Single Authoring for Multi-Device Interfaces

Elmar Braun, Andreas Hartl, Jussi Kangasharju, Max Mühlhäuser

Telecooperation Group, Department of Computer Science
Darmstadt University of Technology
{elmar, andreas, jussi, max}@tk.informatik.tu-darmstadt.de

Abstract. Users in a ubiquitous computing world will interact with several devices simultaneously. This multi-device interaction paradigm presents significant challenges to both authoring and runtime application rendering. In this paper we present our work on spanning user interfaces across federated devices. Our first contribution is a single-authoring language for multi-device interfaces which allows authors to specify how their applications should be rendered. Our second contribution is a mechanism for selecting the appropriate devices and rendering the user interface on them.

1.   INTRODUCTION

The vision of ubiquitous computing is bringing about the "third age" of user-computer relation. First came mainframes, single computers shared by many users. PCs have changed that to a one-to-one relation between the user and the computer. Ubiquitous computing promises to extend this, by having each user use many computers. Indeed the use of several mobile devices (PDA, cell phone) in addition to the desktop PC has become commonplace.

To illustrate the need for user interfaces to span several devices, consider the following scenario. Alice works in an office where work takes place using all means of interaction available. Once Alice decides to leave her office, she can continue interacting through a voice-only headset. In her car, the headset associates with the screen of the car navigation system, to present a multimodal interface. At the client site, Alice's mobile device again associates with a public screen provided by the office infrastructure.

Isolated use of a mobile device becomes an extreme case that occurs only in few, non-augmented spaces, such as on the way to and from the car. The normal use case is that the UI spans multiple devices, which work together to render the UI through several channels and modalities concurrently. At conceptual level, this is an MVC (model-view-controller) pattern, with multiple views and controllers that happen to run on different devices. For simplicity, we usually refer to the set of associated devices as *federated devices* or a *device federation*.

In this paper we present our on-going work on spanning user interfaces across device federations. Our approach has several advantages. First and foremost, it allows the UI to escape the limitations of a single mobile device. In fact, we consider the device limitations as exerting "pressure" on the UI. Traditionally, mobile UIs have either tried to "cram" as much UI into the small "UI container" of a typical mobile device, or, at the other end of the spectrum, tried to reduce the "amount of UI" in order to conform to the device limitations. Our approach seeks to find a larger container in the surrounding infrastructure, to which the UI can "overflow".

The second advantage of our approach is that the pressure, which current approaches exert on mobile devices, limits the degree to which they can be minimized. Modern mobile devices cannot be minimized much further, as their isolated usage model means that they alone have to provide all features necessary for their use. A device that is usually used in conjunction with other interactive devices, can be much lighter, as it only needs to provide minimal, last resort capabilities for isolated use. One example of such a device is the voice-only *Talking Assistant* (TA) headset [Aitenbichler 02].

This paper is organized as follows. In Section 2, we present the challenges of authoring content for multi-device interfaces and present our specialized single-authoring language which addresses these challenges. Section 3 describes our runtime environment for rendering and testing multi-device interfaces, as well as our experiences in implementing it. In Section 4 we review related work. Finally, Section 5 concludes the paper by presenting a summary and directions for future work.

2. AUTHORING MULTI-DEVICE INTERFACES

The first challenge in creating a UI that can be rendered on a device federation is the authoring of the UI. New authoring techniques are needed because some properties of multi-device interfaces go beyond the capabilities of existing authoring techniques.

