Universitat Konstanz
FB Informatik und Informationswissenschaft
Bachelor-Studiengang Information Engineering

Bachelor Thesis

A Framework for an Infinitely Zoomable Information Landscape

zur Erlangung des akademischen Grades eines
Bachelor of Science (B.Sc.)

Studienfach:	Information Engineering
Schwerpunkt:	Computer Science
Themengebiet:	Angewandte Informatik

von

Andreas Engl

Matr.-Nr.:	01/617679
Erstgutachter:	Prof. Dr. Harald Reiterer
Zweitgutachter:	Prof. Dr. Oliver Deussen
Einreichung:	31.10.2008

Abstract

Since the advent of the very first graphical environments, the desktop metaphor claimed its place as a design that has shaped computing to this very day, yet hasn't undergone any significant evolution apart from keeping itself visually up-to-date. Popularity is not always a measure for quality, though, especially when taking the lack of alternatives into consideration. With the introduction of zoomable user interfaces in the late seventies, an idea has been set into motion that promises to address the weaknesses and inconsistencies users had to accept so far. The ZOIL paradigm is a specialised form of zoomable user interfaces that aims to create a consistent interface by allowing navigation through a set of visualisation and interaction techniques, rather than exploring information through a multitude of applications. This work introduces a practical implementation, aiming to provide a technical foundation to demonstrate the theory, as well as to test design alternatives and to evaluate ideas. It will highlight and explain several key concepts of ZOIL and explain their practical implementations. Finally, a demonstrator along with a roadmap for future development is presented, serving as a design study for the framework's capabilities and as a guideline for following revisions.

Contents

1 Introduction

> *"Creating an interface is much like building a house: If you don't get the foundation right, no amount of decorating can fix the related structure."*
>
> – Jef Raskin

When Raskin used this quote as the opening sentence of "The Humane Interface", he was no doubt referring to the desktop metaphor he perceived as inherently flawed. But he is certainly not the only visionary who wants us to step into the post-desktop era. In "Beyond the Desktop Metaphor", the result of a collective effort of researchers working on alternative approaches, Kaptelini and Czerwinski argue that "is has become evident that the desktop metaphor has inherent limitations" [KC07]. They mention the gap between information access and information display as one of the major issues, which, as early implementations have demonstrated [Don78], is the very aspect Zoomable User Interfaces (ZUIs) excel best in.

While the idea of ZUIs is basically around for decades, it's been popularised only by recent efforts. Donelson already stated in 1978 that people organise information spatially when he introduced the SDMS[1], where users would explore information contained in the database by navigating an infinite plane (the "data surface") using a joystick [Don78]. 15 years later, Perlin and Fox reinforce the concept by developing Pad [PF93], a system that allowed user interface elements, such as text or images, as well as programmatically generated content to be placed in a zoomable information landscape. With Pad, the longest running effort of ZUIs was born, which in its fourth incarnation Piccolo[2] - after two re-implementations called Pad++ and Jazz - marks the most widely spread and commercially used ZUI toolkit.

Today, with the advent of mobile devices and high resolution content, zooming as an interaction technique is mostly used for exploring maps, browsing media content and surfing the world wide web. Several web browsers have been devised that heavily utilise zooming, such as the Opera Mini[3] for handheld devices, the Nintendo Wii[4] Internet channel and Safari for the iPhone. While scaling is a necessity for legible content on small displays, it can be used to improve accessibility in the desktop domain as well. Not only does every modern browser consequently provide means to scale font-size or content, but some newer applications such as Microsoft Expression Blend[5] also allow user to magnify the entire interface. Projects such as PicLens[6], PhotoMesa[7] and Microsoft's Photosynth[8] employ ZUIs to explore a vast amount of

[1] Spatial Data Management System
[2] http://www.cs.umd.edu/hcil/jazz/
[3] http://www.operamini.com/
[4] http://www.nintendo.com/wii/channels/internetchannel
[5] http://www.microsoft.com/expression/products/Overview.aspx?key=blend
[6] http://www.cooliris.com/
[7] http://www.cs.umd.edu/hcil/photomesa/
[8] http://livelabs.com/photosynth/

pictures and videos without sacrificing overview and make it an enjoyable experience at the same time. Lastly, Google Maps[9], Microsoft Virtual Earth[10] and similar services use zooming in the most classic fashion, which is the exploration of maps and satellite images.

