2002

Designing for Serendipity: Supporting End-User Configuration of Ubiquitous Computing Environments

Mark W. Newman
Carnegie Mellon University

Jana Z. Sedivy
Carnegie Mellon University

Christine M. Neuworth
Carnegie Mellon University

W. Keith Edwards
PARC

Jason I. Hong
University of California - Berkeley

See next page for additional authors

Designing for Serendipity:

Supporting End-User Configuration of Ubiquitous Computing Environments

Mark W. Newman, *PARC*
Jana Z. Sedivy, *PARC*
Christine M. Neuwirth, *Carnegie Mellon University*
W. Keith Edwards, *PARC*
Jason I. Hong, *University of California at Berkeley*
Shahram Izadi, *University of Nottingham*
Karen Marcelo, *PARC*
Trevor F Smith, *PARC*

ABSTRACT

The future world of ubiquitous computing is one in which we will be surrounded by an ever-richer set of networked devices and services. In such a world, we cannot expect to have available to us specific applications that allow us to accomplish every conceivable combination of devices that we might wish. Instead, we believe that many of our interactions will be through highly generic tools that allow end-user discovery, configuration, interconnection, and control of the devices around us. This paper presents a design study of such an environment, intended to support serendipitous, opportunistic use of discovered network resources. We present an examination of a generic browser-style application built on top of an infrastructure developed to support arbitrary recombination of devices and services, as well as a number of challenges we believe to be inherent in such settings.

Keywords

Ubiquitous computing, user experience, recombinant computing, Speakeasy

INTRODUCTION

As we move toward a world in which computation is ubiquitously embedded in our environment and in the objects we carry with us, the range of possible interactions and interconnections among computational devices will explode. Users will have available to them far greater numbers of devices, and far greater numbers of *types* of devices, than we now experience. We believe that, in such a future, users will wish to create configurations and combinations of these devices, and will likely want to create particular configurations that no application developer has foreseen.

Imagine that you want to connect your digital camera to your friend's PDA in order to show her the photos you took on your vacation. This is a task that may be difficult given the state of technology today. Such an interaction might require that both the camera and the PDA share a common memory card format, or that one of the users have a cable appropriate for connecting both types of devices, or that both users be able to connect their devices to larger computers that can send and receive the pictures over the Internet. In the future, if we are lucky, the vendors of these technologies will likely develop a number of ways to accomplish such an exchange, perhaps by creating sets of software services that provide the glue between the camera and the PDA. In a particularly ambitious future, we might even develop standards that allow *all* PDAs and cameras to interconnect with one another.

DIS2002, London © Copyright 2002 ACM 1-58113-2-9-0/00/0008 $5.00

The point to take away from this example however, is that, even if we solve this particular problem we have not even scratched the surface. A solution to this problem does nothing to allow my digital video camera to connect to the same PDA, or my digital camera to display a picture on the new wall-sized interactive surface in the meeting room, or any number of other possible combinations that we can easily imagine.

This scenario highlights one aspect of interacting with technology which we believe will become increasingly important in the future: the ability to improvisationally combine computational resources and devices in serendipitous ways. In the digital camera-PDA example, all of the possible means of interconnection require *some* degree of advance planning. Either you must purchase devices that share a common memory card format, or your must find and purchase an appropriate cable (which will likely not work with any other two cameras and PDAs), or the developers of the devices must explicitly allow the interaction you wish to undertake, by creating special software and even standards to support it. If none of these cases hold true, then the interaction between these two devices is impossible to you. The co-occurrence of any particular set of tasks with the resources required to support it must be foreseen. In a world in which we are surrounded by potentially orders of magnitude more devices, the resources that allow arbitrary ad hoc interconnection are unlikely to be conveniently at hand. Such a mismatch between a desired combination of functions and the tools needed to accomplish them might also arise because users will certainly make use of the resources in their environments in ways unexpected by the creators of those resources [3, 11].

One of the starting points for our work is the belief that systems should inherently support the ability of users to assemble available resources to accomplish their tasks. In a world of richly embedded and interconnectable technologies, there will *always* be particular combinations of functionality for which no application has been expressly written. Supporting serendipitous discovery and use of resources requires that we rethink certain aspects of the way that we design those resources and the frameworks in which they exist. Our research group has been investigating an approach to such problems in the Speakeasy project; we call this approach "recombinant computing."

Central to the recombinant computing approach is the notion that, in such a world, we cannot expect applications to be expressly written to have prior knowledge about all of the myriad sorts of devices they

may encounter. The range of possibilities is simply too large, and anticipating all the potential desired interactions is impossible. But we *can* expect users to have knowledge about the devices they encounter in their environments. In the recombinant approach, users themselves must be the final arbitrator of the semantics of the entities they are interacting with. For example, if my Palm Pilot knows nothing about (i.e., does not have software written to use) printers, the software infrastructure can still provide the necessary technical building blocks to allow my mobile device to communicate with the printer. However, it is the *user* who understands what a printer does and makes the decision of when and whether to actually print, as well as what to print.

This philosophical stance implies requirements not only for the user interface to the system, but also for the underlying technical architecture of the system.

