Integrating Information Appliances into an Interactive Workspace

Armando Fox, Brad Johanson, Pat Hanrahan, and Terry Winograd
Stanford University

We present a robust, infrastructure-centric, and platform-independent approach to integrating information appliances into the iRoom, our interactive workspace.

Most of today's computing environments—by design—support interaction between one person and one computer. The user sits at a workstation or laptop, or holds a personal digital assistant (PDA), focusing on a single device at a time—even with several devices around, linked and synchronized. Collaboration occurs over the network using e-mail, shared files, or in some cases explicitly designed groupware. In noncomputerized work settings, on the other hand, people interact in a rich environment that includes information from many sources—paper, whiteboards, computers, physical models, and so on. They can use these simultaneously and move among them flexibly and quickly. The few integrated multidevice computer environments existing today tend to be highly specialized and based on application-specific software.

The Interactive Workspaces Project at Stanford explores new possibilities for people to work together in technology-rich spaces with computing and interaction devices on many different scales. It includes faculty and students from the areas of graphics, human-computer interaction (HCI), networking, ubiquitous computing, and databases, and draws on previous work in all those areas. We design and experiment with multidevice, multiuser environments based on a new architecture that makes it easy to create and add new display and input devices, to move work of all kinds from one computing device to another, and to support and facilitate group interactions. In the same way that today's standard operating systems make it feasible to write single-workstation software that uses multiple devices and networked resources, we are constructing a higher level operating system for the world of ubiquitous computing. We combine research on infrastructure (ways of flexibly configuring and connecting devices, processes, and communication links) with research on HCI (ways of interacting with heterogeneous changing collections of devices with multiple modalities).

The Interactive Room (iRoom) infrastructure described in this article is brand new: the physical plant for the room was constructed during the summer of 1999, and the room became operational for the first time in late September 1999. We report here on our very early work on our strategy for integrating PDAs into this infrastructure.

Application target areas

We chose to focus our current work on an augmented dedicated space (a meeting room, rather than an individual's office or home, or a tele-connected set of spaces) and to concentrate on task-oriented work rather than entertainment, personal communication, or ambient information. In this section we describe some of our initial research goals in terms of specific applications we developed. These applications also serve as motivating examples for the programming mechanisms described in later sections.

The photo of the current iRoom configuration (Figure 1) illustrates the basic room hardware: three touch-sensitive SmartBoard displays; a bottom-projected table; a front-projected (non-input-responsive) screen; a variety of wireless mice, keyboards, and PDAs for interacting with the screens; and approximately eight PCs (not visible) providing computing, rendering, and display server capabilities.

Our interest in multimodal input forms the basis of our investigation of *human-centric interaction*,[1] in which contextual information provided by software observers is integrated to identify user intent based on multiple input sources and modalities. The focus remains on the

0272-1716/00/$10.00 © 2000 IEEE

1 The Interactive Workroom (iRoom).

person and the task (which will be integrated across multiple devices) rather than on device-bound applications. The architecture makes it easy for application designers to mix and match devices in a way that best enhances the interaction affordances for the user. This can include using large touchscreen devices, ordinary laptops and workstations, and PDAs.

For our initial explorations of this area, we wanted the ability to rapidly prototype new applications and new usage scenarios for existing applications, in particular those that would integrate PDAs into the application usage model. The rest of this article describes our specific strategies and our experience building prototypes to date. We applied our techniques in two areas. The first involves augmenting widely deployed legacy applications such as Web browsers and desktop productivity tools with collaborative behaviors and the ability to handle multimodal input. The second focuses on the use of PDAs as remote controllers for logical or physical entities in the iRoom.

Augmenting legacy applications. Rather than immediately starting to design new applications explicitly written to interact with PDAs, we focused on augmenting existing applications. For example, SmartPPT (Smart PowerPoint), a student project using the iRoom, seamlessly integrates the multiple displays available in the workspace to provide the speaker presenting PowerPoint slides with more creative space. It also supports the audience in using laptops, PDAs, and other handheld devices to participate in the discussion by allowing them to browse through the information presented, annotate it, and communicate with the speaker and the rest of the audience. Audience members can view an outline of the presentation along with thumbnails and details of individual slides; preview slides yet to be presented; review slides already presented; post questions to be integrated into the presentation; and store meta information about the presentation (such as e-mail addresses, URLs, and so forth). The presentation author can script the desired behaviors (for example, the on-screen slide layout) in advance with a separate authoring tool that integrates with PowerPoint or rely on some built-in default behaviors. SmartPPT was directly inspired by the StuPad application of the pioneering Classroom 2000 project,[2,3] but as we describe later, its infrastructure-centric approach has resulted in a lighter weight implementation.

Universal interactors. PDA-like devices can control physical devices or applications running in the room. Prior work[4] explored how applications or devices might export canonicalized descriptions of their interfaces, permitting on-the-fly creation of a user interface and its export to a handheld device. Our Java-based room display manager shows a list box with various potentially interesting URLs and a schematic of the screens in the room. Users can drag and drop the URLs on the desired screen to view the corresponding Web page on that display. During meetings, this provides a convenient way for participants to take advantage of all the display surfaces for displaying information, without requiring them to choreograph a presentation in advance. We also wrote two versions of an application for remotely controlling projectors and lights in the room, described in a later section.

A number of other projects are developing multidevice interactive workspaces, but with a different underlying software architecture. For example, the iLand environment built at GMD-IPSI in Darmstadt[5] physically resembles our interactive workspace. It includes an interactive electronic wall (DynaWall) based on three back-projected touchscreen displays, an interactive table (InteracTable) with a bottom-projected display, computer-enhanced chairs (CommChair) that incorporate docking for wireless use of laptops, and the Passage mechanism, which uses physical tokens to represent information objects.

However, iLand's software philosophy differs from our approach. They build software for use in the environment using their own tools, rather than incorporating preexisting interfaces and applications. Their Beach software platform is built in Smalltalk, based on an object-oriented framework called Coast for synchronizing multiple simultaneous access to objects. This enables them to develop sophisticated groupware envi-

The human-computer interaction focus of our research agenda leads us to an interesting notion of application portability: applications should be portable between devices with similar usage models.

ronments that depend on object synchronization, but it does not support standard Unix or Windows applications as regular components of the environment.

