Lessons Learned from the Design and Implementation of Distributed Post-WIMP User Interfaces

Thomas Seifried[1], Hans-Christian Jetter[2], Michael Haller[1], Harald Reiterer[2]

[1]Upper Austria University of Applied Sciences, Media Interaction Lab, Hagenberg, Austria
[2]University of Konstanz, Human-Computer Interaction Group, Konstanz, Germany

Abstract Creating novel user interfaces that are "natural" and distributed is challenging for designers and developers. "Natural" interaction techniques are barely standardized and in combination with distributed UIs additional technical difficulties arise. In this paper we present the lessons we have learned in developing several natural and distributed user interfaces and propose design patterns to support development of such applications.

Introduction

In the recent years, the ever-increasing miniaturization of interactive products with great computational power and network connectivity has resulted in a proliferation of computing into almost every facet of our physical and social world. This process has turned many aspects of Mark Weiser's vision of ubiquitous computing into a part of our daily life. However, as HCI researchers, we consider two essential challenges of Ubicomp as still unmet: creating novel user interfaces that are natural and distributed.

Fig. 1. Interactive spaces based on post-WIMP DUIs. (a) NiCE Discussion Room [5], (b) DeskPiles [2] and (c) Facet-Streams [10].

Our understanding of Distributed User Interfaces (DUI) is based on Melchior et al. [11]: a DUI is a UI with the ability to distribute parts or whole of its components

across multiple monitors, devices, platforms, displays and/or users. For our research, this ability is essential to realize interactive spaces [16] in which multiple interactive surfaces and devices act as one distributed UI for co-located collaboration (Fig. 1). In these spaces we try to achieve a "natural" interaction, i.e. the UI is perceived as something unobtrusive or even invisible that does not require the users' continuous attention or a great deal of cognitive resources. A well-proven approach to achieve this is the visual model-world interface for "direct manipulation", in which a tight coupling of input and output languages narrows the gulfs of execution and evaluation [7]. While direct manipulation originates from 1980s desktop computing, its principles are also the foundation of novel post-WIMP (post-"Windows Icons Menu Pointing") or reality-based UIs [8]: Their interaction styles (e.g. tangible, multi-touch or paper-based UIs) "draw strength by building on users' pre-existing knowledge of the everyday, non-digital world to a much greater extent than before." Users can apply the full breadth of their natural, non-digital skills, e.g. the bimanual manipulation of objects, handwriting or their awareness of objects or persons in their physical and social environment.

Our two research groups have designed and implemented a great variety of such post-WIMP distributed UIs for co-located collaboration in augmented meeting rooms [5], on tabletops for scientific discussion [2] or for collaborative product search [10] (Fig. 1). Based on these experiences, we have to conclude that the combination of natural and distributed UIs poses a particular hard challenge to UI designers and developers [9]. As discussed by Shaer and Jacob, the typical challenges that creators of natural UIs face are the lack of appropriate interaction abstractions, the shortcomings of current user interface software tools to address continuous and parallel interactions, as well as the excessive effort required to integrate novel input and output technologies [14]. The distribution of natural interactions across device and display boundaries adds greatly to this complexity.

In the following, we summarize our "lessons learned" to share them with DUI researchers and practitioners by extracting two design patterns (DP) and an anti-pattern (AP). These three patterns address both sides of UI creation: interaction design patterns [1] and software design patterns. All were tested extensively during our projects. While the first two patterns have become a part of our open source software framework ZOIL that facilitates DUI implementation [17], the anti-pattern was implemented, tested and discarded as ineffective. We conclude with a brief summary of our findings and formulate research questions for future work.

Design Patterns for Post-WIMP DUIs

To understand the origin of our patterns, it is important to notice the commonalities of the projects from which they are derived: All of them are aimed at creating interactive spaces for co-located collaboration of multiple users. As shared surfaces we either use large Anoto-pen enabled front-projected surfaces [5] or smaller

vision-based multi-touch enabled tabletops (e.g. Microsoft Surface) [9]. Personal and mobile surfaces are realized using sheets of Anoto paper & laptops [5] or tablet PCs [2]. To achieve a natural post-WIMP interaction, the dominant input modalities throughout the projects are multi-touch and/or Anoto pens. Furthermore the design follows a fundamental principle of natural UIs: "the content is the interface" [6]. This means, that the amount of administrative UI controls known from WIMP (e.g. menus, window bars, tool bars) is minimized so that the content objects themselves become the first-class citizen of the UI. This natural provision of content for direct manipulation also has implications on the flexibility of interaction. By abandoning traditional page- or dialog-oriented sequences of interaction (e.g. typical Web applications), users can act directly and flexibly on the objects of the task domain. Apart from multi-touch and pen-based manipulations and gestures, tangible props such as physical tool palettes [5] or glass tokens for query formulation support users in their tasks [10].

