. She might configure it to lower her thermostat, switch off all lights, and activate her security system, for example. This type of button is analogous to our Start iRoom button mentioned earlier.

Smart home design: Architects and interior designers could use iStuff to fine-tune the placement of controls, speakers, and other interactive elements of a Smart Home. Researchers and technology developers could use iStuff to quickly prototype and test their products before putting them on the market for addition to smart homes.

Discussion and Summary

Technology advancements have made much of the original vision of ubiquitous computing feasible. A software framework, however, that integrates those heterogeneous technologies in a dynamic, robust, and legacy-aware fashion and provides a seamless user experience has been missing. We have created the Stanford iRoom, a physical prototypical space for ubiquitous computing scenarios that has been in constant use for almost two years now, to address this need. iROS, our iRoom Operating System, runs as a meta-OS to coordinate the various applications in the room. iROS is based on a tuplespace model, which leads to the aforementioned desired characteristics. Its failure robustness has been better than average for both induced and real faults. Its ability to leverage and extend existing applications has been critical for rapid prototyping in our research.

The iStuff project builds on iROS, and tackles the problem that customizing or prototyping physical user interfaces for ubiquitous computing scenarios (e.g., smart homes) is still a very arduous process. It offers a toolbox of wireless physical user interface components that can be combined to quickly create nonstandard user interfaces for experimentation. The iROS infrastructure has proven invaluable in making the software integration of these custom devices very straightforward. The flexibility of the technology we developed for Stanford's iRoom has potential benefits in a smart home scenario, for example, by enabling users to quickly create a customized task-based interface to a system in their home.

In all, we hope that our approach to building a software and hardware framework for ubiquitous computing environments, and the various building blocks we have implemented and deployed, are general and useful enough so that others will find them of value. For more information on our Stanford Interactive Workspaces project, access iStuff documentation, or download iROS software, please visit our project homepage at http://iwork.stanford.edu/.

A user can create an iButton or similar device to set the house's "state" as she leaves for work everyday, and mount this button by her door. She might configure it to lower her thermostat, switch off all lights, and activate her security system, for example.

Acknowledgments

The authors would like to thank Maureen Stone, Michael Champlin, Hans Anderson and Jeff Raymakers for their contributions to this work, as well as the Wallenberg Foundation (http://www.wgln.org) for its financial support.

References

[1] D. Gelernter and N. Carriero, "Coordination Languages and their Significance," *Commun. ACM*, vol. 32, no. 2, Feb. 1992.
[2] B. Johanson and A. Fox, "The Event Heap: A Coordination Infrastructure for Interactive Workspaces," to appear in *Proc. WMCSA 2002*, Callicoon, NY, June 2002.
[3] T. D. Hodes et al., "Composable Ad-Hoc Mobile Services for Universal Interaction," *Proc. ACM MobiCom '97*, Budapest, Hungary, Sept. 1997.
[4] S. R. Ponnekanti *et al.*, "ICrafter: A Service Framework for Ubiquitous Computing Environments," *Proc. UbiComp. '01*, Atlanta, GA.
[5] T. Lehman *et al.*, MoDAL (Mobile Document Application Language); http://www.almaden.ibm.com/cs/TSpaces/MoDAL
[6] H. Ishii and B. Ullmer, "Tangible bits: Towards seamless interfaces between people, bits and atoms," *Proc. CHI '97*, Atlanta, GA, Mar. 22–27, 1997, pp. 234–41.
[7] S. Greenberg and C. Fitchett, "Phidgets: Easy Development of Physical Interfaces Through Physical Widgets," *Proc. UIST 2001*, Orlando, FL, Nov. 11–14, 2001, pp. 209–18.

Additional Reading

[1] W. K. Edwards and R. E. Grinter, "At Home with Ubiquitous Computing: Seven Challenges," *Proc. UbiComp '01*, Atlanta, GA, pp. 256–72.
[2] E. Kiciman and A. Fox, "Using Dynamic Mediation to Integrate COTS Entities in a Ubiquitous Computing Environment," *Proc. HUC2K*, LNCS, Springer Verlag.

Biographies

JAN BORCHERS (borchers@cs.stanford.edu) is an acting assistant professor of computer science at Stanford University. He works on human-computer interaction in the Stanford Interactivity Lab, where he studies post-desktop user interfaces, HCI design patterns, and new interaction metaphors for music and other types of multimedia. He holds a Ph.D. in computer science from Darmstadt University, and has been known to turn his research into public interactive exhibits.

MEREDITH RINGEL (merrie@cs.stanford.edu) is a first-year Ph.D. student in computer science at Stanford, with a focus on human-computer Interaction.She received her B.S. in computer science from Brown University.

JOSHUA TYLER (jtyler@cs.stanford.edu) is a second-year Master's student in computer ccience at Stanford with a spe-

The iStuff project builds on iROS, and tackles the problem that customizing or prototyping physical user interfaces for ubiquitous computing scenarios is still a very arduous process.

cialization in human-computer interaction. He received a B.S. in computer science from Washington University.

ARMANDO FOX (fox@cs.stanford.edu) joined the Stanford faculty in January 1999.His research interests include the design of robust Internet-scale software infrastructure, particularly as it relates to the support of mobile and ubiquitous computing, and user interface issues related to mobile and ubiquitous computing. He received a B.S.E.E. from M.I.T., an M.S.E.E. from the University of Illinois, and a Ph.D. from UC Berkeley. He is a founder of ProxiNet, Inc. (now a division of PumaTech), which commercialized thin client mobile computing technology developed at UC Berkeley.