ZOIL: A Design Paradigm and Software Framework for Post-WIMP Distributed User Interfaces

Michael Zöllner, Hans-Christian Jetter, Harald Reiterer
University of Konstanz, Human-Computer Interaction Group
{michael.zoellner,hans-christian.jetter,harald.reiterer}@uni-konstanz.de

ABSTRACT
This paper introduces ZOIL (**Z**oomable **O**bject-Oriented **I**nformation **L**andscape), a design paradigm and software framework for post-WIMP distributed, zoomable and object-oriented user interfaces. We present ZOIL's design principles, functionality and software patterns to share them with other researchers. Furthermore, ZOIL's implementation as a software framework for C# and WPF (Windows Presentation Foundation) is available as open source under the BSD License (see http://zoil.codeplex.com).

Author Keywords
zoomable user interfaces, post-WIMP user interfaces, object-oriented user interfaces, software framework

ACM Classification Keywords
H5.2. [Information interfaces and presentation]: User Interfaces.

INTRODUCTION
To this day, a great deal of DUI research has focused primarily on the distribution of standard graphical UIs and its widgets (e.g. [2, 14]). We introduce ZOIL, a design paradigm and software framework targeted at post-WIMP (post-"Windows Icons Menus Pointer") user interfaces that can be distributed over multiple devices and displays. Post-WIMP DUIs are particularly challenging: While the essence and the building blocks of a standard desktop "WIMP" GUI are well established, there's no such concise pattern or comparable blueprint for the post-WIMP world. Furthermore, the implementation of such post-WIMP user interfaces is not explicitly supported by the established user interface toolkits and programming models and therefore requires great expertise in many heterogeneous soft- and hardware technologies. ZOIL provides help for design and implementation: In the initial stage of design, the ZOIL design paradigm suggests possible patterns of solution or blueprints as heuristics for a post-WIMP DUI architecture and for appropriate visualization and interaction techniques. The prototyping and implementation phases are supported by the ZOIL software framework that facilitates an efficient implementation of ZOIL's defining features [8, 18].

CHI 2011, May 7–12, 2011, Vancouver, BC, Canada.

THE ZOIL DESIGN PARADIGM
Applications following the ZOIL design paradigm integrate all kinds of information items from the application domain with all their connected functionality and with their mutual relations into a single visual workspace under a consistent interaction model [10]. This visual workspace is called the "information landscape" and its interaction model follows the principles of Zoomable and Object-Oriented User Interfaces (ZUIs [16, 20], OOUIs [3, 13]): All information items are integrated into a single object-oriented domain model and appear as visual objects at one or more places in the information landscape. This way, the domain model becomes directly user-accessible similar to the "naked objects" pattern of Pawson [15]. Unlike in today's desktop metaphor, the information landscape is not limited to the visible screen size but resembles a virtual canvas of infinite size and resolution for persistent visual-spatial information management. In the information landscape all items and their connected functionality can be accessed directly in an object-oriented manner [3] without the use of application windows simply by panning to the right spot in the information landscape and zooming in [20]. ZOIL thereby uses "semantic zooming" [16] which means that the geometric growth in display space is not only used to render more details, but to reveal more and semantically different content and all available functionality "on-the-spot". This smooth transition between iconic representation, meta-data, and full-text/full-functionality prevents the problems of information overload and disorientation which are caused by traditional WIMP approaches with multiple overlaying windows or occluding renderings of details-on-demand.

Although the previous description focuses mainly on a single user's view, ZOIL's key strength lies in its great flexibility in distributed and collaborative scenarios: Using ZOIL, DUIs can be designed either as *homogeneous* or *heterogeneous* [2], i.e. the UIs can either offer multiple instances of the same user interface at different sites or the functionality provided at each site may be different. This is achieved by the real-time distribution of the object-oriented data model of the information landscape to all devices in an interactive space (Fig. 1). Every connected client is free to choose which part of the landscape it provides to the user. Thus, users can navigate and manipulate the shared landscape independently using the full breadth of available device-specific input and output modalities. A client is also free to choose other visualization styles, e.g. for a device-, user-, or task-specific screen representation.