In contrast, we chose to leverage existing applications as much as possible. We wanted both to maximize the impact of our work and to minimize the resources expended in constructing the "scaffolding" needed to begin conducting interesting HCI and applications research in the iRoom.

Meta-goals

Beyond the specific functionality required to support PDA applications, some of our desiderata for PDA integration involve robustness and maintenance. We believe these to be of primary importance if others are to benefit from our work by adapting our infrastructure to their own projects.

Hardware and software diversity. We already support a diverse array of PDA devices and similar ubiquitous-computing gadgets, such as PC-controlled lighting via X10, a programmatic interface to a surround-sound system for the room, and so forth. To the extent possible, our programming infrastructure should shield application developers from having to deal directly with hardware and protocol heterogeneity.

The HCI focus of our research agenda leads us to an interesting notion of application portability: applications should be portable between devices with similar usage models. From a research standpoint, Windows CE palm-sized PCs and Palm Pilots might well be considered the "same" device type because even though their operating environments and programming models differ, their usage models are similar. In contrast, both sketchpad-sized Jupiter-class subnotebooks and the wall-sized SmartBoards run Microsoft Windows-based operating systems, but clearly their usage models differ markedly. We would like our programming environment to reflect this HCI-centric notion of portability.

Legacy application support. We want to interact with unmodified legacy applications (Web, productivity, computer-aided design, and other domain-specific tools) from PDAs. We emphasize "unmodified" because we don't in general have source code for the legacy applications. Even if we did, we would like other researchers to benefit from iRoom software without having to obtain specialized versions of these applications.

Leverage. Historically, PDA programming has proved awkward and difficult even for seasoned programmers of desktop applications, in part because of the limited programming environments and nontrivial resource constraints PDAs present when compared with their desktop counterparts. We would like to facilitate application prototyping for PDAs by greatly reducing the level of expertise required to implement simple behaviors. At the same time, we want to provide sufficient flexibility for advanced programmers to prototype and design sophisticated PDA user interfaces in a way that will permit reusing their efforts in the future.

To realize these goals, we have favored platform-neutral languages and development environments (for example, Java instead of C++). We also adapted widely deployed protocols and infrastructures in building new applications rather than using lower level specialized protocols, even at the expense of some efficiency (such as Web applications with Web-based GUIs instead of custom client-server applications with specialized GUIs). We extended these strategies to PDAs relatively painlessly.

No-futz functionality. Entering the iRoom and doing useful work should not require a user to be an expert in the organization of the software infrastructure. As much as possible, the software should be self-managing and self-repairing in the face of simple transient faults. Similarly, application programmers who want to write fairly simple applications, or add collaborative behaviors to existing applications, should not have to understand the entire software infrastructure.

Infrastructure-centric approach to ubiquitous computing

A large body of work goes by the names ubiquitous, pervasive, and mobile computing. Although many people use the terms loosely and often interchangeably, we find it useful to categorize each contribution according to the degree to which it is infrastructure-centric. By this we mean that a ubiquitous-computing device such as a PDA can make strong assumptions about its environment: the availability and quality of network connectivity, and the availability and quality of computation embedded in the infrastructure to assist PDAs in interacting with the rest of the environment and even with each other.

For example, consider a group of PDA-equipped users entering a room. They would like to share the notes each user has taken on her PDA. One solution is for the PDA software to assume very little about the infrastructure: each device must be able to locate or discover the presence of other devices, establish or participate in a small ad-hoc network for physical communication, and agree on distributed protocols for information exchange. This solution (and its formidable engineering challenges) may suit scenarios in which we cannot assume the availability of a single, centrally managed, resource-rich infrastructure. For example, if the users weren't all part

of a common user community, administrative or security concerns might prevent them from enjoying a common infrastructure.

A different solution assumes that the room into which the users enter already has a centralized wireless base station and logically centralized software components that facilitate communication and data interchange among devices. Clearly this approach removes much of the burden from individual devices, assuming the infrastructure assumption holds. We call this scenario the infrastructure-centric scenario.

We chose the infrastructure-centric scenario for several reasons:

1. We focus on using PDAs in a specific location, namely, the iRoom. It's reasonable to collocate network gateways and servers in the room and keep them highly available, since the room is fixed and centrally managed.
2. In the common case, iRoom users do in fact belong to a single community. However, we can provide at least some services to visitors, if the visitors are willing to install minimal software on their devices. (Even for visitors the infrastructure approach proves advantageous, since it minimizes the amount of functionality each visitor must install on her device in order to participate in the iRoom.)
3. We can leverage a large body of existing work that has successfully addressed specific pieces of the PDA problem with infrastructure-centric solutions. For example, information access solutions such as ProxiNet's ProxiWeb browser (http://www.proxinet.com) permit accessing most standard Web pages unmodified from a PDA, including graphics, forms, and so on. We would like to use this existing technology as building blocks for more sophisticated applications.

Note that most of the high-level functionality resides in infrastructure software. The infrastructure assumption is more about software than about hardware and raw network connectivity. A substantial literature exists on how infrastructure-software approaches can greatly simplify ubiquitous computing, from the seminal ParcTab project[6] to generalized infrastructure support for Web-like applications on thin or low-bandwidth clients.[7] Several research projects concentrate exclusively on the software frameworks needed to deploy such infrastructures.[8]

Information appliance programming approaches and examples

Our PDA arsenal currently includes

- A few standard Palm III devices
- Two Vadem Clio Windows CE devices with touch-sensitive screens, keyboards extendable (to create a subnotebook-like device) or hidden (to create a sketchpad-like device), and WaveLAN PC cards
- Three Cassiopeia E-10 Windows CE palm-sized PCs with a form factor approximately the same as the Palm devices, but higher resolution screens (320 × 240 versus 160 × 120) and support for 16-bit color (versus 2-bit grayscale)
- One HP Jornada 680 handheld PC—the "clamshell" model that unfolds into a keyboard and screen—equipped with a WaveLAN PC card
- A variety of notebooks and subnotebooks running Windows and Linux, equipped with WaveLAN PC cards

Assume the room already has a centralized wireless base station and logically centralized software components. Clearly this removes much of the burden from individual devices. We call this scenario the infrastructure-centric scenario.