All of the aforementioned developments are fairly recent and reveal a strong tendency towards ZUIs in the mainstream market, yet none of them take a step beyond using zooming as a mere exploration tool and thus still use the traditional WIMP[11] paradigm as a basis. Consequently, we're seeing more and more of a mixture between the classic desktop metaphor and zoomable components which differ greatly from one application to another in terms of their underlying principles, input mapping and usability. This work will therefore build upon the more strictly defined concept ZOIL, which is meant to replace every aspect of WIMP and to provide a consistent experience of use across the entire workspace. The theoretical foundation of ZOIL has already been explored in great detail in previous works [Kön06, Ger06, Jet07] and is only briefly outlined here; the majority of this document aims to provide a practical framework for building both, prototypes for demonstrating the paradigm, as well as entire end-user applications or environments.

The name "ZOIL framework", as opposed to "application", "toolkit" or "environment" has been chosen due to the nature it is used. Generally, there is an end-user application, prototype or demo that builds on top of the framework (which, by itself, produces no visible output). Even though some helpful functionality is exposed, it mainly integrates itself passively by providing a set of building blocks which can be embedded and configured. Thus it is neither a fully qualified program, nor a mere set of methods.

To exemplify the framework's use, a fully interactive prototype built for the Euro iTV 2008 conference in Salzburg is presented as a case-study for the framework [JESR08]. The work is then summed up with a set of requirements and lessons learnt, forming a roadmap for future development.

1.1 ZOIL: A Brief Explanation

While it's not necessary to know each of ZOIL's numerous design philosophies to appreciate the framework's architecture and design decisions, it is undoubtedly helpful for the overall understanding to have a reasonable grasp on the general idea behind ZOIL, which this section is meant to convey.

The term ZOIL stands for "zoomable object-oriented information landscape", a term that already reveals the general idea of the paradigm. It is an "application- and

[9] http://maps.google.com/
[10] http://www.microsoft.com/virtualearth/
[11] Windows, Icons, Menus and Pointers

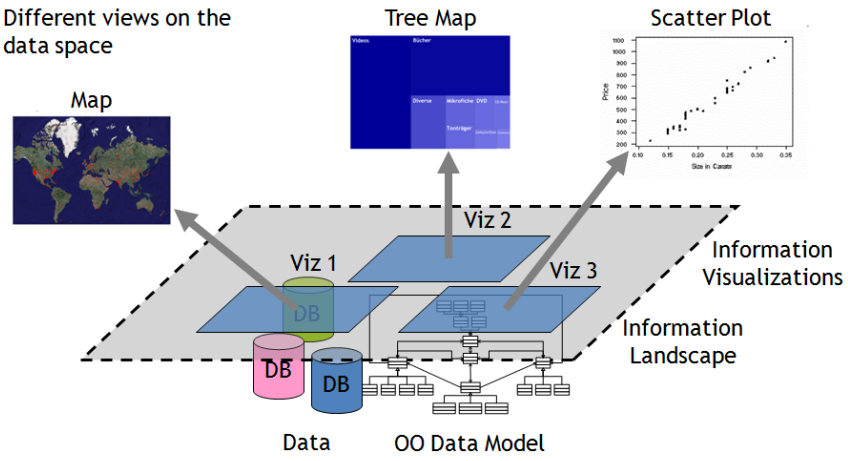

Figure 1: *A graphical representation of the ZOIL paradigm: different visualisation that are situated on the information landscape organise and display the data taken from an object-oriented data model.*

platform-independent UI concept", that is "aimed at unifying all types of local and remote information items with their connected functionality and with their mutual relations in a single visual workspace", referred to as the information landscape [JKGR08]. This landscape follows a consistent interaction model, including basic ZUI navigation techniques such as zooming and panning. As it is not limited by the visible screen size, but rather resembles "virtual canvas of infinite size and resolution", interest in particular information items can be expressed by zooming onto them, thus increasing the amount of available screen real-estate which is then used to enrich the objects by additional information, a process called semantic zooming. By embedding object-related functionality (such as a movie player that is directly part of a movie component), ZOIL hopes to supersede application-based exploration, replacing it with an entirely object-oriented approach. At any point, a set of information item can be viewed in a different visualisation by grouping them in so-called "portals", which allow different ad-hoc structures to be applied, for example viewing their location on a map. The resulting subset can be further broken down using different visualisations, effectively "formulate complex queries" by the means of "nested information visuali[s]ations". A resulting benefit from the nature of ZUIs is that the "user interface scales to different display sizes and screen resolutions", allowing ZOIL-based user interfaces to be used on various devices such as Ultra Mobile PCs, PDAs and smart phones.