First, both the user interface and the underlying system must allow users to assess what devices and services are available ("there is a printer here; I can access this video projector from my PDA"); determine what their capabilities are ("this is a format conversion service; there is a display here that can show video"); understand their relationship to one another and to the physical and social environment ("this printer is nearby but, it is 'owned' by a user and is not considered public"); and be able to predict likely outcomes from interaction ("the video projector is currently in use by another person"). This implies that devices and services must be able to provide enough information about themselves in a human-understandable form, so that users can make informed decisions about what they and what they will do.

Second, both the user interface and the underlying system must provide a way for users to create and manage the interconnections among devices and services. In addition to making this possible, it is important that the underlying mental model of interconnections, as supported by the system and exposed through the user interface, be as simple and general as possible.

In the remainder of this paper we will give an overview of our recombinant computing approach and discuss a browser based user interface for interacting with the Speakeasy infrastructure. We will also outline our design rationale and report some preliminary user feedback on our system.

RECOMBINANT COMPUTING

Recombinant computing is a philosophy of system design which dictates that computing environments be created from the bottom up, by creating individual entities to be part of an elastic, always-changing whole. Furthermore, this philosophy dictates that these entities, or *components*, be designed and introduced with the thought that they might be used in multiple ways, under different circumstances, and for different purposes.

So far, the recombinant computing "philosophy" sounds similar to the philosophies of encapsulation and reuse behind component frameworks like COM (Component Object Model) [9] or Java Beans [6]. Indeed, our approach builds on the foundations of long established software development approaches such as object oriented design [7] and component frameworks. However, we extend these concepts in one important way: these entities are exposed to *end-users* to be used and configured in highly dynamic and ad hoc ways, such that components can interact and interoperate with each other even though they were built with *no prior knowledge of one another*.

The additional constraint that users be able to use and arbitrarily recombine sets of potentially unknown components suggests additional constraints on the design of the computing framework. One important constraint is that components must expose *recombinant interfaces*. These are specific, simple, well-known programmatic interfaces that govern how components can be made to interoperate with one another. They are basic so that they do not require a large time investment for programmers to conform to them and they are *domain independent*.

This last feature is where our philosophy differs from the established technology of component frameworks such as the aforementioned Java Beans and COM objects, as well as from their more network-friendly descendents like Jini [19]. In these systems, a developer can write a piece of software to communicate with a printer and the programmer of a word processing application such as Microsoft Word can use it as long as they agree beforehand on exactly how the programmatic components will communicate with each other. However, if later on, someone would like to extend the functionality of MSWord to deal with another domain such as digital cameras, they must rewrite part of the application to deal with the new device.

The recombinant computing notion attempts to address this issue by creating *semantically neutral* interfaces that allow entities to interact in at least a very basic way without having to rewrite code. For a more detailed technical discussion, see [5].

Speakeasy: A Recombinant Computing Infrastructure

The Speakeasy infrastructure is a test bed for exploring systems and usability issues in the recombinant computing approach. In this section, we will describe some of the core concepts of the infrastructure insofar as they are relevant to how the user perceives the system.

Components

As mentioned, Speakeasy components are simply entities that can be connected to and used by other components, as well as by Speakeasy-aware applications. Examples of components might include devices, such as microphones, printers, and cameras; software services, such as file servers, databases, and instant messaging clients; and even modules of functionality within larger systems or devices, such as the address book on a PDA.

Any Speakeasy component is described in terms of one or more of our semantically neutral programmatic interfaces. Currently, these include interfaces for data exchange, discovery, user interaction, and the representation of contextual information.

Components can also supply user interfaces at run-time, which allow users to control them directly. Several interfaces can be associated with a single component giving it the facility to cope with several target platforms and tasks. For example, a printer component could provide an administrative user interface implemented in both HTML

and Java as well as a primary user interface implemented as a voice interaction in VoiceXML.

Connections

One of the central concepts in Speakeasy is the notion of a *connection* between components, which is an association among two or more components for the purpose of transferring data. Such a transfer may represent a PDA sending data to a printer, or a laptop computer sending its display to a video projector. A connection typically involves a component that is the sender of data, a component that is the receiver of data, and an application that initiates the connection. The initiating application can monitor and control the state of the connection, and could be a domain-specific tool or a generic browser tool, such as might run on a user's PDA or laptop.

Context

For the purposes of our current discussion, the final important aspect of Speakeasy components is that they all have the ability to provide structured information about themselves, their location, their owners, and so on. This information is largely intended for human consumption, although it could be used by applications to provide particular organizations of a set of available components, or assist the user in selecting components.

INTERACTION APPROACHES

Our work in this area straddles two trends in current research in ubiquitous computing. On the one hand, there is work that attempts to address the user interface with the endpoints of ubiquitous computing, such as work in calm technology [20], information appliances [14], and others [1, 10]. This body of work has a strong emphasis on the user experience but does not generally address how users will cope with unexpected resource availability or adapt the technology to new kinds of tasks.

On the other hand, there is work that addresses the infrastructures that will make ubiquitous computing feasible [8, 17]. This latter category depends, in large part, on the existence of application developers or at least technically sophisticated systems administrators to wire together the available resources into applications that meet the needs of users.