All the Palm devices and the palm-sized Windows CE devices have the ProxiWeb browser and Waba Virtual Machine installed.

The iRoom is part of the Gates Computer Science Building IP network. In addition to Ethernet taps and serial cables connected to a point-to-point protocol (PPP) server, the iRoom provides wireless Ethernet access via WaveLAN (IEEE 802.11) base stations. The Cassiopeia E-10 and the Palm III use PPP over serial cables; all other devices, including the Clios and iRoom team members' laptops, support WaveLAN. All communication uses protocols based on TCP/IP.

The event heap

The primary software abstraction in the iRoom is the *event heap*. An event heap resembles a traditional GUI event queue, with a few important differences:

- Multiple entities can subscribe to a given event stream, enabling multicast-style groupware applications.
- Events auto-expire and are eventually garbage-collected from the heap if not consumed, making it unnecessary for event senders to verify whether any receivers are present.
- The event data structure is largely self-describing and extensible, so events can be subclassed without explicitly informing all entities about changes in a class hierarchy.

Later we motivate the choice of an event heap mechanism over explicit client-server couplings. Functionally, the event heap is a tuple-space-like mechanism by which entities in the iRoom communicate. (In fact, our

Multibrowsing lets users construct Web sites in which following links can cause the destination page to appear on any of the iRoom's screens, not just the screen displaying the page containing the link.

current implementation is based on TSpaces.[9]) Entities can post events to the event heap, query the event heap for the presence of events matching a template, or subscribe for notification when another entity posts events matching a template. We have devised naming schemes to identify the intended source, receiver classes, and other relevant event fields. For concreteness, see Table 1 for details of the event heap data structures, including required fields and currently supported event types.

We created event-heap procedure interfaces in Java and C/C++. Both call the underlying TSpaces code, which uses socket communications to invoke against a TSpaces server process running on a machine in the iRoom.

For the present discussion, what's relevant is that all iRoom applications use the event heap to some degree as an interapplication communication mechanism. An application can be "aware" of other iRoom entities if it can post and subscribe to event heap events. Applications can either communicate with the event heap directly, through Java or C library wrappers around TSpaces calls, or indirectly through a gateway. Gateways provide protocol conversion that allows non-Tspaces clients to effect limited communication with the event heap. We describe how to do this for HTTP clients—that is, a Web interface to the event heap—in the next section and for clients too lightweight to support a full Java virtual machine (VM) implementation in the following section, and what specific limitations are imposed by each type of gateway compared to a direct communication path to the event heap. Figure 2 illustrates direct and indirect communication paths between iRoom entities and the event heap.

Approach 1: Web front end plus infrastructure proxy

Since one domain for PDA usage includes remote-control and universal-interactor applications, it would be valuable to leverage the Web's advantages: familiar user interface widgets, mature development and authoring tools, extensive deployed infrastructure, and the ability to deploy new infrastructure (for example, new servers) using only off-the-shelf commodity hardware and software. To do this, we created an application called the usher that behaves like an HTTP server whose sole purpose is to convert well-formed HTTP GET and POST requests into operations against the event heap. (In other words, it tunnels event-heap remote procedure calls over HTTP.) The usher is currently implemented as several Java servlets that plug into standard Java supporting Web servers.

To post an event, the event fields and values are encoded into either a "fat URL" for a GET (as is done in HTTP GET form submissions) or an HTTP POST form submission. The usher parses the submission and posts the appropriate event to the event heap. To query the event heap for events matching a template, the template is encoded in a fat URL or HTTP POST form, and the event data is returned encoded in the destination page of the HTTP form request. Using this mechanism, the usher can be invoked via any action that causes a link to be followed: user clicking a button, user following a link, HTTP redirect, and so on. The encoding for an event posting can specify the URL of the page that should be returned once the event has been successfully posted to the event heap; this means that Web links can be set up to both trigger an event to be placed into the event heap and send the local browser to a new page. In essence, the usher provides a limited way to interact with the event heap that is understandable by existing off-the-shelf Web browsers and controllable using standard Web pages with links and forms.

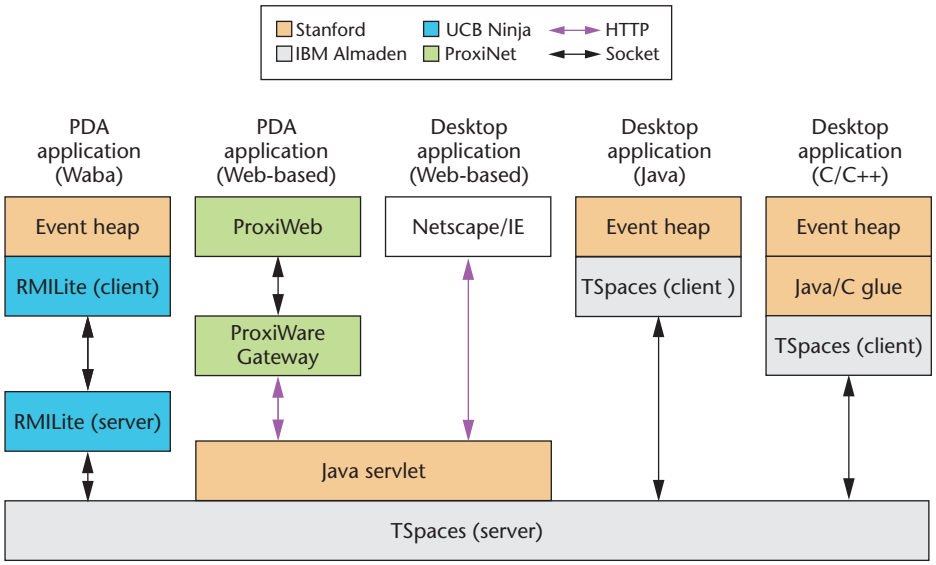
To interact with the usher from a PDA requires only that the PDA be able to run a Web browser that can download and allow interaction with standard HTML pages and forms. Fortunately, such a browser exists: ProxiWeb handles secure and insecure HTML text, forms, images, image maps, and most other basic Web content types. It does not handle dynamic HTML or client-side scripting in Java or JavaScript, so Web applications for PDAs are limited to static content. The ProxiWeb client and service are free; a commercial transformational proxy gateway, based on research described elsewhere,[8] is used to transparently fetch pages from destination servers, convert them to a form viewable on the PDA, and transmit them to the PDA. Figure 3 illustrates the Web interface to the event heap both via a standard desktop Web browser such as Netscape Navigator and via the PDA Web browser, ProxiWeb.