2 The ZOIL Framework

2.1 Motivation and Design

> *"We argue that ZUIs, like 3D interfaces have embraced a whole new paradigm, affording new representation possibilities and new interaction capabilities – in effect creating a new medium. One of the resulting challenges is that new authoring technologies are needed." [FZ98]*

The project to build a new ZUI framework started primarily out of the need to provide a basis to easily and quickly create prototypes that could be employed for usability testing and demonstrations of the ZOIL paradigm. Therefore, the focus has always been on ease of use rather than trying to embed as many features as possible. As a result, the most fundamental design decision was to build upon an already existing framework, leveraging its advantages while being careful not to introduce a complex new class hierarchy for programmers to learn. The framework that provided us with most flexibility and ease of use turned out to be the Windows Presentation Foundation (WPF), which will be highlighted in the following section. Chapter 3 will outline various reasons against using existing, more ZUI-aware frameworks such as Piccolo.

Bederson et al. point out that the most fundamental difference in the conception of a framework is its approach to composing functionality [BGM04]. Jazz primarily uses run-time composition of unrelated controls to create more complex interfaces, called a polylithic framework. Monolithic toolkits such as Piccolo, in contrast, mainly use class inheritance to extend controls in functionality. While polylithic systems tend to achieve greater re-usability and thus a more flexible class hierarchy they impose a greater knowledge upon the programmer, who has to "create, understand and manage more objects". Since WPF - like most other general purpose frameworks - heeds a polylithic philosophy, the presented framework tries to adhere to its underlying foundation. The hope is to overcome the aforementioned drawback of polylithic systems by trying to re-use as many of the native classes as possible, so that a programmer adept in using WPF can use this knowledge to his/her advantage.

Ultimately, the ZOIL framework aims to be a software framework that completely replaces existing desktop-based solutions and provide a zoomable environment for personal information management. Each of the subsequently discussed aspects, as well as the presented prototype, is designed with this goal in mind. The framework's scope is as wide as to actually implement user interface details and to include visual designs.

2.2 Framework Foundation

As previously mentioned, the ZOIL framework uses Microsoft's WPF, a subset of the .NET 3.0 framework as its foundation. An important factor in that decision was the introduction of an XML-based interface specification language called XAML[12]. The primary focus was the rapid creation of prototypes; thus by using a declarative UI system, the quick design of interfaces that would still employ free and directed, as well as semantic zooming (see chapter 2.4.1 and 2.4.2), becomes possible without writing a single line of procedural code. Moreover, almost all of it can be done with the help of visual language tools such as Microsoft's Expression Blend or the in-built designer in Microsoft's Visual Studio 2008.

Since animation and scaling of content has to be a real-time operation to successfully support the user to create a mental map of the spatial environment, a refresh rate of more than 10, but ideally 20 to 30 frames per second has to be achieved [BB99]. If the aim is to - in the long run - provide a viable alternative to the desktop metaphor, displaying and scaling of large quantities of high resolution images, as well as full-length high definition movies should not be a significant burden to the framework's performance. The mere use of a computer's central processing unit is often insufficient, which typically means that programmers have to resort to low-level application programming interfaces such as OpenGL or Direct3D. Pietriga, however, makes the justified observation that "[these] are powerful but cost more implementation and maintenance time, [...] [and] as a consequence, [...] HCI aspects of visual language environments and visual language meta-tools are often under-developed" [Pie05].

In answer to the growing number of CPU-intensive applications, newer framework such as WPF delegate much of the processing work to the graphical processing units, in addition to providing improved rendering algorithms. The first benchmark prior to the work on the ZOIL framework consisted of zooming 30 different unscaled and uncompressed 8-mega-pixel bitmaps on an average, 1.6 GHz notebook. Both when navigating close-up onto certain regions, as well as in the overview the demo ran smoothly, without any additional algorithms or other optimisations. Apart from hardware acceleration, WPF also provides high-level rendering constructs such as the element tree, a Direct3D aided implementation of what is commonly called a "Scene Graph" [Mic08c] (further discussed in 2.4.4) that already takes care of many of the optimisations done in Jazz and Piccolo, as listed by Bederson et al., such as calculating bounding rectangles, hit-testing, culling and dirty region updates [BGM04].