DP1: Real-Time Distribution of a Zoomable Workspace

A prerequisite for any kind of collaboration is a shared workspace accessible to all users. As a first interaction design pattern, we therefore suggest the use of a shared visual workspace that uses a 2D virtual plane containing all necessary functionality and content of the application domain as visual objects for direct manipulation. All user changes to the location, orientation, size, annotation or nature of these objects are immediately executed and visualized in real-time. The workspace serves as a model-world representation of the application domain that shares an essential property with the real world: actions on objects lead to immediate feedback and persistent results. Thus, the workspace resembles a physical whiteboard for natural, co-located and synchronous collaboration [5]. We extend this pattern further with a Zoomable User Interface (ZUI). ZUIs largely increase the amount of accessible objects because the workspace is not limited to the visible screen size and becomes virtually infinite in size and resolution. Nevertheless ZUIs still maintain a natural feel during navigation as they tap into our natural spatial and geographic ways of thinking [12]. Thereby "semantic zooming" is employed and geometric growth in display space is also used to render more and semantically different content and functionality. Ideally ZUIs can thereby "replace the browser, the desktop metaphor, and the traditional operating system. Applications per se disappear" [13]. Most importantly, when put on a central server to make it accessible from different clients, such a shared ZUI enables many scenarios of real-time distribution. Every ZUI client can access the local or remote ZUI server to render an arbitrary section of the shared ZUI at an arbitrary zoom level. Thus each client acts as a kind of camera into the shared workspace that users can control using zooming and panning operations. This enables many distribution modes:

1. By running several instances of a ZUI client on one PC, different views of the real-time synchronized workspace can be displayed simultaneously, e.g. for distributing the workspace to multiple windows or multiple monitors to create an overview and detail solution.
2. When using multiple devices each device can run one or several clients that connect to the central server, so that different devices can access and visualize the same shared workspace. Thus the physical boundaries of devices can be overcome to achieve a cross-device distribution. This can be used to provide multiple access points, e.g. several co-located PCs with large touch-enabled vertical or horizontal displays at a local or even a remote site. Depending on the use case, a device's view onto the shared workspace can be used to navigate independently, but it can also be tightly coupled to another device. For example a handheld device can act as a standalone personal navigation device or as a zoomed out overview of the details currently displayed on a nearby tabletop.
3. The same mechanisms enable the distribution of the workspace to multiple users: By introducing personal devices (e.g. smart phones, laptops or tablet PCs) that run a ZUI client, a distribution to multiple users becomes possible (e.g. to several users around a tabletop each carrying a tablet PC for annotation [2]).

We have made extensive use of this interaction pattern in [2] and [10]. We share more details about our software design and implementation choices for this interaction pattern in the following.

DP2: Distributed ViewModel

Synchronizing the UI between distributed instances of an application is a typical use-case of DUIs, but the development of such a system can be challenging especially when designing novel interfaces. The Distributed ViewModel aims to simplify the development of such Post-WIMP DUIs by introducing an additional abstraction layer between UI and networking logic.

Content and interaction are often much closer to each other in Post-WIMP UIs than it used to be in traditional user interfaces. This motivates developers to bring those two properties closer together in the software design. Content and basic view information, such as position and size of a view, can be easily modeled with standard design patterns, but the interaction itself cannot. Interaction with post-WIMP user interfaces is not as standardized as in WIMP interfaces, therefore UI developers still need a lot more freedom to design and test new interaction techniques. But in contrast to non-distributed UIs, designing interaction for DUIs lacks many tools and design patterns and still requires much know-how about the technical background. For example, a DUI developer needs to know how to distribute the UI onto other machines. But network synchronization with all its issues is a very complex topic and a UI developer should not need to worry much about it.

The concept of a Distributed ViewModel tries to address those problems by providing a network-synchronized model of a view to ease development of a shared UI. It provides an additional abstraction layer that contains the content of a view as well as view-dependent properties. The distributed view model is based on the Model-View-View-Model (MVVM) design pattern [15]. In the MVVM pattern the data model of an application is separated from its view, similar to the Model-View-Controller (MVC) pattern. In contrast to MVC, MVVM provides an additional abstraction layer, the so-called “ViewModel” which is an abstract description of the view. The “ViewModel” can also be seen as a "Model of the View" containing only view-related information and logic. This allows an UI designer to mainly focus on UI design and it provides a clean interface to the non-UI parts of the application. The Distributed ViewModel pattern, as depicted in Fig. 2, facilitates this clean interface to provide a transparent distribution of view-related properties which are defined in the ViewModel. The Distributed ViewModel is much like the ViewModel as it contains the same information, but in contrast its contents and structure are already prepared for network synchronization. All information stored in a Distributed ViewModel is automatically serialized and synchronized with all other connected instances of the application or UI. In practice, the ViewModel can often be completely replaced by a Distributed ViewModel, if data types used in the view are compatible with network synchronization.