We argued earlier that there is a middle ground however, where end users should be able to configure and assemble the devices and services of interest to them. While allowing users the control to configure programmatic entities is not in and of itself a novel idea, such end-user control is especially challenging in ubiquitous computing environments, in which the devices and environments that are the objects of users' control are likely to be highly dynamic and highly fluid.

There are several interaction paradigms that we considered as models for user interaction in a recombinant world: pipes (à la UNIX), scripting languages, wiring diagrams/dataflow representations, file browser style dragging and dropping and form filling wizards.

The Unix pipe system [13] is a unidirectional interprocess communication channel consisting of three main pieces: a streaming data source, a data pipe, and an endpoint which accepts the data. A user typically types a command string which indicates the programs to use as source and endpoint for the pipe. For example, in a Unix command line window, a user could type "cat DISPaper.txt | grep weasel" to pipe the contents of the DISPaper.txt file to a program that can display any lines containing the string "weasel".

Some scripting languages allow users to assemble predefined UI components into new programs and others allow the automation of existing applications at the UI level. For example, Python [2] is an object oriented scripting language that provides a simple windowing system which is usable with small amounts of programming. The MacOS scripting language, AppleScript [16], allows users to create series of UI choices (open this application, press this button, choose this radio button, close the application...) that mimic the actions that a user would take while using the application. Some scripting environments also allow users to record their actions into scripts which can then be edited and replayed later and others allow these scripting commands to flow over the network to remote computers.

Data flow applications allow users to choose and persist connections between components of a system. Such applications often present a "wiring diagram" model to their users. For example, the musical composition application, Max [4], allows users to place musical components (such as tone generators and timers) onto a canvas and through drag and drop gestures create connections between them. Large networks of components can be created to generate dynamic, complex and original music. The connections in these systems can often have characteristics that are distinct from the components they connect. For example, Max connections can be gated such that they only pass data under certain conditions.

File browsers provide a hierarchical view of files with the ability to take actions such as copy and move. Most users trigger these actions with a drag and drop gesture and the browser indicates action status in a modal popup window. File browsers like the Windows Explorer represent network resources such as printers and shared directories with the same interface language as files and folders, allowing file transfer and printing with the same drag and drop gesture, but delegating more complex interaction to configuration windows or external applications.

Wizards provide a user with a sequence of screens querying for data which can then be used to achieve a goal. Wizards are typically designed for specific branching tasks such as detecting, configuring, and enabling a newly installed device. Often wizards allow users to reverse their progress in the face of new questions that they are unable to answer or recognize as not applicable to their goal. Although this style of configuration is quite rigid and constrained, it has the advantage that it generally requires very little technical expertise.

OUR APPROACH

In choosing an interaction style, our goal was to strike a balance between flexibility and ease of use for non-technical users. For our purposes, we rejected the options of scripting languages or Unix style pipes because they would require a level of technical expertise that we felt was inappropriate for our system although they provide a high level of flexibility and control. The dataflow/wiring diagram approach also seemed to complex for less technical users, as well as being very demanding of screen real estate.

We also rejected the form filling and wizard approaches because, although they are very simple and easy formats for guiding users through a configuration, the very structured nature of this interaction style was not flexible enough for our needs. We wanted to allow users to assemble components in ways that were not predetermined by developers. Nevertheless, we recognized that this style was very effective for structured tasks and adopted it as the interaction style for handling "task templates" which we describe in a later section.

We initially explored an interface with an interaction style much like a file browser. It was primarily intended to be a simple tool for the development team to test the integrity of their software components. It combined a tree-view pane of all available components with a large canvas area upon which users could drag and drop components onto each other to perform simple data transfer actions such as dragging from a service that captures white board images service onto an image viewer service. The application also allowed users to be able to access the user interface provided by each component and view its context.

However, this "drag and drop" approach has the fundamental limitation that it requires significant screen real-estate and an input mechanism to perform direct manipulation. In our target environment, however, we expected our users to interact with the infrastructure via resource poor mobile devices such as hand held PDAs and mobile phones. While it would be possible to develop drag-and-drop wiring diagram-style applications for a hand held device, such application styles are perhaps best served when run on devices with large screens. Such an application must be developed for one particular PDA platform (operating system, screen size, color, and so on), and would not be easily portable to different PDA platforms. This was an important consideration for us since there is little uniformity in the PDA models already in use among our expected users. Furthermore, if at all possible, we did not want to require our users to install any additional software on their devices so that they could simply walk into a networked room and immediately begin interacting with our system.

As a result of these practical deployment considerations, we decided to implement our interface as a web application. Web browsers are already commonly found on hand held devices and many of the PDAs in use by our colleagues have the capability to connect to the lab's wireless (802.11b) network. Moreover, the same application framework could potentially be used on many different devices with different form factors and screen sizes. However, the implementation choice of a web application restricted us with very significant constraints. In particular, the interaction styles available to us were essentially limited to hypertext links, buttons and drop down menus because extensions to web technology that would allow more direct manipulation such as dynamic HTML, are not supported on web browsers for small devices.