It is also noteworthy that WPF's rendering is entirely based on vector graphics [Mic08d]. Contrary to raster graphics, an interface constructed from vectors can be scaled to any size without having to stretch it, revealing jagged etches and pixel blocks.

This allows the framework to re-use existing WPF components along with their layout

[12] Extensible Application Markup Language

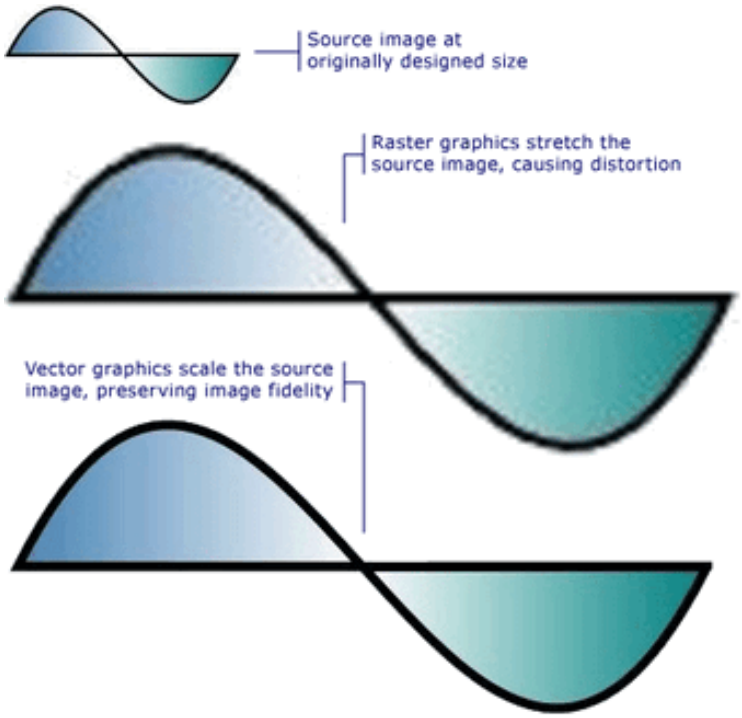

Figure 2: *Comparison between the scaled versions of a source image using raster graphics (on top) and vector graphics (below) (Source: MSDN).*

in a zoomable environment, as well as greatly simplifying the work needed to implement custom, vector based graphics.

2.3 Framework Architecture

Even though it is not the focus of this work, some concepts are better understood with some basic implementation knowledge in mind. Technically, the ZOIL framework is a DLL[13] that end-user applications can utilise. Its architecture is best described by dividing it into ”six logical blocks” [Eng08a]: the framework core, components, controls, visualisations, input and themes.

The **core** is responsible for global functionality, partly used internally and partly offered to external programs. It represents the only interface that applications built on top of the framework directly communicate with. A typical ZOIL application uses the core to load a theme (a collection of structural and visual information defined in the ”**themes** block”) and attach any number of **input** devices (in case it is interactive). **Components**, in turn, are high-level visual elements used to populate the information landscape, for example movies, photos, mail or contacts. These come with a logical implementation, as well as various visual representations (defined by the respective