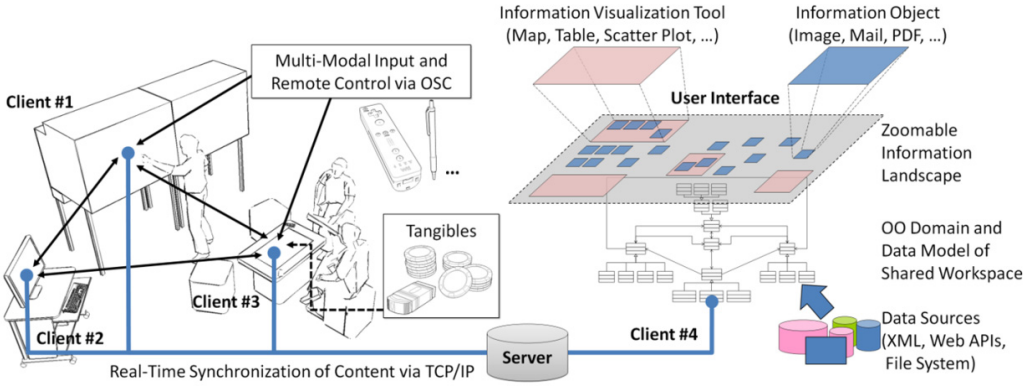

Figure 1. A ZOIL-based multi-user, multi-surface, and multi-device interactive space (left). The right section shows the architecture and user interface of a ZOIL client application in greater detail.

THE ZOIL SOFTWARE FRAMEWORK

Implementing a post-WIMP distributed user interface following the ZOIL design paradigm without tool support is a difficult task that demands a wide range of expertise ranging from distributed databases to vector-based rendering of UI controls. To facilitate ZOIL's application in practice, we have developed the ZOIL software framework [19] that provides the necessary tool support as an extensible collection of software components and classes for C#/WPF. Although individual aspects of ZOIL's implementation are already covered by existing frameworks such as Piccolo [1] or DiamondSpin [21], incompatibilities between languages, UI frameworks, and device SDKs have led us to build a new custom framework for our purposes. The framework has been used for the implementation of various user interfaces (e.g. [9, 17]) and its API usability was evaluated in a longitudinal study [5]. In the light of the framework's volume and broad scope, we focus our description here on defining features to share our lessons learned from the implementation. We describe successful software patterns and architectures used in our framework that can be employed by other researchers in own projects to realize important functionality for post-WIMP UI design.

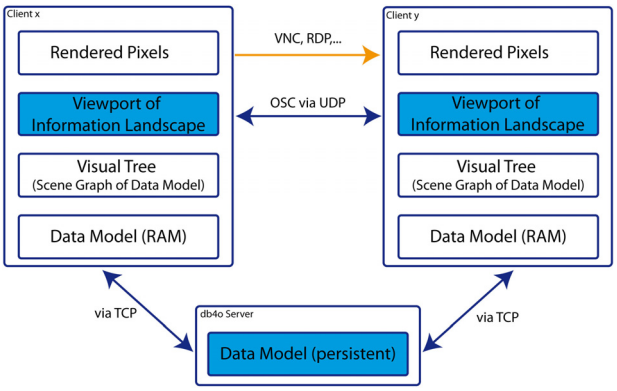

Figure 2. Overview of ZOIL's distribution capabilities

Client-Server Architecture with Transparent Persistence

For realizing the distribution from Fig. 1, ZOIL employs a central database server for real-time synchronization and persistence. Following the nature of the real world, ZOIL's objects and information landscape do not lose their individual state, properties, or location when disappearing from the screen or after closing the application. Furthermore they are synchronized with all other connected clients in real-time to allow for multi-device and multi-user interaction. We have decided to distribute the UI by synchronizing the data model of the shared visual workspace instead of the UI's visual appearance using protocols on the "pixel level" such as VNC or RDP. Thus, the overall network load of a ZOIL setup is relatively small, since only small changes of the data model are transmitted (Fig. 2). To achieve this, the ZOIL software framework uses the object database db4o [4] with its Transparent Persistence (TP) mechanism. The aim of TP is to allow programmers to access or change persistent data in their application with the same ease as non-persistent data models in main memory. This means that with TP and db4o, ZOIL developers are not concerned with persistence at all since persistence simply becomes a natural property of all ZOIL objects. Using TP, programmers can simply change the states and locations of objects on the client-side using standard C# code. Invisible to the programmer, all changes are observed by db4o's TP implementation. It collects all changes and transmits them via TCP/IP to the database server in regular intervals (typically 100ms). The server persists the new state of the landscape and informs all other clients about the changes which are then automatically retrieved and executed on the remote clients within 100-300ms (in typical setups). Thereby the use of the object database db4o instead of traditional Object-Relational Mapping libraries (e.g. Hibernate) also facilitates iterative design, since changes in the objects' class definitions in C# are directly recognized by db4o without the need to update external XML mappings.