Example: Multibrowsing. Using the technique just described, we created a technology called *multibrowsing* for the iRoom. Multibrowsing lets users construct Web sites in which following links can cause the destination page to appear on any of the iRoom's screens, not just the screen displaying the page containing the link. Displays in the room are multibrowsable if the machine driving the display is running *multibrowsed,* a simple daemon that subscribes to special multibrowsing events from the event heap and causes the browser running on that display to visit the URL encoded in the multibrowse event (or starts a browser if needed). Any event-heap application that knows the format of multibrowse events can therefore trigger any display in the room to show a specified Web page.

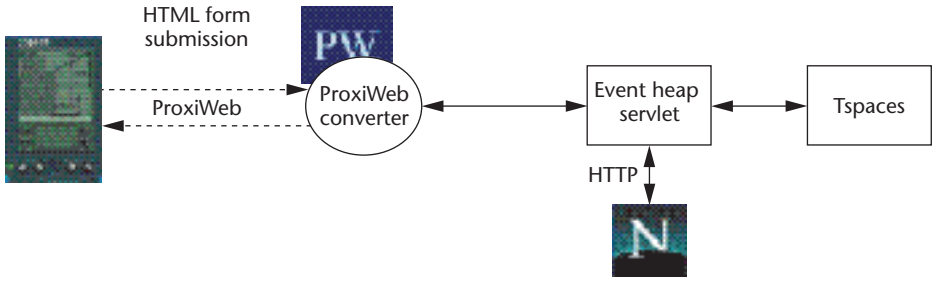
Multibrowse-enabled Web pages are constructed with "fat URLs" that submit a form request to the usher to generate a multibrowse event encapsulating the link URL to follow and the multibrowsable target screen that

Table 1. Some event heap details.

Mandatory Event Fields	Some Currently Supported Event Types
OpCode: Indicates type of fields to expect	Multi-Browse: Specify URLs or applications to load on a target machine
SourceID: All participants are known	Projector: Commands to turn on/off and switch projector inputs
GroupID, PersonID, TargetID: Automatically controlled to allow application grouping and targeting of events	View Change: Causes a 3D viewer to rotate views
Time To Live: Events are cleared from the heap this long after posting, whether they have been read/consumed or not	New Image: Causes an application to load in a specified image
Number of Accesses: Event removed after specified number of accesses	Mouse/Keyboard: Used to retarget mouse and keyboard between machines



2 The various paths through the event heap.



3 PDA Web pathway.

should display the page. Since being a multibrowse controller requires no special support, PDAs can participate as first-class citizens in multibrowsing by running the ProxiWeb browser.

We currently use multibrowsing for running meetings. The moderator constructs a multibrowsable page to choose which applications or Web pages to bring up on a given display as the meeting progresses. Displaying this control page on a PDA permits using all of the screen real estate of the large displays for group content, while leaving the control capabilities on the moderator's PDA.

We've also used multibrowsing to enhance HTML-converted PowerPoint presentations. A control page is created containing links for each step in the presentation. Each such link encodes a mapping of slides to displays for that particular point in the presentation. Note that for each page of information, up to one link per output screen can be created, allowing the user of the control page to display any information on any screen at any point during the meeting.

Example: Projector controller. Since our schema for generating events from fat URLs is quite general, we have also been able to use this mechanism to

InFocus iRoom Projector Control Page

Turn On Projector

Turn on All Projectors: projsubmit 1 1 0 0 null 2 1 0 0 null 3 1 0 0 null 5 1 0 0 null 6 1 0 0 null
Turn on Smartboard 1: projsubmit 1 1 0 0 null
Turn on Smartboard 2: projsubmit 2 1 0 0 null
Turn on Smartboard 3: projsubmit 3 1 0 0 null
Turn on Table: projsubmit 5 1 0 0 null
Turn on Front: projsubmit 6 1 0 0 null

Switch to Table Laptop Drop Display

Switch All Projectors to laptop drop: projsubmit 1 1 14 0 null 2 1 14 0 null 3 1 14 0 null 5 1 14 0 null 6 1 14 0 null
Switch Smartboard 1 to laptop drop: projsubmit 1 1 14 0 null
Switch Smartboard 2 to laptop drop: projsubmit 2 1 14 0 null
Switch Smartboard 3 to laptop drop: projsubmit 3 1 14 0 null
Switch Table to laptop drop: projsubmit 5 1 14 0 null
Switch Front to laptop drop: projsubmit 6 1 14 0 null

4 Part of the projector control Web page that uses multibrowsing.

build a Web page (reproduced in part in Figure 4) that allows control of the various room projectors (turn projectors on/off, remotely control the multiplexer that determines which machine drives which projector, and so forth). Currently the interface is stripped down—clicking on various textual links causes the actions to happen—but it would be simple to decorate the interface with form elements, image maps, and so on. By using URL-passing as a gateway to the event heap, we have enabled very rapid prototyping of this style of remote-control application. Note that this "application" can run on either a standard (laptop/desktop) Web browser or on a PDA via ProxiWeb.

Discussion. Multibrowsing is a simple mechanism with obvious drawbacks. For example, you can open Web pages or start applications, but currently you can't close Web browser windows or kill running applications, nor in general take control of an already-running application not started via *multibrowsed*. Nonetheless, it has proved useful for a variety of tasks, such as those mentioned above, and is implemented entirely using simple event-heap mechanisms and existing Web infrastructure. Other than the creation of pages with properly constructed fat URLs, authoring multibrowsable pages requires no programming skills.

HTML-only user interfaces have well-known drawbacks. The transformations done by the ProxiWeb service have two relevant side effects: control over widget layout is limited, and the ability to do client-side scripting of any kind is lost. The latter restriction implies that every user action requires a roundtrip to the server for the result of the action to be indicated on the PDA screen, rendering this method almost useless for interactions in which the action-perception coupling must be tight and therefore requires low latency—for example, pointer tracking by dragging a finger on a PDA screen. However, the method is still useful for limited-interactivity applications, such as remote controls.