[13] Dynamic-link library

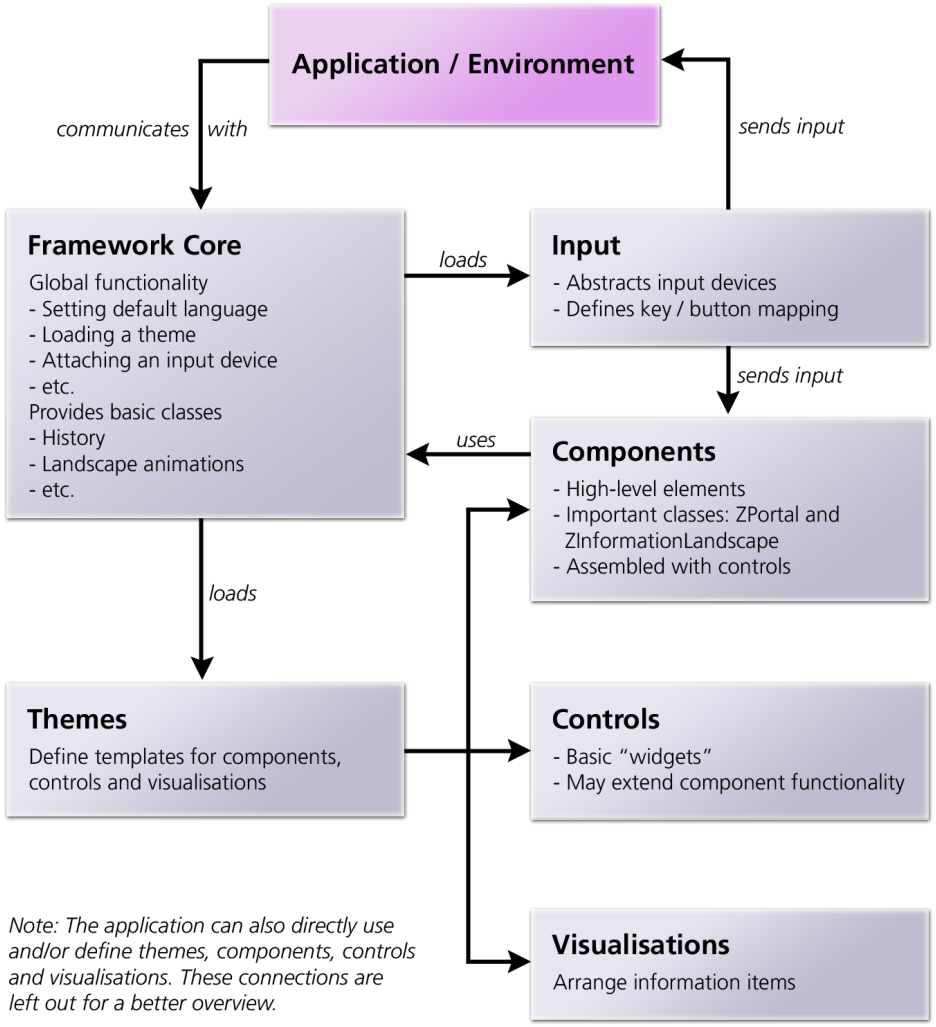

Figure 3: *Architectural overview of the ZOIL framework (Source: [Eng08a]).*

theme) that can, but don't have to be used: an application can chose to define its own visuals. Functionality of the application is largely defined by using a hierarchy of components, usually with an information landscape component at its root (which, again, stresses the polylithic philosophy of the framework). How these components are arranged on screen is entirely determined by **visualisations**, which are simply a layer in the hierarchy that uses an algorithm to determine the coordinates of all information items it contains. Lastly, **controls** are a loosely defined set of "widgets" that may be either building blocks for components (such as buttons and sliders) or interface elements (such as a pie menu). Figure 3 shows a graphical overview of the aforementioned logical blocks and how they relate to each other.

2.4 Concepts and Implementation

This chapter intends to examine all major concepts of the ZOIL paradigm and provide a basic overview of the ways they are supported by the framework. While the subsequent sections do not claim to fully analyse each and every aspect of ZOIL, they aim to clarify design, structure and rationale behind major framework implementations.[14]

2.4.1 Zooming

> *"With [traditional hypertext] it is like closing your eyes*
> *and when you open them you're in a new place.*
> *Zooming lets you keep your eyes open."*
>
> – A. Druin, as cited in [BB99]

Not only is zooming a requirement for any ZUI, but it represents the most common form of interaction in ZOIL. Generally we can distinguish "Jump Zooming", a technique that - much like hyperlinks - immediately take the viewport onto the target position. Animated zooming, in contrast, smoothly interpolates between the initial and final view [Kön06]. Additionally, zooming can be distinguished between "Pad-mode" and the "lodestones and leylines technique" or "leylines-mode" [Jul02]. Pad-mode zooming means that the user can magnify or diminish any given point on the screen and will henceforth be referred to as "free zooming". Leylines-mode "limits conceptual movement to locations [...] that are relevant to the user's task or are necessary to navigation". For the object-oriented ZOIL paradigm, this translates into a fixed viewport behaviour for each item on the information landscape that is picked as a zooming target, such as fitting it to the screen. For simplicity's sake and easier understanding, the latter mode will therefore be called "directed zooming" in the following.