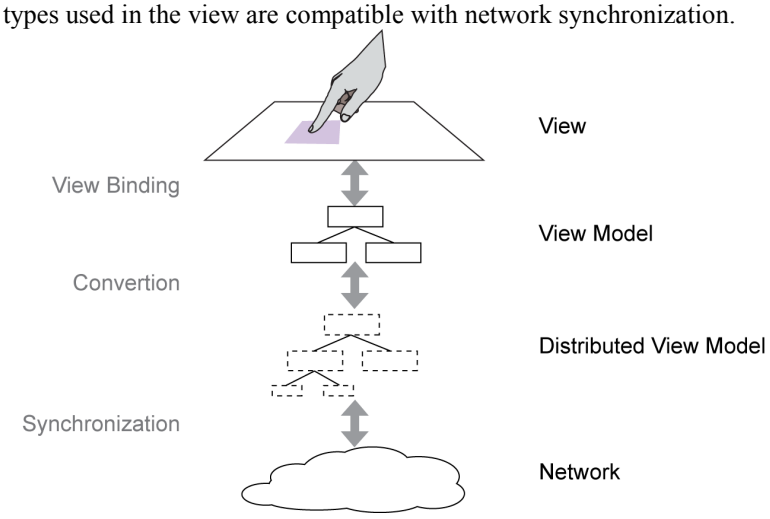

Fig. 2. Concept of Distributed ViewModels.

In difference to the original MVVM design pattern, the Distributed ViewModel pattern is designed for transparent synchronization of ViewModels among all connected instances. Thereby the Distributed ViewModels are handled as “network shared objects” which update all other networked instances of the same object if a property changes. The update mechanism makes use of a change notification system within the view and ViewModel. If a property of a view and consequently of a

ViewModel is modified, the ViewModel fires an event allowing other objects, such as the Distributed ViewModel, to be notified. Consequently, distributed instances of the Distributed ViewModel can be updated accordingly. It is important to note, that the distributed update of these objects needs to take care of concurrency issues that might arise if two instances of the same objects are changed concurrently.

The update mechanism of the Distributed ViewModel can be developed in a nearly transparent way. In our DUI applications we provided a base class which hid the network synchronization from the UI development. In our implementation we have used an object database that provides a "transparent persistency" mechanism as back-end [17]. Hence a UI developer never came in direct touch with networking issues.

AP1: Input Forwarding (Anti-Pattern)

Not every software design that has been tested was successful. Forwarding of input events from input devices such as the mouse, stylus or keyboard failed. The motivation behind input forwarding is to distribute interaction techniques by simply forwarding the raw input received from the input devices to all other instances of the DUI. For example a touch input event on display A is forwarded onto display B and is processed by both UI instances in the same way. This design is based on the idea that input from a remote UI instance is handled in the same way as local input. Therefore, new interaction techniques on a DUI would be very simple to implement, because the UI developer does not need to care about network synchronization problems at all. The underlying assumption behind this design is that the CPU is a deterministic and standardized system. Hence, the same input always results in the same output. This would also mean that the underlying visualization of the controls does not need to be distributed, because the operations on those would result in the same result on every instance.

This pattern relies on a server-client architecture. All input events are sent to a server on which they are synchronized and ordered. This solves many concurrency problems because every instance gets the same events in the same order. Additionally, since the resolution of the UI may vary on different devices, all input data containing position information (e.g.: mouse-pointer coordinates) need to be normalized before distribution. Accordingly the receiving instance needs to map those normalized input data in its own coordinate space. The UI controls are simply scaled on such systems, therefore input on a certain control on display A would be also on the same control on display B, even when the resolution is different.

Although this design has been successfully used on single-display, multi-user applications [4] this design failed for DUIs. The state of the UI on distributed instances of the system did diverge after a short time. This was caused by three problems that may occur in such a system:

1. Input coordinates have been normalized by using floating-point values. Since floating-point numbers are imprecise, results of de-normalization on DUIs using displays with different resolutions always contained a small rounding error. This small rounding error can be enough that a button is clicked on one instance and on another it is not.
2. Even when every instance of the DUI used the same resolution, interactions based on relative motion (e.g.: translating a UI element beneath a moving touch point relative to its last position) caused divergence. The GUI system used for rendering also used floating-point numbers to position UI elements. In contrast to our assumption floating-point calculations do not always have the same result on different computers [3].
3. This design only works if the UI is exactly the same all the time. Even small differences in the layout may result in divergence of the instances.

Therefore, we suggest that input forwarding should only be used if no floating-point values are involved and the layout and resolution of the UI is always exactly the same.