A less obvious but very significant drawback of this approach is that you cannot asynchronously "push" events all the way to the PDA—the best you can do is have the PDA periodically poll for events by refreshing the Web page being viewed. This makes it impossible to write applications in which state changes in some other iRoom entity cause immediate state changes on the PDA.

On the other hand, the advantages of the HTML approach (at least for prototyping) are clear:

- new applications require no compile or install cycle;
- applications work from both desktop browsers and PDAs;
- application designers can use extensive and familiar Web tools to design the application; and
- there is no need to tightly integrate the application with the iRoom infrastructure (for example, no need to publish pages on a specific server).

Given these advantages, Web front-end applications in the iRoom showcase an aggressive embracing of infrastructure computing. First and most obvious, they leverage existing HTTP standards and servers, converting Web data in these protocols to event data transmitted to the event heap using its native Java interface. Less obviously, but perhaps more interestingly, the transformational proxy gateway through which all ProxiWeb requests are routed is not hosted on any machine in the iRoom, nor indeed any machine at Stanford. It is a shared, publicly available service hosted offsite, on a cluster of workstations somewhere in Santa Clara. This offers a strong case for computing in the infrastructure—it illustrates how our ability to exploit existing infrastructure services gives us a huge advantage in leveraging an entire set of solutions (desktop Web browser compatible access to iRoom mechanisms) directly into another domain (PDAs). Finally, pages embedding HTML links or forms addressed to the usher can be published anywhere—they don't need to be tightly integrated with the iRoom infrastructure.

An interesting consequence of our aggressive adoption of this approach is that, if not for our IP firewall around the iRoom, a stranger could publish a Web page on any server, pull out a PDA with a wireless modem (such as a Metricom Ricochet or Novatel CDPD Minstrel), surf to the published page from anywhere in the world, and cause the lights in the iRoom to blink on and off—or any number of other amusing events. We are working on a crisp description of the desired security model for iRoom applications so that we can investigate more flexible security and authentication mechanisms.

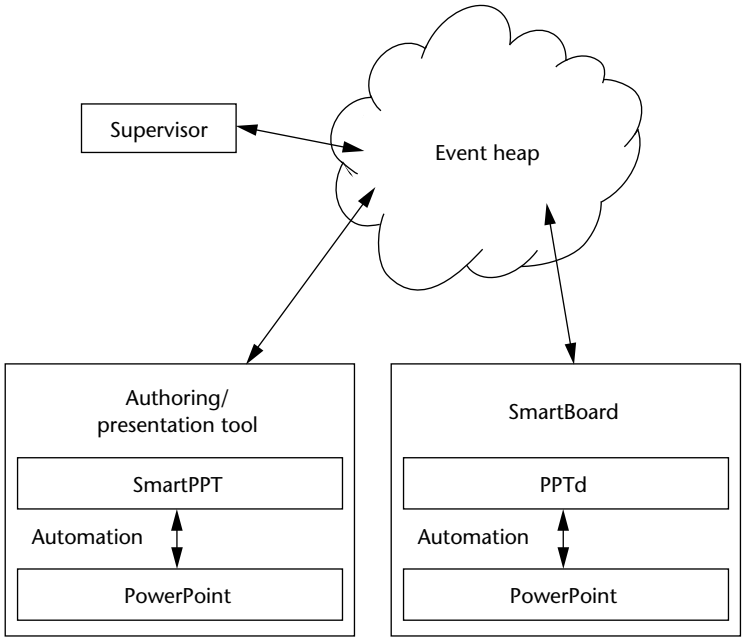
Approach 2: Pseudo-native code plus infrastructure translator

We also used the Waba virtual machine (from WabaSoft, http://www.wabasoft.com) as a method of PDA integration. Waba is a proper subset of Java; virtual machines are available for PalmOS and Windows CE devices. Waba includes class files for common user-interface widgets (menus, buttons, and so on) and a class library that lets Waba applications run on standard desktop PCs that include a Java runtime. Because every legal Waba application is a legal Java application, applications written this way also run in both desktop and PDA environments.

Unfortunately, the memory and CPU speed required to support on-demand class loading and serialization in full generality—both required for true Remote Method Invocation (RMI) support—exceed the resources of current-generation PDAs. Our solution to this comes from another project exploring infrastructure-based computing, the Ninja project[8] at UC Berkeley. One software component from that project is a lightweight (about 1,000 lines), stateless, active proxy module and a lightweight (2,000 lines) client stub that enable a client and proxy to exchange a limited repertoire of messages using RMILite, a limited subset of RMI. RMILite encodes only a few specific types of messages using a minimalist communication protocol and marshalling rules, rather than providing a general class loading and serialization facility. The proxy side of RMILite parses the messages and converts them into remote procedure calls for one of a small number of systems supported, including TSpaces. (The other remote invocation systems supported are Jini and NinjaRMI, a secure implementation of Java RMI.) It also converts messages from TSpaces into RMILite messages and forwards them to the client.

The application designer writes arbitrary Waba code that runs on the PDA and uses the RMILite client stub (a Waba class) to make TSpaces calls via the RMILite proxy to communicate with the event heap. The RMILite stub is written to resemble the native Java RMI methods of its clients; the Waba source for communicating with the event heap via RMILite is almost identical to the Java source for communicating with the event heap using normal Java RMI. One of the event heap infrastructure machines always keeps a copy of the RMILite proxy side running. The application designer then uses the standard Java Development Kit (JDK), plus some Waba tools provided with the distribution, to create a runnable application then installed on the PDA.

Example: Smart PowerPoint. The SmartPPT application described in the first section consists of two distinct components: a presentation management/authoring system and a PDA-based viewer/annotation program. The individual pieces interact with PowerPoint using Office Automation, a set of function calls that lets programmers control Microsoft Office applications. See Figure 5.