2.1. Challenges in Authoring Multi-Device UIs

There has already been some research on multi-device interfaces (see Section 4) and their evaluation. However, this work has concentrated on manually programmed UIs and such handcrafting of UIs is typically only an option for specialized applications with a fixed set of devices known in advance.
Consider the scenario from the introduction. In the office, we can use whatever devices are available. When moving to and from the car, we are briefly restricted to voice-only interaction on a TA-like device. Although both in the car and at the client site we have a federation between the headset and a screen, the screen at the office is much larger and may allow touch interaction. So, the application in our example is rendered consecutively on four different federations of devices, and might use entirely different devices next time it runs, depending on the context of the user.
While it may be feasible to create manually a UI for any single federation, the effort to cover all feasible federations with handcrafted UIs would be too large. This problem is essentially that of *single authoring*. In other words, what is needed is a single, device-independent user interface description, which can be mapped to a good concrete UI for each feasible target device. The goal for single authoring is to avoid the effort of manual authoring for each target device. In the case of federated devices, the set of target devices is not simply the set of possible device types, but rather the set of all feasible federations of these devices.
However, we have found existing single authoring languages to be insufficient for the needs of multi-device interfaces. Most single authoring languages are intended for building multi-channel applications (terminology from [Maes 00]). These are applications that are able to employ different channels, but only *one channel at a time*. Since we render UIs through multiple channels *concurrently*, we are faced with the question which widgets to render on which channel. Depending on the characteristics of the channels and of the UI, it may make sense to render some widgets on multiple channels concurrently to achieve multimodality, while in other cases rendering once is sufficient.

2.2. A Single-Authoring Language for Multi-Device Interfaces

A single-authoring language for multi-device UIs shares many basic requirements with the single device case, like for example a device independent representation of widgets (e.g. "select one" instead of "radio button"). We decided to base our experiments on existing work and use subsets of XForms [XForms] and XHTML [XHTML]. We have extended them by adding our own tags and attributes. XHTML has some advantages over other languages, such

as UIML [Abrams 99]. It is well-known, there are many available tools for processing it, and transcoding to real HTML is simple, which allows us to use standard web browsers as clients (see Section 3.4).

In a single-channel UI, each element of the UI has to be rendered exactly once. Rendering more than once on the same channel has little benefit, while rendering a widget not at all would make the UI inoperable. The two main problems with multiple channels and devices is choosing on which channel to render a widget, *and* deciding whether to render it on multiple channels. The task of our authoring language is to provide hints to the runtime that will make such decisions (see Section 3.2). We assign an importance attribute, or weight, ranging from 0 to 1, to each widget (e.g. `<xforms:input value="0.5">`). When rendering, we use this attribute to place more important widgets first on devices that are easier to interact with.

In some cases a single importance value is not sufficient. Consider a federation of a PDA with a wall-mounted display that has no means of input. In this case, the large display is to be preferred for displaying text, but the PDA is the only means of input. Hence, we have decided to provide the possibility for specifying the importance attributes separately for input and output (e.g. `<xforms:trigger valueIn="1" valueOut="0">`).

The weight is also used in determining whether to render a widget on multiple channels concurrently. This decision also depends on the available devices. The default behavior is to render widgets concurrently only on different modalities to achieve multimodality. But we provide an attribute to override the default behavior: `<xforms:input replicate="…">`. Setting this attribute to `once` prevents duplication, whereas `always` tries to render a widget multiple times even on equal modalities.

In ubiquitous computing environments it is normal to associate publicly available devices. Hence, privacy is an issue. Private data should not be displayed on a large public display, even if it was the best available device. Widgets can be marked as private (never shown publicly), shared (made public if possible), or non-sensitive (no preference). This attribute can be set at runtime; elements default to private, but can be toggled to shared by a switch rendered on a private device. In addition we have two levels of "public": one that uses any public device, and one that uses public devices only within a trusted environment like a private office.

We also provide a facility for grouping widgets, in order to avoid creating unusable UIs by scattering widgets across devices in a chaotic manner. We use the XForms `<group>` tag with custom attributes for this purpose. This tag indicates that the grouped widgets should be rendered as a single unit. All of the above attributes apply also to the `<group>` tag.

3. ADAPTING USER INTERFACES TO MULTIPLE DEVICES AT RUNTIME

Rendering the interfaces created by our single-authoring language consists of a number of steps. First, the runtime environment needs to discover which devices are in the user's vicinity. Then it must decide which devices will show which parts of the UI, and transcode the device independent UI into a format that the devices can process. Finally, while the application is running, we need to synchronize input and output of the federated devices.

3.1. Discovering Devices

In order to assemble a device federation, all available devices near the user need to be discovered first. One of our main goals is that, in the spirit of invisible computing, interaction should take place through the devices that seem most intuitive to the user, rather than having the user worry about selecting the devices. Therefore it is necessary to detect automatically

which devices are already at the focus of the user's attention, or which can be brought to the user's attention with minimal disrupting effect.