Conclusion

In this paper we have summarized our experiences or "lessons learned" from developing several "natural" and distributed UIs. The applications we have developed are all based on either a shared workspace or shared view; therefore our results may be limited to those types of DUIs. Still we believe that some aspects of our results are generalizable and also relevant for other types of DUIs.
In the combination of both, relatively new, types of UIs a set of new problems arises. Our software and interaction design patterns are the results of our first attempts to solve these problems. However, not all of our attempts were successful. At this early stage of post-WIMP DUI research we therefore think that it is important to also report about these failed approaches, so that other developers and researchers can avoid making the same mistakes. In the light of the novelty and complexity of the field, several new questions for future research are raised:

1. Applying our suggested design patterns requires skills in software development. However, today's UIs are usually created by designers using visual UI design tools. How do we incorporate the necessary software patterns into a new generation of design tools for post-WIMP DUIs?
2. Some of the functionality of post-WIMP DUIs cannot be mapped to continuous direct manipulations of objects, e.g. commands such as undo or redo. Future design patterns must also address the need for distributing such command-based interactions among multiple devices and users.

In conclusion, we believe that only empirical work on developing real-world applications can reveal the entirety of DUI design challenges. Apart from the theoretical work on models and meta-models for DUIs only the development, testing and deploying of applications to real-world users will enable researchers to fully understand the possibilities, challenges and constraints of DUIs.

References

1. Borchers J, Buschmann F (2001) A Pattern Approach to Interaction Design. John Wiley & Sons Inc, Chichester
2. Jetter HC, Gerkin J, Milic-Frayling N, Oleksik G, Reiterer H, Jones R, Baumberg J (2009) Deskpiles. http://research.microsoft.com/en-us/projects/deskpiles. Accessed 16 June 2011
3. Goldberg D (1991) What every computer scientist should know about floating-point arithmetic. ACM Comput. Surv. 23:5-48. doi:10.1145/103162.103163
4. Haller M, Leithinger D, Leitner J, Seifried T, Brandl P, Zauner J, Billinghurst M (2006) The Shared Design Space. In: ACM SIGGRAPH 2006 Emerging technologies, ACM Press, New York
5. Haller M, Leitner J, Seifried T, Wallace J, Scott S, Richter C, Brandl P, Gokcezade A (2010) The NiCE Discussion Room: Integrating Paper and Digital Media to Support Co-Located Group Meetings. In: Proc. CHI 2010, ACM Press, New York
6. Hofmeester K, Wixon D (2010) Using metaphors to create a natural user interface for microsoft surface. In: Ext. Abstracts CHI 2010, ACM Press, New York
7. Hutchins E, Hollan J, Norman D (1985) Direct manipulation interfaces. Human-Computer Interaction 1,4:311-338.
8. Jacob RJK, Girouard A, Hirshfield LM, Horn MS, Shaer O, Solovey ET, Zigelbaum J (2008) Reality-based interaction: a framework for post-WIMP interfaces. In: Proc. CHI 2008, ACM Press, New York
9. Jetter HC, Gerken J, Zöllner M, Reiterer H (2010) Model-based Design and Prototyping of Interactive Spaces for Information Interaction. In: Proc. HCSE 2010, Springer, New York
10. Jetter HC, Gerken J, Zöllner M, Reiterer H, Milic-Frayling N (2011) Materializing the Query with Facet-Streams – A Hybrid Surface for Collaborative Search on Tabletops. In: Proc. CHI 2011, ACM Press, New York
11. Melchior J, Grolaux D, Vanderdonckt J, Van Roy P (2009) A toolkit for peer-to-peer distributed user interfaces: concepts, implementation, and applications. In: Proc. of EICS '09, ACM Press, New York
12. Perlin K, Fox D (1993) Pad: an alternative approach to the computer interface. In: Proc. SIGGRAPH 1993, ACM Press, New York
13. Raskin J (2000) The humane interface: new directions for designing interactive systems. Addison-Wesley, Boston
14. Shaer O, Jacob RJK (2009) A Specification Paradigm for the Design and Implementation of Tangible User Interfaces. ACM TOCHI 16,4:20:1-20:39. doi:10.1145/1614390.1614395
15. Smith J (2009) WPF Apps with the Model-View-ViewModel Design Pattern. MSDN Magazine. http://msdn.microsoft.com/en-us/magazine/dd419663.aspx. Accessed 16 June 2011
16. Wigdor D, Shen C, Forlines C, Balakrishnan R (2006) Table-centric interactive spaces for real-time collaboration. In: Proc. AVI 2006, ACM Press, New York
17. Zöllner M, Jetter HC (2010) ZOIL Framework. http://zoil.codeplex.com/. Accessed 16 June 2011.