THE SPEAKEASY BROWSER

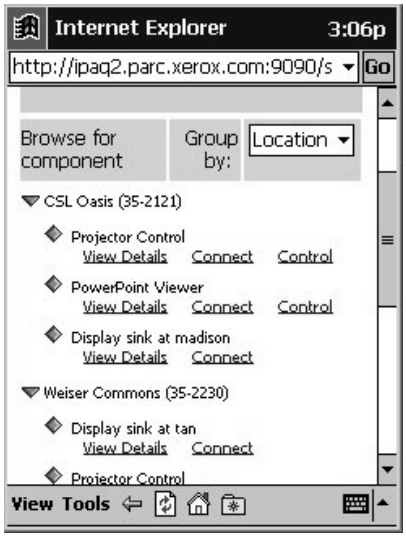
The rest of this paper focuses on the design and preliminary evaluation of a browser for the Speakeasy environment. The purpose of the browser is to allow users to discover, interact with, and connect arbitrary components. We implemented two ways to do this: task-oriented templates and connecting components directly.

Direct Connections

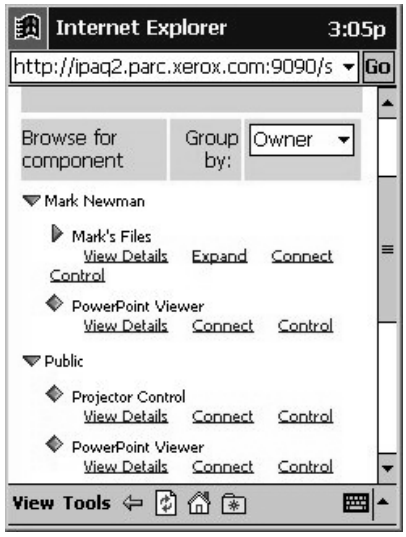
In the most basic "direct connection" mode of operation, the browser provides access to the "raw" functionality of components—by allowing users to directly connect them to one another and to access their user interfaces to control them. In direct connection mode, users can discover, control, and connect any available components—any component on the network is accessible by users at any time.

The browser provides a number of ways of viewing and organizing the available components, each of which might be appropriate under different circumstances. For example, for a given task, some users might want to view components by their location, while another task might be better facilitated by grouping components by their owner. The browser provides support for viewing and grouping components by metadata such as types, locations, and owners (see Figure 1).

The browser also provides access to custom UIs associated with the components. These UIs are provided *by the components themselves*,



(a) Components grouped by location



(b) Components grouped by owner

Figure 1: The browser's component selection screen, shown here grouped by location and owner

and do not require any prior knowledge of the component or the UI on the part of the browser. For example, a projector component may provide a custom UI that presents controls for turning on the projector's power and controlling other features of it. The browser is able to present these controls to the user, even though it knows nothing about what they do.

Task-oriented Templates

While the browser's direct connection mode provides "raw" access to components, an issue of particular interest to us was supporting the ability of users to make sense of the components available to them, especially given the highly dynamic, heterogeneous, and potentially unfamiliar nature of ubiquitous computing environments such as ours. According to sensemaking theory [18], sensemaking needs cannot be considered in isolation of the situations that create them. The theory inspired us to focus on what users might be trying to accomplish by connecting components in particular situations. For example, users might want to connect components in order to give a presentation, send contact information to someone, monitor the dryer in the basement while up in the living room, etc.

We believed that the browser could potentially help automate such situations while remaining ignorant of their specifics. This hypothesis led us to the incorporation of task-oriented *templates*, which are "prototypes" of common tasks that can be created and shared by users. Templates contain a partially specified set of connections and components to be fully specified and "instantiated" by users at the time of use. For example, a template named "Give a presentation" might have slots for a file to be displayed and a projector to display it on, as well as for controlling the room lighting, the projector, speakers, and a file viewer. Slots in a template can refer to fully specified components (e.g., the NEC MT 1030+ projector in Room 17) or can be defined generally, using constraints (e.g., any projector in the building). Thus for a given task, there can be a general template, say, "Give a presentation," in which the "projector" slot is constrained to accept any projector that the user chooses, or a more specific template, say, "Give a presentation in Room 227," with the specific projector for that room already filled in.

Templates must be instantiated by users by filling in their slots to make them concrete (e.g., a file to view must be specified; if a constrained slot is not fully specified, a selection from components that match the constraints must be made). In the language of the browser, when a template is fully instantiated, it becomes a "task" and can be found on a list of "current" tasks. Unlike in direct connect mode, where users have an unconstrained set of components available for selection at all times, a particular template will constrain the components available at any given time according to the semantics of the template.

The browser is designed to support the creation of new templates by example. Users can modify existing tasks by adding and removing connections and components, as well as by changing the components with which the slots are instantiated. At any point, tasks (as well as connections made directly) can be saved as templates. When saving, users are presented with a dialog that allows them to (optionally) generalize each slot according to constraints such as "accept components whose type is 'Projector'" or "accept components whose owner is the current user." Currently we do not support creating templates from scratch, though we plan to support such a feature in the future.

We contend that templates fill an important niche in dynamically changing computational environments, because they stand in a middle ground between completely unstructured interaction with individual components and a predefined, potentially inflexible application written by a developer. Templates impose a layer of semantics on top of the raw facilities offered by the infrastructure, and assist in sensemaking by constraining the choices available to the user to only those components appropriate for the task at hand. They also make it possible for more guided forms of user interaction such as the wizards and form filling styles mentioned earlier. Furthermore, the ability to potentially share these templates between users could eventually support communities that share and exchange useful templates [12].

