INTERACTING WITH FEDERATED DEVICES

Elmar Braun[*] and Max Mühlhäuser[†]

Abstract. *Some ubiquitous computing visions propose to embed an abundance of input and output devices in the environment. Depending on the context, the user can either interact with her limited personal device, or input and output devices from the surrounding infrastructure. But what about situations in which the user has need for the capabilities of both types of devices at the same time? A user might need to display something privately, while needing the screen real estate of a large wall mounted display. For such situations we propose using mobile devices together with those provided by the environment. We have developed a runtime environment that coordinates multiple devices to do just that. Currently we are exploring the usability of the use of multiple devices concurrently. We are also investigating the problems of authoring such interfaces.*

1. Introduction

Many ubiquitous computing projects have explored embedding all sorts of input and output devices in the environment. It started by making desktop computers context aware. Rather than being tied to one's personal computer, users could walk up to any computer and start using it, because it recognized the user and automatically displayed her desktop [3]. With modern hardware it is becoming affordable to embed large flat touch screens into the walls of buildings.

All this hardware in the environment should make interaction very convenient. Of course there are a few cases in which users cannot rely on the infrastructure. If that is the case, the ubiquitous cell phone (or some other private device like a PDA) offers last resort fall-back means of interaction. However, upon closer examination, hardware embedded in the environment is quite limited in many contexts:

- **Spatial restrictions:** If a screen (and the displayed content) is sufficiently large, it can be read from some distance. However, the context from which one can interact with a *touch* screen shrinks down to an arm's length.
- **Device restrictions:** Although it might be inconvenient in some cases, a touch screen at least allows the user to approach it to interact. But what about a display without touch interaction? It might not be a touch screen because it is mounted out of the user's reach, because of cleanliness, or simply because it is cheaper. Unless the environment provides other means of input, communication with that display will be one-way only.
- **Social restrictions:** If a large display is mounted in a public space, it is not unlikely that it can be seen by multiple people. This prohibits privacy, so any interaction that involves private data

[*]Telecooperation Group, Darmstadt University of Technology, Germany, email: elmar@tk.informatik.tu-darmstadt.de
[†]Telecooperation Group, Darmstadt University of Technology, Germany, email: max@informatik.tu-darmstadt.de

cannot be performed on the display. This restriction also applies to public versions of other modalities. A loudspeaker for voice interaction is at least as non-private as a public display, and even more disrupting for people other than the user.

If the user has a personal mobile device, she can resort to using that device in all those cases. However, this means that little remains of the convenience that embedded input and output devices in the environment promised.

2. Federated Devices

Our proposal is simply to use multiple devices *together*. Instead of forcing the user to choose between using *either* the infrastructure *or* her personal device, the two should cooperate. If the user can use two (or more) devices together, each of the cooperating devices can specialize on rendering that aspect of the user interface which it is best suited for. Consider the three points from the last section:

- **Spatial restrictions:** A personal mobile device is usually carried within the user's reach. Using the mobile device allows to interact with the public screen from anywhere as long as it is visible.
- **Device restrictions:** When a display has no means of input, the mobile device can substitute. While input on many mobile devices is far from perfect, it is better than no means of input at all.
- **Social restrictions:** If only some parts of a user interface handle privacy sensitive data, the private device can display this part, while the public display handles the non-sensitive remainder.

Another way to look at it is from a mobile user's viewpoint. Interaction with mobile devices is constrained by the small screen size. Associating a public display gives the user interface a lot of space into which it can "overflow". A user who is roaming within some augmented environment can move about freely while using her personal device. When she has need for a larger display, she simply associates it temporarily from the environment. When she is done, she moves on, again interacting solely through the handheld.

We refer to a set of devices, which cooperatively and concurrently renders a user interface, as a *federation*. We call the user interface rendered on a federation a *federated user interface*. The obvious term "multi-device user interface" was not used, as it is already used elsewhere in a different sense (see below).

3. The Multi-Device Design Space

In order to analyze related work as well as our own project, we have developed a classification for multi-device applications. Figure 1 shows a simplified version of it.

We distinguish between sequential and concurrent use of multiple devices. Sequential use means that a user is roaming from one device to another. Concurrent use means interacting with multiple devices at the same time. Concurrent use does not rule out sequential use. Some systems might allow roaming from one federation to another.

		Use of Devices	
		Sequential	Concurrent
Supported Device Types	Multiple	Device Independent Roaming	*Federated User Interface*
	Single	Teleporting	Multimon

Figure 1. The Multi-Device Design Space

The other distinction is whether a system supports multiple types or only one type of device. For example, Teleporting [3] allows users to roam from device to device, but only supports desktop computers. It is not possible to teleport from a desktop to a voice-based device. Hence we classify Teleporting as *sequential use of a single device type*. If a system allows roaming from one type of device to a different one, then it is classified as device independent roaming *(sequential use of multiple device types)*.