Model-View-ViewModel Pattern (MVVM)

TP offers low-threshold persistency for data objects, but it cannot be applied directly on WPF's visual hierarchies or user controls. Moreover such exact replications of widgets or controls would not be desirable in ZOIL, since each

client should provide the user with an individual view on the landscape's shared data model, so that users can independently navigate and manipulate content. For this reason, the ZOIL framework uses the MVVM Pattern [7] to provide a novel way of separating the persistent data *model* of an object from the non-persistent *view* of the object in the landscape. Using TP, each object's *model* is shared via the server with all others clients, but the corresponding *view* is not shared and only resides on the client-side. As soon as a *model* is created (or destroyed) in a local or remote client, the corresponding *views* on all other clients are created (or destroyed) accordingly. The key advantage of integrating MVVM in ZOIL over using a traditional MVC approach lies in the *ViewModel*, which plays the role of an adapter between the XAML (Extensible Application Markup Language) markup of the *view* and the C# *model* by tailoring the data of the model to a view-friendly format. Properties and commands of the *ViewModel* are bound via two-way data binding to the *view*. *Model* and *ViewModel* always stay synchronized, so changes in the model get routed to the view and vice versa. Using MVVM, *views* can also be defined more conveniently in declarative XAML, without the need for C# code.

```
<ZPhoto
    ZObjectDragDropBehavior.IsDraggable="True"
    ZObjectResizeBehavior.IsResizable="False"
    ZObjectRotateBehavior.IsRotatable="True"
    ZObjectPhysicsBehavior.HasInertia="True"
    ZInformationLandscape.ZoomTarget="True">
    ZInformationLandscape.ZoomMargin="5,10,5,5">
    <!-- further definition of ZEmail -->
</ZPhoto>
```

Code Sample 1. XAML code for making ZPhoto a draggable, resizable, and rotatable zoom target.

Declarative Definition of Zoomable UI Components

One key feature of our framework is the comprehensive support for authoring and implementing rich ZUIs with semantic zooming using WPF and its declarative XAML language. Unlike Java's SWT, Swing, or WinForms, WPF natively supports vector-based and hardware-accelerated rendering of controls. This enables ZOIL designers to use the full range of WPF controls (e.g. sliders, player controls) in ZUIs without pixelation or the need for wrapper classes as in [1]. This also allows programmers to easily integrate rich media content such as video streams or 3d models and increases the prototyping efficiency in comparison to other hardware-accelerated graphics APIs without any or only very basic sets of user controls (e.g. OpenGL, DirectX). A further advantage of WPF is the possibility to separate the object's implementation logic in C# from its visual design in XAML. This allows designers to use a low-threshold HTML-like language and visual editors for defining an object's appearance. As a consequence ZOIL supports typical designer-developer workflows and lowers the threshold for non-programmers to use the framework. Following this declarative paradigm, we have extended the available set of controls and behaviors in XAML with a collection of ZOIL-specific components that enables designers to define semantic zooming and other behaviors without the need for procedural C# code (Code Sample 1). The different appearances are selected by ZOIL's `ZComponentFrames` container depending on the current display size and are exchanged using a smooth zooming and opacity animation.

```
<ZEmail x:Class="ZComponent" … >
 <ZComponentFrames>
  <ZComponentFrame WidthNeeded="0">
   … appearance at 1st zoom level …
  </ZComponentFrame>
  <ZComponentFrame WidthNeeded="150">
   … appearance at 2nd zoom level …
  </ZComponentFrame>
  … add appearances for more zoom levels here …
 </ZComponentFrames>
</ZEmail>
```

Code Sample 2. XAML code for defining the semantic zoom of an email object.

Attached Behavior Pattern (ABP)

For designing object-oriented interaction it is not sufficient to define an object's visual appearance alone. Objects also have to be assigned the desired behaviors: Can an object be dragged, resized, or rotated using multi-touch manipulations? Does an object simulate physical behavior such as inertia or friction during these manipulations? Is the object a zoom target, so that a click or tap on it starts a zooming animation into the object until it covers the screen? Or should there be a small margin remaining around the object? To help designers to easily assign such behaviors to an object without reinventing the wheel for every type, our framework makes extensive use of the Attached Behavior Pattern (ABP) [6]. ABP encapsulates behaviors into a class outside the object's class hierarchy and allows it to be applied on classes or individual instances of objects. When using WPF this threshold can be further lowered using XAML instead of C# (Code Sample 2).We believe that the Attached Behavior Pattern introduces a very natural view of interactive behavior into post-WIMP programming that facilitates iterative design without the need for changes in procedural C# code or class hierarchies. Furthermore by using ABP, a framework can provide designers with an extensible library of potentially useful behaviors that can be easily assigned in visual editors.

Input Device and Client-Client Communication with OSC

The ZOIL framework provides simple ways to connect to multi-modal input libraries or other kinds of input device middleware [11, 12]. Applications can use the stateless UDP-based OSC protocol to connect to open source tools like Squidy [11] that facilitate the integration of external post-WIMP devices such as Anoto pens or the Nintendo Wiimote. OSC has several advantages for prototyping: First, clients stay tolerant to unavailable devices or changing infrastructures during their lifetime, since no permanent connections have to be established. Second,

UDP packets can be broadcasted to all clients in the subnet and therefore sending to specific IP addresses is unnecessary. Instead, packet destinations can be specified above the network layer using OSC addresses. This is desirable in early phases of prototyping and experimentation where devices are often added or removed. In some scenarios (e.g. in an augmented meeting room) it might be important to tightly couple two or more clients' views. By sending OSC commands between clients it is possible to control the current camera or viewport of a remote client from a local client. This allows scenarios of collocated collaboration between multiple users and devices. For example, a view on a region of the landscape can be transferred from a personal device to a public large display or vice versa. OSC can also be used to implement an overview and detail coupling between two clients.