DESIGNING AND TESTING THE BROWSER

The design and development of the browser application emerged in parallel with the evolution of the underlying infrastructure. The purpose of deploying and testing the browser was twofold. First, we wanted to confirm that the core concepts of components, connections and context in Speakeasy were understandable to users and could be conveyed in a simple interface on a resource-poor environment such as a handheld device. Second, we wanted to validate the viability of our recombinant programmatic interfaces at the engineering level. Thus we designed and tested two versions of the browser. The first consisted entirely of mockups, and focused on the core application concepts and their presentation to the user. The second version was a functional version of the browser that communicated with the set of "live" components (such as projectors, printers, cameras, and the like) we had already built during earlier phases of the Speakeasy project.

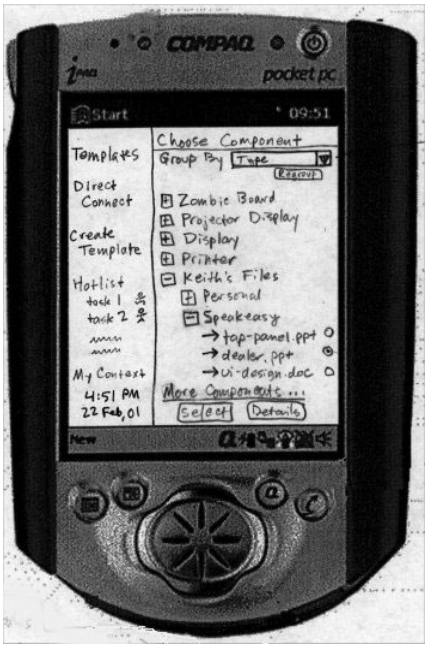


Figure 2: An early, low-fidelity sketch of one of the screens for the Speakeasy Browser. The sketch was drawn inside a photocopy of a Compaq iPAQ PDA, and shows an earlier version of the screen shown in Figure 1

We began our design exploration with sketches on paper (see Figure 2) to outline core aspects of the browser's functionality, such as finding, connecting, and controlling components, as well as the notions of tasks and templates. These sketches allowed us to develop the basic UI concepts of connections and templates, so that we were able to evolve the general task "Give a presentation in the Oasis (a conference room)" into the following scenario:

> Pat, a manager, is planning to give a presentation in the Oasis later in the afternoon. Pat brings up the Speakeasy browser on a PDA, goes to a list of templates and chooses "Give a presentation." The template configuration screen appears, and Pat selects "File name: Choose" to bring up a list of components including Pat's file space. After selecting a PowerPoint file, as prescribed by the template, Pat is returned to the template configuration screen with the name of the specified file filled in (as in Figure 3). Next, Pat selects the slot for the Projector and is shown a list of projectors, arranged by location in the building. Pat selects the projector in the Oasis, at which point slots for controlling the projector and for controlling lights in the room are also filled in with defaults specified by the template.
>
> Having finished configuring the template, Pat names the task "Give a presentation in the Oasis" and places the task on standby. That afternoon, she takes her PDA to the Oasis, and uses the browser to select the task from the list of current tasks and chooses to "Run" it. The controls for the components involved in the task appear in the browser and she is able to dim the lights and control the presentation (e.g., advance to the next slide, return to a previous slide, etc.).

After several iterations of the sketch-based prototype, we split the effort into two parts—one effort began developing the backend functionality of the browser, such as discovering components and making information about them available to our Web application, and the other effort focused on refining the browser's user interface.

This latter effort began by developing a higher-fidelity, interactive, "mockup" prototype in HTML. We developed the prototype and iterated on it, gaining valuable experience by employing walkthroughs and user evaluations between iterations. In order to simulate an environment in which a myriad of components would be available, we populated the mockup with several dozen components to choose from, as well as about a dozen templates representing potentially useful tasks (e.g., give a presentation, share my slides, listen to voice mail, capture the whiteboard, print a document, etc.).

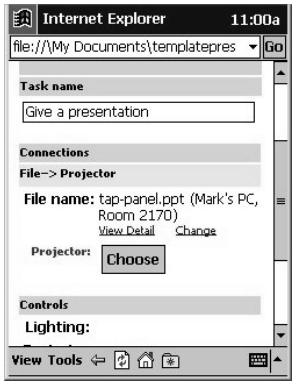


Figure 3: A partially configured template. In this case the user has selected the file to be presented but has not yet selected the projector.

The fact that the development of the functional version of the browser proceeded in parallel with the development of the prototype meant that design elements from the prototype could be folded into the "real" browser as they were established and tested. At the time of the second evaluation (described below), most of the features of the mockup had been implemented in the browser, though a few key features were still missing (for example, the ability to filter out components that are irrelevant or invalid for a given connection endpoint or template slot and the ability to define templates with sophisticated constraints).

Testing With Users

We performed two sets of evaluations with users. The first was conducted using the mockup prototype and the second was conducted using the functional browser. For both sets of evaluations, we gave each person a Pocket PC with Internet Explorer displaying the start page for our browser application. Both sets of evaluations were based on the scenario that we outlined earlier, namely that of setting up and giving a presentation using the PDA-based Speakeasy browser.