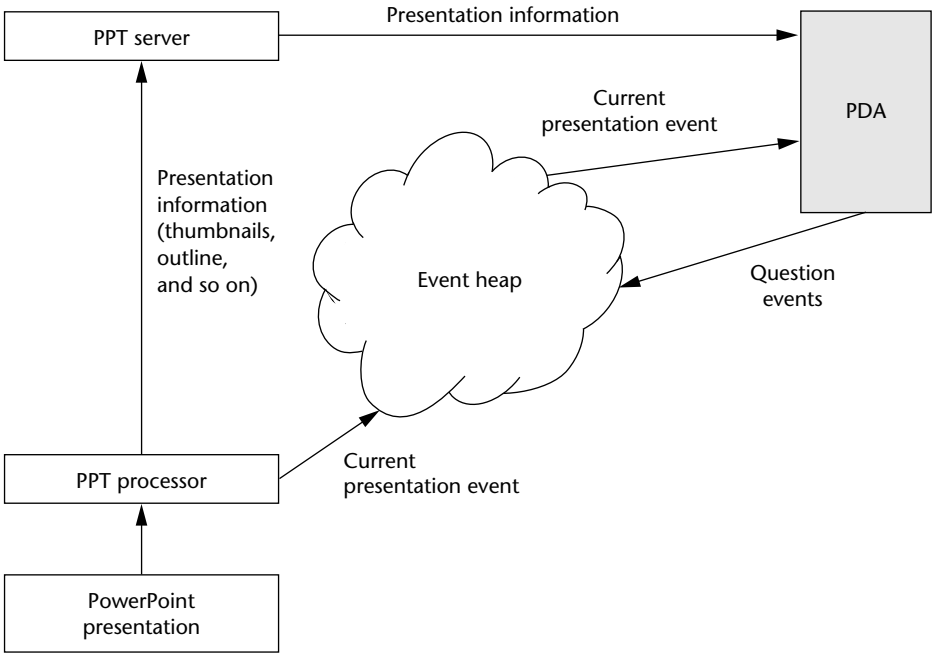


5 Event-heap paths for the SmartPPT presentation system.

The presentation management/authoring system consists of a presentation manager running on the presenter's machine, a supervisor that tracks the available displays in the room, and display controller daemons that run on each large-screen display in the iRoom. Specifically, at initialization the presentation outline and a set of thumbnail slides is requested from PowerPoint and used by the presentation manager to create an interface for choreographing the presentation. During the presentation, the various displays are directed (via events) to update themselves to the slide requested for that screen at that point in the presentation.

The StuPad-inspired, Waba-based PDA client runs on palm-sized PCs. Users can view and annotate a copy of a presentation, as well as exchange comments and questions with each other during a presentation. PDAs running the viewer application are notified via events from the Presentation Manager of an available presentation, in response to which they download the slide thumbnails, presentation outline, and meta-information extracted by the Presentation Manager. (See Figure 6, next page.) A PDA user can then jump to any slide in the presentation and make annotations (currently private and not visible on the large display screens). Questions posted as events through the event heap can be viewed and responded to by other PDA users running the viewer application. Communication between the PDA client and the event heap is mediated using the RMILite architecture.

Discussion. Like the ProxiWeb proxy, the RMILite proxy uses standard wide-area transports (TCP/IP-based) for remote invocation on both the client and the server side. It could in theory run on any machine in the



6 Event-heap paths for the SmartPPT PDA viewer application.

Internet infrastructure. For efficiency and simplicity, and because the RMILite proxy is a small and lightweight process, we run it on one of the iRoom infrastructure machines. These machines include various facilities to keep certain important daemons such as the RMILite proxy and the event-heap server itself running at all times, restart them automatically if they crash or after a reboot, and so forth.

All of the limitations of the Web front-end approach vanish with the Waba approach. Arbitrary behaviors can be coded, including the ability to be asynchronously notified of (subscribe to) specific events. The applications can be ported to any Java environment or to any platform that supports a Waba VM (currently only Palm and Windows CE). This method affords tremendous flexibility, but at the cost of significantly more work: the programmer must be familiar with Java, and the compile/install/run cycle for Waba, while not complicated, is tedious.

Approach 3: PDA native code

Many types of interaction require low-level control over both the user interface and communication method used. Our initial projector control application required performance and user interface flexibility not possible with a PPP network connection and standard PalmOS widgets. For example, the center slider in the projector user interface (see Figure 7) can control several possible projector attributes. The projector control application consists of more than 30,000 lines of client source code to handle low-level serial communications, manage the complex on-screen widgets including the sophisticated slider widget, and (on the server side) receive incoming serial packets from the PalmPilot and send commands to the projectors.

We have begun work on a second version of the projector controller using the Waba approach, connecting the client and server via the event heap, and using the RMILite proxy and PPP for communication. Figure 8 shows screen shots from the second version, which currently has very limited functionality compared to the first version. So far, however, the basic controls and connection to the server work with only around 500 lines of Waba source code, much of it dedicated to positioning widgets on the display. On the other hand, user interface rendering is detectably slower (precise measurements have not yet been made). Moreover, the flexible slider widget is not available in the standard Waba widget set, so simpler and perhaps less intuitive widgets must suffice. The two projector controllers illustrate the fundamental trade-off we made: flexibility and some performance for greatly simplified application development.

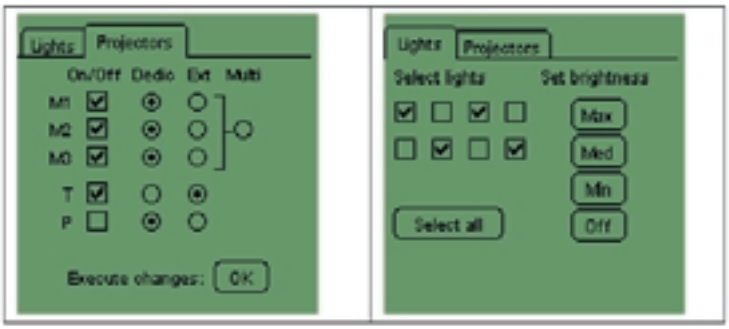
Initial evaluation

The two non-PDA-specific approaches discussed above—Web front-end applications and custom Waba applications—implicitly assume that the device enjoys a relatively fast and always-up Internet connection. Currently we rely on a combination of WaveLAN and PPP over serial cables to achieve this. (We are investigating infrared and Bluetooth.) Note that neither type of application is designed to recover from sustained network outages. Like network computers, these applications can't deal with disconnection in any graceful way.