Context awareness is one of the main points of MUNDO [Hartl 02], the framework project for all ubiquitous computing research at our group. Hence, we can reuse existing technologies, such as the location tracking of the TA [Aitenbichler 03]. We have built a positioning system around the TA which allows us to determine the position of a user in the room, as well as the direction the user is looking. In combination with a world model, i.e., a database of device locations, we can find out which device the user is looking at. For example, a user can associate a display simply by looking at it.

In addition we developed another system, which consists of infrared-emitting badges, similar to the Active Badge [Want 92], which locate in which room their wearer is. This alone is insufficient for an attentive UI, since a display might be in the same room but behind a user. We have complemented the basic badge architecture with *tags*, which we can attach to devices. Tags send a short-range infrared signal, which badges receive only if they are within approximately 1.2 meters *and* are facing towards the tag. This cone of reception corresponds roughly to the area from which a desktop screen can be read.

For mobile devices that are carried by the user, a simple heuristic can determine whether they enjoy their owner's attention. Whenever the device is switched on, it can be assumed that its user looks at it. This works fairly well since most mobile devices go to standby rather quickly when not used. A number of more sophisticated schemes to determine whether a device is in use (e.g. [Hinckley 00]) exist also. These could be integrated into our system in the future.

3.2.   Picking the Right Devices

A prerequisite to making a good choice of devices is to know the capabilities of the available devices as well as the requirements of the application. For example, an application that wants to render a complex spreadsheet requires a screen large enough to fit the spreadsheet. Furthermore, the user's preferences should be taken into account.

We currently use the CC/PP [CCPP] User Agent Profile Specification (UAPROF) to model hardware capabilities. Since CC/PP is rather complex, yet not well suited to describe arbitrary devices because UAPROF seems to be intended primarily for cell phones, we intend to replace CC/PP with a language of our own in the near future.

We can choose which devices to use for rendering, by combining the profiles of all discovered devices with the requirements of the application. In the case of modality-dependent user interface elements, it is not uncommon that only one device can render them (e.g. scribble input is only possible on touch screens). All other elements need to be placed across the available devices sensibly.

For example, how should an audio player, which consists of control buttons (play, pause, etc.), a play list, and a long text about the artist, be distributed to a large screen and a PDA? A sensible choice would be to distribute the buttons to the easy to touch PDA, while the larger play list and artist info are displayed on the screen (see Figure 1). For deciding about the best device for a given purpose, we need to apply heuristics compiled from HCI principles. We are currently compiling a list of useful principles on which to base our heuristics.

These heuristics will be one of the main foci of our future work. A "perfect" set of heuristics would be able to generate a good quality UI even if given no hints whatsoever, and therefore make our extensions to XHTML superfluous. Since we were not yet able to achieve this, we introduced said extensions. Further investigations will need to show whether they are sufficient, too simplistic, or too verbose.
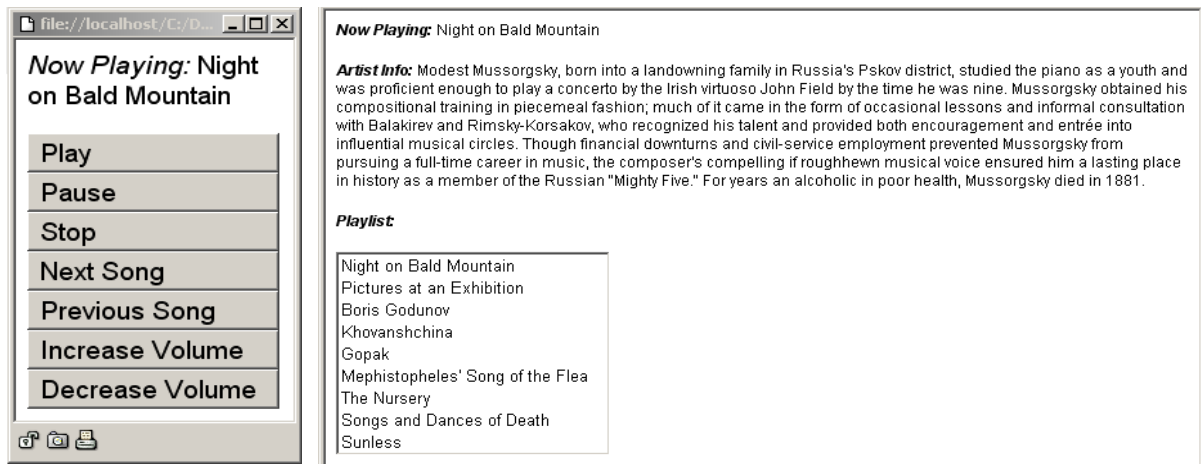
Figure 1: PDA part (left) and large screen part (right) of an audio player application