First Evaluation

For the evaluation of the mockup prototype, we asked users to carry out four activities, including two drawn from the scenario above (create a new task from template and carry out the task by dimming the lights and controlling the slide presentation) and two others (suspend the presentation to allow a colleague to present, and connect two components directly—without a template). We conducted the evaluations over a two-week period, during which we continued to iterate the prototype based on feedback from each evaluation. In all, we observed seven users attempting to use the browser in a "walk up and use" test. With the exception of one user, all participants were experienced computer/Web users (4 were programmers) and familiar with operating a PDA. The following were the only instructions that users were presented with:

> This Speakeasy browser allows you to connect things together wirelessly. For example, you might connect your PDA to your stereo in order to control it, or you might connect your laptop to a projector in order to display a file.
>
> You do this by creating **tasks** and running them. To make this easier, common types of tasks have **templates**, but when there's no template, you can also **connect** things **directly.**

With the exception of the inexperienced Web/PDA user, all users were able to form a mental model of how the system worked and how it could be used to complete the tasks. Although they experienced problems and uncertainties, the six more experienced users completed all of the activities successfully, even in the earliest stages of the prototype.

Second Evaluation

The evaluation of the functional browser took place about six weeks later, and covered two of the activities from the mockup evaluation (setting up and using a task), though the instructions to the participants were worded in a more open-ended way. Rather than direct them to use a template as we had done in the first round of evaluations, we simply asked the user to present several slides from a recent talk of their choosing. Six people participated, and each

emailed us the PowerPoint file for their talk in advance. We placed their file in a "file space" (a collection of files accessible via Speakeasy) that would be accessible through the Speakeasy browser before the evaluation session. All were experienced computer users and four were programmers.

Users had the option to set up for the talk by using one of two templates (named "Give a presentation in Commons" or the more generic "Give a presentation") or by carrying out connections directly. All but one of the participants elected to use the more specific template, and were successful at completing the setup. The lone participant who attempted to make the connection directly was unable to complete the task without intervention. One other participant initially tried to make a direct connection but quickly gave up and used a template. After finishing the setup, all six users were able to carry out the talk with ease.

Observations

Participants in both studies demonstrated significant confusion when presented with large numbers of available components, particularly when the components were not relevant to their current tasks. We had suspected that some form of information filtering would be important, but we hadn't focused on this in our initial designs. After the first couple of mockup evaluations, however, it became clear that this was a significant source of confusion and we modified the prototype to include better filtering. For example, we augmented the functionality of the templates to increase the "intelligence" with which components could be filtered, by allowing the inclusion of constraints based on variables such as "owned by the current user" or "locations near me." In addition, we added assumptions about the ability of the browser to make certain filtering decisions by default, such as restricting the scope of the components shown to those in nearby locations or within a certain administrative boundary. These changes considerably reduced the confusion of for later participants in the mockup study. However, we did not fully implement the augmented information filtering capabilities in the functional browser, and we therefore observed the exact same confusion as we had during with the first mockups, thus reaffirming the importance of filtering.

Different users took different approaches to complete the task of setting up the presentation. For example, one user was focused on finding the meeting room in which the presentation specified by the task was to be held. She believed that once she found and selected the room, everything else would fill itself in correctly. Another user, an experienced user of Microsoft Office, believed that if she could just "find PowerPoint," she would be able to set up the presentation correctly. Several users focused on "finding the file," and believed that once they had located their file, the rest would fall into place. Fortunately, the browser seemed to support all of these strategies fairly well, thus supporting our initial assumption that designing for flexibility in the ways that users organize components and carry out tasks would be important.

Another problem we observed was the lack of feedback about the state of the world resulting from actions taken within the browser. When designing the mockups, we had overlooked this problem in part because we assumed that in a "live" browser, actions taken within the browser such as turning lights on and off or displaying a presentation on a projector would have obvious effects on the environment and would therefore not need to have their results represented in the browser. During the evaluation, however, several participants misunderstood the instructions for one of the tasks and ended up turning on a projector and displaying their talk in a room other than the one in which they were located. Since the browser did not tell them what action had been taken and they could not observe any effects in their immediate environment (and would not have been able to do so, even with a "real" browser and environment), they were understandably confused about what they had done.

Feedback problems did not disappear when we tested the functional browser. For example, after turning on the projector using the browser, it would take a few moments to warm up before it would project an image on the screen. During this time, some users were not sure whether or not they had performed the correct action to turn on the projector and whether the system had successfully carried out their action. There was no feedback from the browser to allay either of these uncertainties. (It should be noted, however, that the remote control normally used to turn on the projector does not fare any better in this regard.)

DISCUSSION AND OUTSTANDING ISSUES

A key motivating question for our work in this area was the issue of how to effectively design user interfaces that would inherently support "serendipitous" discovery of available resources and co-adaptation of tasks. While we believe that we have taken significant steps in this direction, the very nature of the goal makes it difficult to evaluate. Because of the limited range of components we had developed so far, the tasks assigned to our users were by necessity quite constrained and limited in scope. Indeed, by their very nature it is difficult to dictate "serendipitous tasks" to test subjects. Rather, these types of interactions arise out of natural, day to day contact with the technology. Our ultimate goal, naturally, is to develop our system to the point where it is mature enough, and with a rich enough complement of components, to widely deploy in our lab so that our users can use the system in an unstructured way.