Concurrent use of a single device type means that the user is interacting through an array of one type of device. “Multimon” projects, which arrange multiple displays to render a single large desktop, fall in that category. Such projects usually attempt to hide the fact that they are interacting with multiple devices from the user, and are therefore not of much interest to us.

Concurrent use of multiple device types means that the user interacts with a heterogeneous set of devices, such as a handheld and a wall mounted display. Exploiting such heterogeneous device sets is our primary focus of work.

4. Runtime Environment

We have implemented a runtime environment that allows us to coordinate device federations (cf. section 7.). It implements a model-view-controller pattern, in which view and controller are distributed to a set of devices, while the model resides on a server. If an input happens, the device sends an event to the server, which updates its state, and notifies all other devices of the update.

Our implementation is loosely oriented on a web metaphor. One could consider our client-server implementation to be a distributed web browser, which maintains its state on a server, while it can render a window on a number of remote devices.

As normal and large screen client we use a web browser. Communication with the server and interactivity is handled by an embedded Java applet. The other devices for which we implemented clients are PDAs (Java), cell phones (J2ME), and voice recognition. The implementation is explained in more detail in [5].

Devices can be associated and dissociated at runtime. We have connected it to two different sensor systems (infrared proximity sensing; absolute location sensing based on infrared beacons and infrared stereo vision), and thereby support roaming as well. The user can move about, the system automatically detects which devices are near, and uses them to interact with her.

5. Usability of Federated User Interfaces

Currently there are only a small number of implementations of federated user interfaces (see section 9.), and most of these have not evaluated usability through user studies. Therefore little is known about the usability of federated user interfaces.

Designing for federations is difficult, as the design space is much greater than for a desktop computer. The designer has multiple channels of the same modality, but with different characteristics, at her disposal. For each task or even widget she has to decide whether it will be offered on the handheld, on the large screen, or on both.

We have a two pronged approach to developing something like an interaction pattern language (an ordered library of working design ideas) for federated user interfaces. On the one hand, we are evaluating the few existing projects that use federated devices. We want to see whether these designs might be used as a starting point for a pattern language. On the other hand, we are designing some applications of our own, in order to subject them to user studies. We try to pick designs that are not already covered by existing projects, in order to fill the blank spots in our library.

Strictly speaking these are not (yet) real interaction patterns, as they mostly were designed, rather than extracted from the experience of proven successful federated user interfaces. Nonetheless the format of an interaction pattern language proved to be a good way of expressing them. Examples of our design ideas include patterns for remote control, for privacy, for using either the mobile or the fixed device as “cheat sheet” that explains the use of the interface on the other device, and others.

6. Authoring for Federated Devices

But how realistic is it to assume that federated user interfaces will be designed manually? The scenarios we explore are scenarios of frequently changing context: the user moves about, roams from one set of devices to a different one, and so on. This means that a human designer would have to anticipate and make provision for any set of devices that the user might employ to interact.

This problem already exists today, when one tries to author an application that works both on desktop computers and a variety of mobile devices. The solution to this is called device independent authoring, multi-device authoring, or single authoring: the designer specifies the user interface in a device independent manner. This description is then automatically adapted at runtime to the terminal device.

As the set of feasible device combinations is even more diverse than the set of mobile devices, the authoring problem is even worse for federated user interfaces. Therefore we have worked on a single authoring scheme, which picks patterns from our library of design ideas, according to the currently available devices and the interface description, and generates a federated user interface on the fly.

7. System Walk-through

This section will demonstrate how authoring, rendering and use of a federated are done using our system. Figure 2 shows an overview of the participating components. We have a working prototype of this system (cf. section 4.).

The application in this example is a presentation software for displaying (not editing) presentation slides. It has three main components. The first component is the presentation area, which displays the slides. The second component is a note area, which allows the presenter to make and view annotations for each slide. The third component is the navigation to switch between slides.

The author of the application writes a single XML document describing the user interface. She marks the three components using grouping tags around the contained widgets. She also marks the note area as private, since it may contain notes not intended or suitable for the audience. The presentation area is marked as not requiring input, while the navigation is marked as requiring input. The system can also derive this from the contained widgets. However, the author can manually mark this to override automatic behavior.

When a user starts this application from her mobile device, it connects to a server called dialog manager (DM). The DM initiates the transcoding process so that it can send a concrete user interface to the device. Since there is only one device, there is not much choice during transcoding. Each widget is rendered exactly once on the mobile device. Therefore the DM simply transcodes the user interface description to the format required by the mobile device. The device displays the user interface so that the user can interact, albeit constrained by device limits.