3.3. Tailoring the Interface to Devices

Once we have selected the devices, we must generate a concrete UI for their capabilities. This is a straightforward process as long as there is a one-to-one mapping from abstract widget to concrete widget. This may not be the case, e.g. because the target device provides several different realizations of one abstract widget. Our approach to this problem is described in [Hartl 03] for the case of transcoding to a voice-only interface.

3.4. Rendering for a Specific Device

After we have distributed the different elements of the UI to the devices, and provided them with concrete UIs, they can render their portion of the UI. For that purpose the devices need client software which performs the rendering. For our first prototype, we have decided to use a web browser as the renderer, because most of our target devices have a browser available.

For our desktop and large screen client we implemented a wrapper around the Internet Explorer. In this case, transcoding into the native format simply means transcoding to HTML with form elements as input widgets. The browser performs rendering and interaction and a wrapper is responsible for starting the browser when a user steps up to a screen. The wrapper also closes the browser when the user leaves the screen and communicates with the other federated devices (see below). A speech client that wraps IBM ViaVoice in a similar fashion is currently under development.

Once the UI is rendered on multiple federated devices, these devices need to be coordinated. When a user makes an input, she should get some feedback that the input has been recognized correctly. Input and output of the feedback need not occur through the same channel. If a user makes speech input through her headset, an associated display should update the according input field with the recognized value.

For the coordination, all clients immediately send all input they receive to a server which we call *dialog manager* (DM). The DM forwards this input to all other clients, which update their view accordingly. In other words, the DM is the model in an MVC pattern. The DM is also where the actual application connects to the UI. In our case, the application is a Java class that receives events about input from the DM, and can make changes to the model, causing the DM to send out the appropriate updates to all clients (see Figure 2).
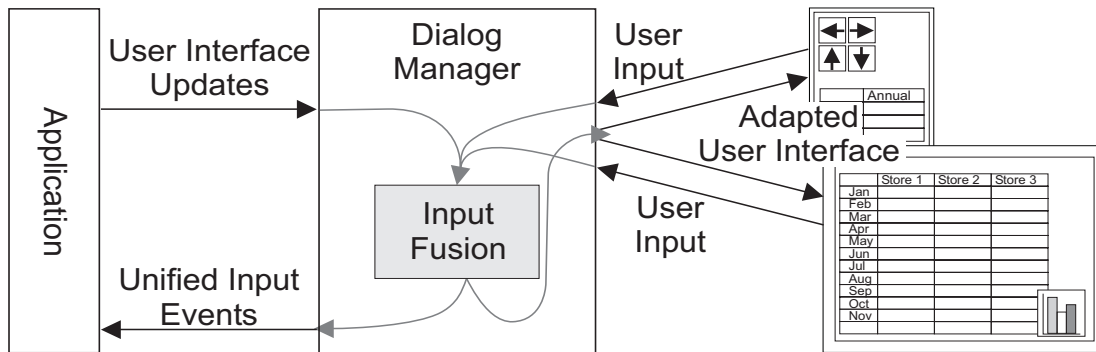
Figure 2: The dialog manager is responsible for synchronizing input and output of all devices in the federation. Arrows indicate the data flow in this figure.

4. RELATED WORK

The problem of single authoring in a device independent manner has received much attention in the past, both in the form of device independent widget toolkits [Gellersen 95] and markup languages (e.g. [Abrams 99]). In [Eisenstein 01], a multi-step transformation process is used to accommodate the requirements of a device. A UI may iteratively be split into several parts, which may be presented simultaneously or separately, depending on the available screen real estate. None of the existing work considers the case of rendering on *multiple devices* concurrently which is the focus of our work. Our work is complementary to existing research; once we have determined which widgets to render on which devices, the existing approaches could be used to render the widgets on the particular devices.