Nevertheless, the deployment and testing of the Speakeasy browser provided us with important information regarding the engineering concepts of recombinant computing and towards user interaction issues in our environment. This discussion focuses on the latter.

We believe (and participants in both studies agreed) that we ended up with a usable browser as well as a substantial list of additional possible improvements. However, we further believe that some of our experiences have implications beyond the specific browser application described in this paper. As mentioned before, we believe that browser-style applications that allow users to discover and interact with arbitrary devices and services in an ad hoc fashion and will play an important role in future ubiquitous computing environments. In the absence of specialized applications for every conceivable task, a more generic tool—one without specialized domain knowledge—will necessarily play a part. In this section, we explore some potential implications of our work on this wider class of applications.

Task-oriented Templates Provide Benefits

Task-oriented approaches to presenting information are commonly found in instruction documentation, where, for example, task-oriented

documents have been found to improve the productivity of users significantly and are strongly preferred over feature-oriented documents [15]. One of our main questions was "Would this hold true when the tasks were as diverse as might be encountered in a ubiquitous computing environment?"

Users experienced far greater success in accomplishing tasks when they used templates than when they did not. Task-oriented templates helped with sensemaking, since they informed the user about what kinds of components they could expect to find and what kinds of actions would be appropriate with them.

In some sense, templates serve as a lightweight middle ground between dedicated, domain-specific applications and fully general access to components (as is available in our "direct connect" mode). Templates capture certain domain semantics—what components are appropriate for a particular task, what constraints are salient for them, how they should be connected—without requiring full-fledged programming on the part of a developer or user.

An interesting question is, "What is the right level of generality for templates?" If you have five meeting rooms it is reasonable to imagine that you would have custom templates for each room to allow users to give presentations, print documents, capture the whiteboard, etc., in each meeting room. What if you have 30 meeting rooms? 200? Are there designs that avoid (a) overloading the number of templates, thereby making it difficult to locate and choose the one relevant to your current task, and (b) overburdening the user by forcing manual configuration of too-general templates?

Information Filtering is Important

In rich environments, some form of information filtering is crucial. In our experiments, the set of components could be filtered by the type of the component (projector versus printer, for example), as well as location, owner, and other contextual attributes. An interesting research question remains: what attributes are most useful, and for what tasks and types of components? These attributes must be reflected in the constraints provided by the templates, and thus have implications for infrastructure builders.

Further, our information filtering was only static—components did not update their contextual information, and the organization of components was not responsive to the *user's* current context. We believe a more dynamic approach to information filtering, in which the organization presented to the user is tailored to the user's location, history, and tasks, could prove useful. This remains an interesting avenue for future research.

Support Multiple Strategies for Carrying out Actions

Different people take different paths to carry out the same task. In our studies, we saw approaches that can be described as location-, data-, and device-/service-centric. We were pleased to observe that our browser supported each of these strategies fairly well.

Several users noticed that they could apply different metadata-based groupings to the components when searching for them. These users were consistent about grouping by location when searching for things like the projector control and the PowerPoint Viewer (in other words, components with a physical embodiment or effects that would be tangible in a particular space), and grouping by owner when searching for files. It remains to be seen whether other types of metadata (e.g., frequency of use) will prove valuable for other types of components.

The ability to discover and organize components based on more than just simple location seems important. We were interested to discover that our users made use of remote components (such as files on their desktop machine), perhaps indicating that proximity-based networking technologies may be insufficient for many users' needs.

Give Redundant Feedback About the State of the World

In both sets of evaluations, it was clear that the lack of feedback within the browser about the results of users' actions was problematic. Users lacked confidence in the results of their actions, not only when the effects of their actions were invisible from the current location, but even when those effects could, in principle, be observed locally. The latter might occur when there is a failure in some other part of the system (e.g., a service failure or disconnected cable), or because the effects of an action are not immediately apparent (e.g., projector is slow to warm up).

While so-called "invisible interfaces" have been espoused by some in the ubiquitous computing community, we feel that users in such environments will need *more* tangible feedback and control, not less. For example, being able to ascertain what components are currently doing would be valuable for users attempting to discover whether components are available for their use, and what communication between components is taking place. We have recently added a facility for determining such information to the core infrastructure, although it is not yet accessible through the browser.

The need for effective, continual feedback about the state of the world and the results of actions in a dynamic setting where components may come and go, failures may occur, and the state of the world may change rapidly, suggests that "pull"-oriented UI technologies (such as the Web), may be inadequate. We plan to investigate other implementations to better understand the tradeoffs involved.

Tiny Laptop or Big Remote Control?

There was quite a bit of variability among the participants in terms of their expectations about the degree of automation provided by the browser and/or environment. For example some users, after having selected the file for their template and "run" the resulting task, expected to have to turn on the projector manually and actively searched within the browser for the projector controls. Other users expected that the projector would be automatically turned on for them and set to the correct input, and were utterly mystified when this did not happen. One member of this latter group thought that if he could just "open" his file, everything would just work. This sentiment was similar to that expressed by the participant in the mockup study who said, "I just want to choose one button and have it work." Tantalizingly, these expectations of greater automation seemed to come more readily from participants with less computer programming background, perhaps suggesting that users who are less familiar with the limitations of technology were more inclined to have higher expectations about its capabilities.