CONCLUSION

We have introduced the ZOIL design paradigm and software framework. ZOIL is unique in its focus on distributed post-WIMP user interfaces that are based on a real-time synchronized multi-user/multi-display/multi-device visual workspace. A ZOIL setup therefore follows Melchior et al.'s notion of a Distributed User Interface: "*A DUI consists of a UI having the ability to distribute parts or whole of its components across multiple monitors, devices, platforms, displays, and/or users*" [14]. Given the broad design space of post-WIMP user interfaces, a generic design or implementation of "optimal" distribution is very difficult, if not impossible. In a ZOIL setup, it is possible to leave such decisions to each connected client, thereby avoiding the risk of losing usability or functionality and getting the most out of every client's input and output modalities. While ZOIL is inherently multi-monitor, multi-display, multi-device and multi-user it is not truly multi-platform yet, since it only supports the MS Windows platform. We will evaluate if future versions of ZOIL can leverage Microsoft's Silverlight to come closer to a multi-platform solution.

REFERENCES

1. Bederson, B., Grosjean, J., and Meyer, J. Toolkit design for interactive structured graphics. *IEEE Trans. Softw. Eng. 30* (August 2004), 535–546.
2. Bharat, K., and Brown, M. H. Building distributed, multi-user applications by direct manipulation. In *Proc. UIST 1994*, 71–80.
3. Collins, D. *Designing Object-Oriented User Interfaces*, 1st ed. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1994.
4. db4o. Native java & .net open source object database. http://www.db4o.com.
5. Gerken, J., Jetter, H.-C., Zoellner, M., Mader, M., and Reiterer, H. The concept maps method as a tool to evaluate the usability of apis. In *Proc. CHI 2011*.
6. Gossman, J. The attached behavior pattern. http://blogs.msdn.com/johngossman.
7. Gossman, J. Introduction to the model view viewmodel pattern. http:/blogs.msdn.com/johngossman.
8. Jetter, H.-C., Gerken, J., Zöllner, M., and Reiterer, H. Model-based design and prototyping of interactive spaces for information interaction. In *Proc. HCSE 2010*, 22–37.
9. Jetter, H.-C., Gerken, J., Zoellner, M., Reiterer, H., and Milic-Frayling, N. Materializing the query with facet-streams – a hybrid surface for collaborative search on tabletops. In *Proc. CHI 2011*.
10. Jetter, H.-C., König, W. A., Gerken, J., and Reiterer, H. Zoil - a cross-platform user interface paradigm for personal information management. Personal Information Management: The disappearing desktop (a CHI 2008 Workshop).
11. König, W., Rädle, R., and Reiterer, H. Interactive design of multimodal user interfaces - reducing technical and visual complexity. *Journal on Multimodal User Interfaces 3*, 3 (Feb 2010), 197–213.
12. Lawson, J., Al-Akkad, A., Vanderdonckt, J., and Macq, B. An open source workbench for prototyping multimodal interactions based on off-the-shelf heterogeneous components. In *Proc. EICS 09*, 245-254.
13. Mandel, T. *The GUI-OOUI War, Windows vs. OS/2: the designer's guide to human-computer interfaces*. Van Nostrand Reinhold Co., New York, 1994.
14. Melchior, J., Grolaux, D., Vanderdonckt, J., and Van Roy, P. A toolkit for peer-to-peer distributed user interfaces: concepts, implementation, and applications. In *Proc. EICS 2009*, 69–78.
15. Pawson, R. *Naked Objects*. PhD thesis, Trinity College, Dublin, Ireland, 2004.
16. Perlin, K., and Fox, D. Pad: an alternative approach to the computer interface. In *Proc. SIGGRAPH 93*, 57-64.
17. Project Deskpiles. http://research.microsoft.com/en-us/projects/deskpiles.
18. Project Permaedia/ZOIL. http://hci.uni-konstanz.de/permaedia.
19. Project ZOIL on Codeplex. http://zoil.codeplex.com.
20. Raskin, J. *The humane interface: new directions for designing interactive systems*. ACM Press/Addison-Wesley Publishing Co., 2000.
21. Shen, C., Vernier, F., Forlines, C., and Ringel, M. Diamondspin: an extensible toolkit for around-the-table interaction. In *Proc. CHI 2004*, 167–174.