Another approach to single authoring is model-based design. The user interface is specified as a number of tasks and their relations, rather than abstract or concrete widgets. The approaches which we are aware of (e.g. [Mori 03]) generate an intermediate abstract widget representation from the task model. We therefore decided to approach our problem with abstract widgets first, and possibly investigate later whether model-based approaches can be used to generate UI descriptions in our language.

The use of portable and stationary devices together has been explored before (e.g. [Rekimoto 98, Myers 01]). However, such research usually focused on one fixed combination of devices. Our work provides single-authoring for an arbitrary set of devices, determined at run-time. Bandelloni and Paternò investigate model-based single authoring for two devices, and allow runtime migration from one device to another [Bandelloni 04]. This is achieved by splitting the UI into a migration visualization part and a non-migrating control part. They refer to this as *partial migration*, while our work, which does not require this separation and allows arbitrary parts of the UI to migrate, matches what they refer to as *mixed migration*.

The effects that the concurrent use of multiple displays has on user performance and satisfaction have been evaluated by Tan and Czerwinski [Tan 03]. This study deals with equally sized screens, while our motivation is mostly combinations of mobile handhelds with larger screens, but its results are still valuable for our considerations whether to split between two screens or not.

Some projects, like the i-Land [Tandler 04] or the Stanford iRoom [Fox 00], have dealt with user interfaces spanning multiple computers in technology-rich work spaces. Such rooms have the benefit that many devices can be assumed to be present and that the room infrastructure usually does not change. This allows some behavior to be hardwired. In contrast, we target dynamic environments, where all the behavior of the system must be

automatically generated. Single authoring has also been explored in these projects (e.g. for the iRoom [Ponnekanti 01]). Despite their roots in heavily multi-device environments, most efforts focus on single authoring UIs rendered on one device at a time, usually to generate some form of remote control on a handheld.

Another feature of the iRoom is Multibrowsing. This allows some form of remote control of a browser on a larger screen from a handheld, and is similar to our testing platform, which consists of several synchronized browsers on different devices. The development of a multimodal browser by synchronizing browsers on multiple devices has been described in [Kleindienst 03]. Pages for this browser are authored by manually creating multiple versions of the same page in HTML, WML, and VoiceXML. WebSplitter [Han 00] also synchronizes multiple browsers, but does not support non-visual modalities.

5. SUMMARY

We have presented a mechanism for distributing a user interface across a federation of devices. Using our methods, applications can escape the boundaries of a single device, yet still benefit from single authoring and well-known technologies such as XForms and XHTML.

We showed how to form federations of devices dynamically at runtime. We use various methods of detecting nearby devices and whether they have the user's attention. Once a federation has been formed, we distribute a concrete realization of a part of the UI to each device. We have also presented a way combining the data coming from several devices into one model using a dialog manager.

REFERENCES

[Abrams 99] M. Abrams, C. Phanouriou, A. L. Batongbacal, S. M.Williams, and J. E. Shuster. *UIML: an appliance-independent XML user interface language.* Computer Networks (Amsterdam, Netherlands: 1999), 31(11–16):1695–1708, May 1999.

[Aitenbichler 03] E. Aitenbichler and M. Mühlhäuser. *An IR Local Positioning System for Smart Items and Devices.* 3rd International Workshop on Smart Appliances and Wearable Computing, pp. 334–339, 2003.

[Aitenbichler 02] E. Aitenbichler and M. Mühlhäuser. *The Talking Assistant Headset: A Novel Terminal for Ubiquitous Computing.* Technical Report TK-02/02, Fachbereich Informatik, TU Darmstadt, 2002.

[Bandelloni 04] R. Bandelloni and F. Paternò. *Flexible Interface Migration.* In Proceedings of ACM International Conference on Intelligent User Interfaces 2004, pp. 148-157, 2004.