Another way of explaining these differences, however, is that users' attitudes might have been conditioned by their perception of what this particular application running on this particular device was most like. The users who expected more automation seemed to also follow a model of how they would perform the task using a desktop or a laptop computer. They expected that once they found their file, they could just "open it" as one would do by double-clicking in the Windows Explorer or the Macintosh Finder. On the other hand, users who expected less automation seemed to regard the browser as something more like a sophisticated remote control, where the main advantage of the browser was that it provided convenient access to controls for the various devices and services. Upon reflection, we realized that the latter model is more akin to what we, the designers, had in mind, but the former model is also quite compelling and we plan to investigate adopting some aspects of it in the next version of the browser.

The challenge here is to understand, and hopefully even exploit, users' expectations about the affordances of an environment, based on the device they use to interact with that environment. Different models are possible, and may be more or less advantageous in certain circumstances.

CONCLUSIONS

We have described a possible future in which arbitrary devices and services can be interconnected and used without prior knowledge of one another. We believe that such a world will bring with it a host of questions about what user actions can—or should—be supported: how will users discover the devices and services that are around them, and how will they organize, understand, and ultimately use these devices and services to accomplish some task?

The browser described in this paper is a first attempt at exploring issues of user experience in a radically interoperable and dynamic world. While perhaps limited, we believe that our experiences with this tool have nonetheless yielded interesting results, and point the way toward further improvements in not only the user interface, but also in the underlying recombinant infrastructure we are using and developing.

CONTACT INFORMATION

Jana Sedivy or Mark Newman
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304
+1 650 812 4427 (Jana)/+1 650 812 4840 (Mark)
{sedivy, mnewman}@parc.com
Also see **http://www.parc.com/speakeasy** for more information about the Speakeasy project.

REFERENCES

1. Abowd, G.D., Classroom 2000: An Experiment with the Instrumentation of a Living Educational Environment. *IBM Systems Journal, Special issue on Pervasive Computing*, 1999. **38**(4): p. 508-530.

2. Ascher, D. and M. Lutz, *Learning Python*. Sebastopol, California: O'Reilly and Associates, 1999.

3. Carroll, J.M., W.A. Kellogg, and M.B. Rosson, The task-artefact cycle, in *Designing Interaction: Psychology at the Human Computer Interface*, J.M. Carroll, Editor. Cambridge University Press: New York. p. 74-102, 1991.

4. Cycling '74, *Max*. http://www.cycling74.com/products/max.html

5. Edwards, W.K., M.W. Newman, and J.Z. Sedivy, *The Case for Recombinant Computing*. Technical Report CSL-01-1, Xerox Palo Alto Research Center, Palo Alto, CA, April 20, 2001.

6. Englander, R., *Developing Java Beans*. Sebastopol, California: O'Reilly and Associates, 1997.

7. Gamma, E., R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*: Addison-Wesley, 1995.

8. Huang, A.C., B.C. Ling, J. Barton, and A. Fox. Making Computers Disappear: Appliance Data Services. In Proceedings of *7th ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 2001)*. Rome, Italy, July 2001.

9. Iseminger, D., *COM+ Developer's Reference*: Microsoft Press, 2000.

10. Kidd, C.D., *et al.* The Aware Home: A Living Laboratory for Ubiquitous Computing Research. In Proceedings of *Second International Workshop on Cooperative Buildings* 1999.

11. Mackay, W.E. More than Just a Communication System: Diversity in the Use of Electronic Mail. In Proceedings of *Conference on Computer Supported Cooperative Work*. Portland Oregon: ACM 1988.

12. Mackay, W.E. Patterns of Sharing Customizable Software. In Proceedings of *Conference on Computer Supported Cooperative Work*. Los Angeles, California: ACM 1990.

13. Newham, C. and B. Rosenblatt, *Learning the bash Shell*. Sebastopol, California: O'Reilly and Associates, 1995.

14. Norman, D., *The Invisible Computer*. Cambridge: MIT Press, 1998.

15. Odescalchi, E.K. Productivity gain attained by task-oriented information. In Proceedings of *33rd International Technical Communication Conference*. Arlington, VA USA: Society for Technical Communication 1986.

16. Perry, B.W., *AppleScript in a Nutshell*. Sebastopol, California: O'Reilly and Associates, 2001.

17. Salber, D., A.K. Dey, and G.D. Abowd. The context toolkit: aiding the development of context-enabled applications. In Proceedings of *Conference on Human Factors in Computing Systems (CHI '99)*. Pittsburgh, PA USA. pp. 434-441, May 15-20 1999.

18. Savolainen, R., The sense-making theory: Reviewing the interests of a user-centered approach to information seeking and use. *Information Processing and Management*, 1993. **29**: p. 13-28.

19. Waldo, J., The Jini Architecture for Network-centric Computing, *Communications of the ACM* pp. 76-82, 1999.

20. Weiser, M. and J.S. Brown, The Coming Age of Calm Technology. 1996. http://www.ubiq.com/hypertext/weiser/acmfuture2endnote.htm