Now the user walks up to a public display to give her presentation. A location system tracks her, and emits an event as she enters the vicinity of the display. When the user started the application, the DM subscribed to such events. Now that an event tells it that the user's device context has changed, it starts the transcoding anew. This time it can use two devices: the public display and the handheld.

The first step of transcoding is choosing appropriate patterns for each device in this federation. Based on the characteristics of this particular UI, the transcoder decides to use a remote control pattern. Then it fragments the user interface description according to the pattern.

The handheld is assigned the control part of the remote control pattern. This means that the transcoding for the handheld registers a high preference for the input-intensive navigation, and a low preference for the other parts of the UI. The public display registers a high preference for rendering the presentation area and a lower preference for the navigation. The device characteristics of the public display do not match the privacy requirements of the note area, and the transcoder registers the public display's inability to render this part.

Finally, the transcoder resolves these preferences. The presentation part is assigned to the public display, while the navigation part is assigned to the handheld. The note area is also assigned to the handheld, because the transcoder ensures that each widget is rendered at least once, and the handheld is the only private device available.

After transcoding them to the device specific formats, the two user interface fragments are pushed to the devices. While the user interacts, the clients on the two devices are synchronized through the DM. When the user makes an input on one device, an event is sent to the DM, which updates all other

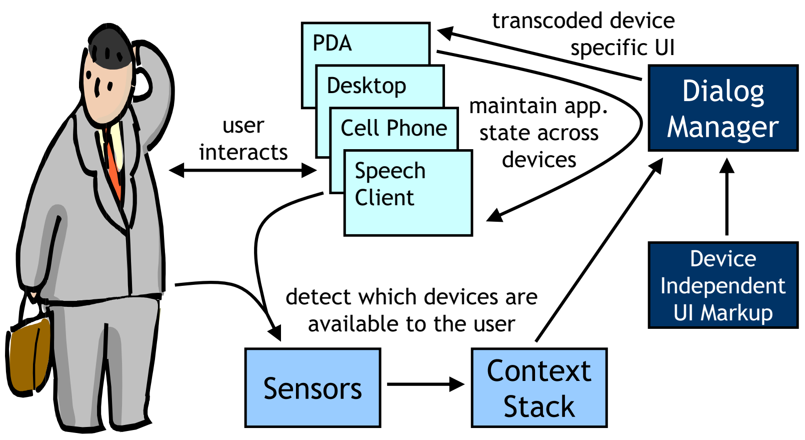
PDA
Desktop
Cell Phone
Speech Client
transcoded device specific UI
Dialog Manager
maintain app. state across devices
user interacts
Device Independent UI Markup
detect which devices are available to the user
Sensors
Context Stack

Figure 2. Architecture sketch

clients. For example, if the user presses the "next slide" button on the handheld, the application logic (running on the DM) receives this event and updates the presentation by switching to the next slide. Then the DM sends this update to the public display.

8. Future Work

Currently we are working on a user study to evaluate different proposed patterns for federated devices. With the results of this study we will try to increase the quality of our transcoder. We hope that the experiences from the study and our patterns will independently of our implementation also help other designers of federated user interfaces.

Our study will measure both performance (speed and error rates) and user acceptance of federated user interfaces. The first test case is a route planning software, of which three different version were developed. A pure handheld and a pure large screen version of the application were developed for comparison with the federated application. The second test case will try out privacy related patterns; a third test case has not yet been finalized.

9. Related Work

Our work was inspired by various projects about roaming, such as Teleporting [3]. To our knowledge, none of these deal with the concurrent use of multiple devices.

A number of applications using personal mobile devices together with larger fixed devices have been implemented as research prototypes. Examples are Pebbles [10] and Rekimoto's work on combining handhelds and whiteboards [13]. These projects have inspired us. However, these projects implemented a small number of applications for one fixed set of devices rather than arbitrary federations, and do not support roaming.

Also strongly related are a number of projects such as the iRoom [8]. These projects have built interactive office spaces with embedded large screens. Fox et al. have explored how handheld devices can be integrated into the iRoom. Special web pages can be prepared, which contain links that open the target of the link on another screen as opposed to the same screen. If such a web page is operated from a handheld device, it allows some form of remote control of other screens.

A project very similar in scope to ours is “Opportunistic Annexing” [12]. Their motivations match ours closely. The term Opportunistic Annexing refers to the process of associating public devices from the environment to augment a handheld user interface. We are currently not aware of actual implementations of the concept.

The concept of a web browser that interacts through multiple devices concurrently has been implemented before (e.g. [9]). However, to our knowledge none of these projects has tackled the authoring problem. Authoring web interfaces for these projects usually entails authoring a separate web page for each involved devices. Also there is no support for roaming.