[CCPP] G. Klyne, F. Reynolds, C. Woodrow, H. Ohto, J. Hjelm, M. H. Butler, and L. Tran. *W3C recommendation: Composite capability/preference profiles (CC/PP): Structure and vocabularies 1.0.* http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/, Jan. 2004.

[Eisenstein 01] J. Eisenstein, J. Vanderdonckt, and A. Puerta. *Applying Model-Based Techniques to the Development of UIs of Mobile Computers.* In Proc. of Intelligent User Interfaces, pp. 69–76, Jan. 2001.

[Fox 00] A. Fox, B. Johanson, P. Hanrahan, and T. Winograd. *Integrating Information Appliances into an Interactive Workspace.* IEEE Computer Graphics and Applications, 20(3):54–65, May/June 2000.

[Gellersen 95] H.-W. Gellersen. *Modality Abstraction: Capturing Logical Interaction Design as Abstraction from "User Interfaces for All"*. In Proceedings of the 1st ERCIM Workshop on 'User Interfaces for All'. ERCIM, 1995.

[Han 00] R. Han, V. Perret, and M. Naghshineh. *WebSplitter: A Unified XML Framework for Multi-Device Collaborative Web Browsing*. In Proceedings of ACM CSCW'00 Conference on Computer-Supported Cooperative Work, pp. 221–230, 2000.

[Hartl 03] A. Hartl. *A Widget Based Approach for Creating Voice Applications*. In Proceedings of MobileHCI, Udine, Italy, 2003.

[Hartl 02] A. Hartl, E. Aitenbichler, G. Austaller, A. Heinemann, T. Limberger, E. Braun, and M. Mühlhäuser. *Engineering Multimedia-Aware Personalized Ubiquitous Services*. In IEEE Fourth International Symposium on Multimedia Software Engineering (MSE 2002), pp. 344–351, Newport Beach, USA, 2002.

[Hinckley 00] K. Hinckley, J. S. Pierce, M. Sinclair, and E. Horvitz. *Sensing Techniques for Mobile Interaction*. In Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology (UIST 2000), pp. 91–100, 2000.

[Kleindienst 03] J. Kleindienst, L. Seredi, P. Kapanen, and J. Bergman. *Loosely-coupled approach towards multi-modal browsing*. Universal Access in the Information Society, 2(2):173–188, June 2003.

[Maes 00] S. H. Maes and T. V. Raman. *Position paper for the W3C/WAP Workshop on the Multimodal Web*. http://www.w3.org/2000/09/Papers/IBM.html, Sept. 2000.

[Mori 03] G. Mori, F. Paternò, and C. Santoro. *Tool Support for Designing Nomadic Applications*. In Proceedings of ACM International Conference on Intelligent User Interfaces 2003, pp. 141-148, 2003.

[Myers 01] B. A. Myers. *Using Handhelds and PCs Together*. Commun. ACM, 44(11):34–41, 2001.

[Ponnekanti 01] S. R. Ponnekanti, B. Lee, A. Fox, P. Hanrahan, and T. Winograd. *ICrafter: A Service Framework for Ubiquitous Computing Environments*. Lecture Notes in Computer Science, 2201, 2001.

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

[Tan 03] D. S. Tan and M. Czerwinski. Effects of Visual Separation and Physical Discontinuities when Distributing Information across Multiple Displays. In Proceedings of OZCHI'03 (Australian Computer Human Interaction Conference), pp. 184-191, 2003.

[Tandler 04] P. Tandler. *The BEACH Application Model and Software Framework for Synchronous Collaboration in Ubiquitous Computing Environments*. The Journal of Systems and Software, 69(3):267–296, Jan. 2004.

[Want 92] R. Want, A. Hopper, V. Falcão, and J. Gibbons. *The Active Badge Location System*. ACM Transactions on Information Systems (TOIS), 10(1):91–102, 1992.

[XForms] M. Dubinko, L. L. Klotzs, and T. V. Raman. *XForms 1.0*. http://www.w3.org/TR/2003/REC-xforms-20031014/, Oct. 2003.

[XHTML] J. Axelsson, B. Epperson, M. Ishikawa, S. McCarron, A. Navarro, and S. Pemberton. *XHTML 2.0*. http://www.w3.org/TR/2003/WD-xhtml2-20030506, May 2003.