This fundamental design decision reverberates throughout the design of the software infrastructure. We believe it's justified because

- from a hardware perspective, inexpensive and reliable wireless communication is already real and getting better, and
- from a software perspective, we prefer to concentrate on providing robust and highly available software infrastructure in the iRoom precisely because of the

7 Screen shots from C/C++ native-code version of the projector control application for a Palm Pilot. The slider appears as a tick (left), a gray box (middle), or an "X" (right) depending on whether a particular attribute is settable for a single projector, settable for a group of projectors, or not settable at all.



8 New projector controller written in Waba running on a PalmPilot.

tremendous simplification it affords in architecting solutions to problems such as PDA access.

Note that there exists substantial technology to facilitate operation of sporadically connected (actually, mostly disconnected) PDAs. This holds particularly true in the commercial world, where the vast majority of PDA applications to date (even those that feature Internet access or synchronization against desktop applications) are explicitly based on a "synchronize-then-disconnect" usage model. We haven't yet explored the utility of such applications in the iRoom, but we believe that truly interactive applications, in which the PDAs are essentially always connected, lie closer to the research goals of the Interactive Workspace project as a whole.

Lessons and experience so far

In evaluating what we've learned from our experience so far, we need to consider robustness of the infrastructure, the success of an infrastructure-centric approach, and implementation.

Design for robustness. The event heap as the central abstraction in the software infrastructure came directly from our desire to promote infrastructure software robustness as a first-class goal. The event heap enables improved robustness through decoupling of communicating applications. Rather than establishing tight client-server bindings (or bindings of a number of peer components into a tight group), we prefer indirect communication through the event heap for the following reasons:

- Although applications must still deal with situations such as server failures at the application level, they can make stronger assumptions about lower-level communication mechanisms because the event heap is always available. In particular, it's generally easier to write code to handle the case in which a desired recipient has not responded to a posted event than to write code to handle the case in which the event cannot be posted at all due to a lower level failure.
- The event heap enables the broadcast-style communication required for many group applications, without requiring clients to engineer group communication explicitly. Conventional communication mechanisms, including Active Messages, client-server remote procedure call, and strong group membership with explicit joins and leaves, can all be implemented on top of the event heap if needed, albeit with some efficiency cost.

The decoupling provided by the event heap promotes better fault containment and more robust applications by default. It's still possible to write brittle code using the event heap, but it's more difficult to do so. We have observed this resilience in action during visitor demos: most of the applications being demonstrated are by nature multidisplay applications, typically with autonomous components running on various machines. We have watched a visitor manually kill some of these components (often unwittingly, such as by tapping the close box of a window to get it out of the way), yet the remaining components continue to function gracefully. Specifically, they don't freeze, lock up the machine, or stop responding to user input. We feel we have a good start on providing the tools for building distributed multidisplay applications that actually work and robustly coexist with legacy applications and novice users.

Infrastructure-centric approach. Our PDA programming strategies reflect our infrastructure-cen-

tric computing philosophy: the ProxiWeb proxy, although not even on Stanford premises, is effectively a critical part of the PDA-enabling infrastructure. We're working on deploying a multibrowsing proxy that will transparently intercept and rewrite existing Web pages to make them browsable across the multiple displays in the room. The high-level mechanisms we take advantage of, in particular exploiting HTTP as a transport for simple events, have made prototyping very fast.

Choose a few simple mechanisms and implement them carefully. Using the Web and URL-passing has certainly paid off in trying to get the room running. Still, the real reason for success is the simplicity of the event mechanism itself: unidirectional by nature (though bidirectional mechanisms can be built on top of it), it lends itself to a simple HTML-forms-based instantiation. Since events are self-describing, embedding them in URLs isn't difficult. Similar arguments apply in constructing the "glue" connecting Microsoft Office Automation to the event heap, as we did for the SmartPPT application. Clearly, the benefit of reusing simple mechanisms is the ability to use widely deployed tools (Netscape Navigator, Microsoft PowerPoint) to experiment with new collaborative behaviors. We expect that this will enhance the impact and usefulness of our work to other researchers in this area.

Although this is very early work, so far we have found the event heap to be a reasonable conceptual model, implementable in a robust way. In addition to the applications described in this article, six student projects used the event heap despite a lack of thorough documentation or example code. The students easily grasped both the conceptual programming model and the gateway mechanisms to the event heap, such as the usher (Web interface) or MS Office.

Our overall experience, then, has been that keeping the conceptual vocabulary for programming deliberately limited (the simple event-heap model) has brought real benefits: robustness, a straightforward programming model, portability, and the ability to leverage widely deployed tools and protocols such as MS Office and the Web. The infrastructure-centric part of our approach is key because it moves much of the programming complexity into the infrastructure, which tends to be more stable and change much more slowly than device technology.

Open issues

So far the iRoom is an open environment. An open mechanism, the event heap doesn't enforce authentication of its own. Applications can certainly provide authentication and secure connections at the application level, but this still leaves the event heap open to attacks such as denial of service (perhaps by flooding it with irrelevant events). Worse, if application-level security is too cumbersome to provide, experience in other arenas suggests that programmers might simply ignore it. We're investigating using UC Berkeley's Ninja infrastructure to host some of the iRoom services (it already hosts the RMILite proxy used in the Waba programming scenario). Ninja services can be accessed via secure and authenticated remote procedure calls based on a public key and certificate authority infrastructure, with authentication tokens distributed by a separate key distribution center.

Some relatively mundane issues involving PDA wireless communications have hampered us. In particular, getting fast local-area wireless networking (such as WaveLAN) working with Palm devices and Windows CE palm-sized PCs has been a nightmare. However, our portable-software approaches have allowed us to deploy some of our PS/PC-targeted applications on the Jupiter-class Clio's as well as Wintel laptops, both of which can be easily equipped with WaveLAN. For this same reason, as wireless connectivity for palm-sized devices improves, our portable approach should allow us to migrate easily. A new initiative within the Computer Science Department at Stanford aims to aggressively adopt Bluetooth short-range wireless technology in a variety of projects. We expect to participate in this initiative as it gains momentum and hope to have some experience integrating Bluetooth by the time this issue reaches print.