There are projects which use mobile devices as input device replacements. For example, “Bring Your Own Device” uses a camera phone as a mouse [2]. The output capability of the mobile device is not used. This means that it is not possible to exchange private information with a public display.

A rather large body of work exists for the topic of multi-device authoring. The most well known example of a single authoring language is UIML [1]. However, most of the research on multi-device authoring has focused on rendering on desktop computers and adapting the same interface downwards to mobile devices with their limited resources. We are not aware of any work that deals with adapting upwards to interaction resource rich target devices, such as a federation with multiple screens.

Some single authoring schemes aim to create some kind of universal remote control (e.g. [11]). The term “remote control” is ambiguous. It can either mean “control your home appliances from work” or “control your TV while you are watching it”. The former requires the interface to control and provide feedback about all aspects of the controlled device, because it is not in sight of the user. The latter means that neither the control nor controlled device need to be useable stand-alone, as they can rely on being used together. Most universal remote control projects focus on the former, while for us the latter case is normal, for example when a PDA “remote controls” an associated screen.

An interesting project for the combined use of multiple displays is Gloss [7]. It allows users to reconfigure the positions of different kinds of displays, including PDAs, at runtime. The system is context aware with respect to their positions and orientations, and assembles them to act as one working space. However, at the current published state it does not seem to exploit different characteristics of different devices, but rather uses a disparate set of devices to display one desktop-like surface.

Using patterns for interaction design has been proposed before (e.g. [4]). However, such patterns are usually applicable only in the single device case. Adapting such patterns to federated devices is left to reader, if possible at all. We are not aware of a specialized pattern language for federated devices, although Chung et al. have included a pattern describing Rekimoto’s “Pick and Drop” in their collection of pre-patterns for ubiquitous computing [6].

References

[1] Marc Abrams, Constantinos Phanouriou, Alan L. Batongbacal, Stephen M. Williams, and Jonathan E. Shuster. UIML: an appliance-independent XML user interface language. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(11–16):1695–1708, May 1999.

[2] Rafael Ballagas, Michael Rohs, Jennifer G. Sheridan, and Jan Borchers. BYOD: Bring Your Own Device. In *Proceedings of the Workshop on Ubiquitous Display Environments, Ubicomp 2004*, 2004.

[3] Frazer Bennett, T. Richardson, and Andy Harter. Teleporting - Making Applications Mobile. In *Proceedings of 1994 Workshop on Mobile Computing Systems and Applications*, 1994.

[4] Jan Borchers. *A Pattern Approach to Interaction Design*. John Wiley & Sons Ltd, March 2001.

[5] Elmar Braun, Gerhard Austaller, Jussi Kangasharju, and Max Mühlhäuser. Accessing Web Applications with Multiple Context-Aware Devices. In Maristella Matera and Sara Comai, editors, *Engineering Advanced Web Applications*. Rinton Press, 2004.

[6] Eric S. Chung, Jason I. Hong, James Lin, Madhu K. Prabaker, James A. Landay, and Alan L. Liu. Development and evaluation of emerging design patterns for ubiquitous computing. In David Benyon, Paul Moody, Dan Gruen, and Irene McAra-McWilliam, editors, *Conference on Designing Interactive Systems*, pages 233–242. ACM, 2004.

[7] Joëlle Coutaz, Christophe Lachenal, and Sophie Dupuy-Chessa. Ontology for multi-surface interaction. In *Proceedings of Interact 2003*, 2003.

[8] Armando Fox, Brad Johanson, Pat Hanrahan, and Terry Winograd. Integrating information appliances into an interactive workspace. *IEEE Computer Graphics and Applications*, 20(3):54–65, May/June 2000.

[9] Jan Kleindienst, Ladislav Seredi, Pekka Kapanen, and Janne Bergman. Loosely-coupled approach towards multi-modal browsing. *Universal Access in the Information Society*, 2(2):173–188, June 2003.

[10] Brad A. Myers. Using Handhelds and PCs Together. *Commun. ACM*, 44(11):34–41, 2001.

[11] Jeffrey Nichols, Brad A. Myers, Michael Higgins, Joseph Hughes, Thomas K. Harris, Rosenfeld Rosenfeld, and Mathilde Pignol. Generating remote control interfaces for complex appliances. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, Papers: infrastructure for ubicomp, pages 161–170, 2002.

[12] Jeffrey S. Pierce, Heather Mahaney, and Gregory Abowd. Opportunistic annexing for handheld devices: Opportunities and challenges. Technical report, Georgia Tech, 2003.

[13] Jun Rekimoto. A Multiple Device Approach for Supporting Whiteboard-Based Interactions. In *Proceedings of ACM CHI 98 Conference on Human Factors in Computing Systems*, pages 344–351, 1998.