The event heap's performance for low-level events such as pointer tracking is currently unacceptable, with event-post-to-event-handling latencies of 30 to 100 ms. We know the TSpaces group at IBM Almaden Research Center is working on improving the performance of TSpaces, so we have chosen not to concentrate effort on this problem right now. Furthermore, we have written our own optimized "event fast path" code, currently used to handle low-level events with latency well below the perception threshold.

Conclusions

Even though the iRoom was "booted" for the first time about six months ago, our initial experience with it has been both encouraging and fun. The event heap has so far proven a capable and flexible abstraction for inter-entity communication. The ability to access it via a Web front-end or from the reduced subset of Java called Waba has helped us quickly prototype applications that integrate existing room entities and PDAs—a task that has traditionally ranged from awkward to technically daunting. Now that our basic infrastructure is in place, we expect the pace of research to accelerate. We look forward to further testing and validating our architectural choices. ■

Acknowledgments

The Interactive Workspaces project is the result of efforts by too many students to name here; see http://graphics.stanford.edu/projects/iwork for an exhaustive list and more complete project information. Thanks to the following students for contributions to this article: Henry Berg, Martin Jonsson, Shankar Ponnekanti, and Ranganath Rao. Thanks also to John Gerth and Susan Shepard for the staff support that keeps the project running, Maureen Stone for persuading us to describe our ideas in this submission despite the very early stage of the work, the designers and implementors of SmartPPT (Janak Bhalodia, Rustan Eklund, Hui Huang, Toli Kuznets, Rama Ranganath, Stephen Sorkin, and Krishna Yeshwant), Roy Want for his patience, and the referees and the editorial staff of *IEEE CG&A* for their

helpful suggestions. The work described here is supported by DoE grant B504665, by NSF Graduate Fellowships, and by donations of equipment and software from Intel, InFocus, IBM, and Microsoft.

References

1. T. Winograd, "Towards a Human-Centered Interaction Architecture," to appear in *Human-Computer Interaction in the New Millennium*, J. Carroll, ed., Addison-Wesley, Reading, Mass., 2000, in press; available as a working paper at http://graphics.stanford.EDU/projects/iwork/papers/humcent/.
2. G. Abowd, "Classroom 2000: An Experiment with the Instrumentation of a Living Educational Environment," *IBM Systems J.*, Vol. 38, No. 4, Oct. 1999, pp. 508-530.
3. K.N. Truong, G.D. Abowd, and J.A. Brotherton, "Personalizing the Capture of Public Experiences," *Proc. 12th Ann. ACM Symp. on User Interface Software and Technology* (ACM UIST 99), ACM Press, New York, Nov. 1999, pp. 121-130.
4. T.D. Hodes et al., "Composable Ad-Hoc Mobile Services for Universal Interaction," *Proc. Third Int'l Symp. on Mobile Computing and Communication* (ACM MobiCom 97), ACM Press, New York, Sept. 1997, pp. 1-12.
5. N.A. Streitz et al., "i-Land: An interactive Landscape for Creativity and Innovation," *Proc. ACM Conf. on Human Factors in Computing Systems* (CHI 99), ACM Press, New York, 1999, pp. 120-127.
6. N. Adams et al., "An Infrared Network for Mobile Computers," *Proc. First Usenix Symp. on Mobile and Location-Independent Computing*, Usenix Assoc., Berkeley, Calif., Aug. 1994.
7. A. Fox et al., "Adapting to Network and Client Variation Using Active Proxies: Lessons and Perspectives," *IEEE Personal Communications* (invited submission), Vol. 5, No. 4, Aug. 1998, pp. 10-19.
8. S.D. Gribble et al., "The MultiSpace: An Evolutionary Platform for Infrastructural Services," *Proc. 1999 Usenix Ann. Tech. Conf.*, Usenix Assoc. Berkeley, Calif., June 1999, pp. 157-170, http://ninja.cs.berkeley.edu/pubs/pubs.html.
9. P. Wyckoff et al., "TSpaces," *IBM Systems J.*, Vol. 37, No. 3, Aug. 1998, pp. 454-474, http://www.almaden.ibm.com/cs/TSpaces.
10. A. Fox et al., "Experience With Top Gun Wingman, A Proxy-Based Graphical Web Browser for the USR PalmPilot," *Proc. IFIP Middleware 98*, Springer, London, UK, Sept. 1998, pp. 407-425.

Armando Fox *joined the Stanford faculty as an assistant professor in January 1999, after getting his PhD from the University of California, Berkeley as a researcher in the Daedalus wireless and mobile computing project. 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. Fox received a BSEE from Massachusetts Institute of Technology and an MSEE from the University of Illinois. He is an ACM member and a founder of ProxiNet (now a division of Puma Technology).*

Brad Johanson *is a PhD candidate in the Department of Electrical Engineering at Stanford University. He holds Bachelors degrees in electrical engineering and computer science from Cornell University, a masters degree in computer science from the University of Birmingham in England, and a masters degree in electrical engineering from Stanford University. In the past he has done research on genetic programming and computer networking, and is currently one of the student leads in the Interactive Workspaces project at Stanford.*

Terry Winograd *is a professor of computer science at Stanford University. He has done extensive research and writing on the design of human-computer interaction. Winograd directs the Project on People, Computers, and Design, and the teaching and research program on Human-Computer Interaction Design. He is one of the principal investigators in the Stanford Digital Libraries project and the Interactive Workspaces Project. He was a founder of Action Technologies and a founding member of Computer Professionals for Social Responsibility, of which he is a past national president. He is also a consultant to Interval Research Corporation and on the editorial board of several journals, including Human-Computer Interaction, Personal Technologies, and Information Technology, and People.*

Pat Hanrahan *is the Canon USA Professor of Computer Science and Electrical Engineering at Stanford University, where he teaches computer graphics. His current research involves visualization, image synthesis, and graphics systems and architectures. While at Pixar he developed volume-rendering software and was the chief architect of the RenderMan Interface. He has received three university teaching awards and in 1993 received an Academy Award for Science and Technology, the Spirit of America Creativity Award, and the Siggraph Computer Graphics Achievement Award. He was recently elected to the National Academy of Engineering.*

Readers may contact Fox at fox@cs.stanford.edu.