As the Druin's quote suggests, traditional navigation based on hyperlinks forces a user onto a new position that has little or no immediately apparent relationship to the previously displayed content. Bederson and Boltman argue, that an "[i]nteractive animation is used to shift some of the user's cognitive load to the human perceptual system", thus allowing him to "track substructure relationships without thinking about it" [BB99]. While these animations can be used to emphasise any change of states in the data or within an interface, mental maps of spatial information are especially useful in the context of zooming. A usability test performed with twenty subjects at the University of Maryland has shown a statistically significant improvement in accuracy using animated zooming. Other research mentioned in the same study has measured improved subjective user satisfaction.

[14] For the actual implementation details on the various topics, please refer to either the project documentation or the Permaedia Wiki [Eng08a]

Figure 4: *A gallery of images after a directed zoom onto one of them. The surrounding images are still partially visible an provide convenient targets for browsing (Source: screenshot of the iTV prototype).*

However, if a system's use of zooming is not restricted, a user may navigate to a region that offers no visual clue as to reveal his/her whereabouts in the information landscape, an issue Jul and Furnas named "desert fog" [JF98]. Both, zooming out until the landscape is no longer recognisable as such, or zooming into either a blank space or closely onto an object can evoke a situation where there's no apparent navigational step that will lead the user back to a more informative view. Jul and Furnas demand a "single unambiguous action" that will solve every instance of a desert fog. Limiting the application to only use directed zooming would prevent desert fog from occuring at all, but contradicts the idea of ZOIL as it keeps the user from reaching and extending the information landscape to yet unpopulated parts [Kön06].

As a consequence, the ZOIL framework does not enforce either zooming mode globally. Free zooming may be disabled altogether or restricted in certain areas or situations, a technique further discussed in chapter 4.4. For directed zooming the framework uses a property that can be attached to any object in the scene, identifying it as a potential zoom target. This approach is therefore "click-sensitive", as employed in the multi-scale editor MuSE [FZ98] and König's ZOIL prototype [Kön06]. The framework extends directed zooming by introducing zoom margins which can be independently specified by each zoom target. Zoom margins are invisible borders that don't affect the layout, but prevent the information item from taking up the entire space after it was zoomed onto. This idea proved to be useful when navigating between group of objects, such as picture galleries, as shown in Figure 4.

Not only do the visual cues of the neighbouring images indicate a gallery-like context, they also provide a short-cut. Originally, a user would have to zoom out and in again or pan manually (which, as Furnas and Bederson point out, is often a longer path since zoom is logarithmic [FB95]) to go to the next picture. By using directed zoom (which is equivalent to a directed panning in this case) onto the partially visible images, he/she can navigate on the same z-plane more comfortably.

This effort, amongst others, aims to put a focus on directed zooming to a point where - navigation-wise - it is a replacement for free zooming and panning in almost all cases. Firstly, because an empirical study conducted by the University of Michigan has measured a significant decrease in task completion time using directed zooming, as well as leaving users ”calmer and more confident in their actions” due to reduced ”mechanical and cognitive demands” [Jul03]. Secondly, it helps to circumvent the ”desert fog” problem, as previously mentioned. Since free zooming is (and has to be) still an option, however, the framework keeps track of the navigation history, in order to provide a single, universally applicable command that will take the user back to the last meaningful position in the information landscape, regardless of his/her current location. In addition, the prototype presented in chapter 4 limits the landscape in a meaningful way to further help navigation.

2.4.2 Semantic Zooming

A ZUI can further aid the user by choosing different representations of an information item depending on its size on the display, a technique Perlin and Fox introduced as ”semantic zooming” [PF93]. A pure geometric zoom scales the dimensions of each item contained in the information landscape uniformly. Thus, a user can focus on a certain area of interest and zoom in to view more and more relevant information as the target increases in size. Likewise, by zooming out and reducing the scale of all displayed items, an ad-hoc overview is created.

Semantic zooming enhances the notion of detail and overview on demand by increasing the amount of both, information and functionality linked with an information item as it comes closer to the viewing plane. The rationale is that by zooming closer, the user actively states an interest for the focused object. While it would make for a very cluttered interface if all associated functionality is provided at all time, semantic zooming allows an overview to only show the level of detail that is necessary to identify information items, yet offer a great amount of meta-data and relevant functionality when there is enough space to do so.

Figure 5 shows an example of a zooming operation that utilises semantic zooming. On the left picture, the movie can easily be recognised by its movie poster and when hovered, a tool-tip reveals the movie’s name along with the most important meta data. After zooming onto the movie the representation changes to reveal both, additional

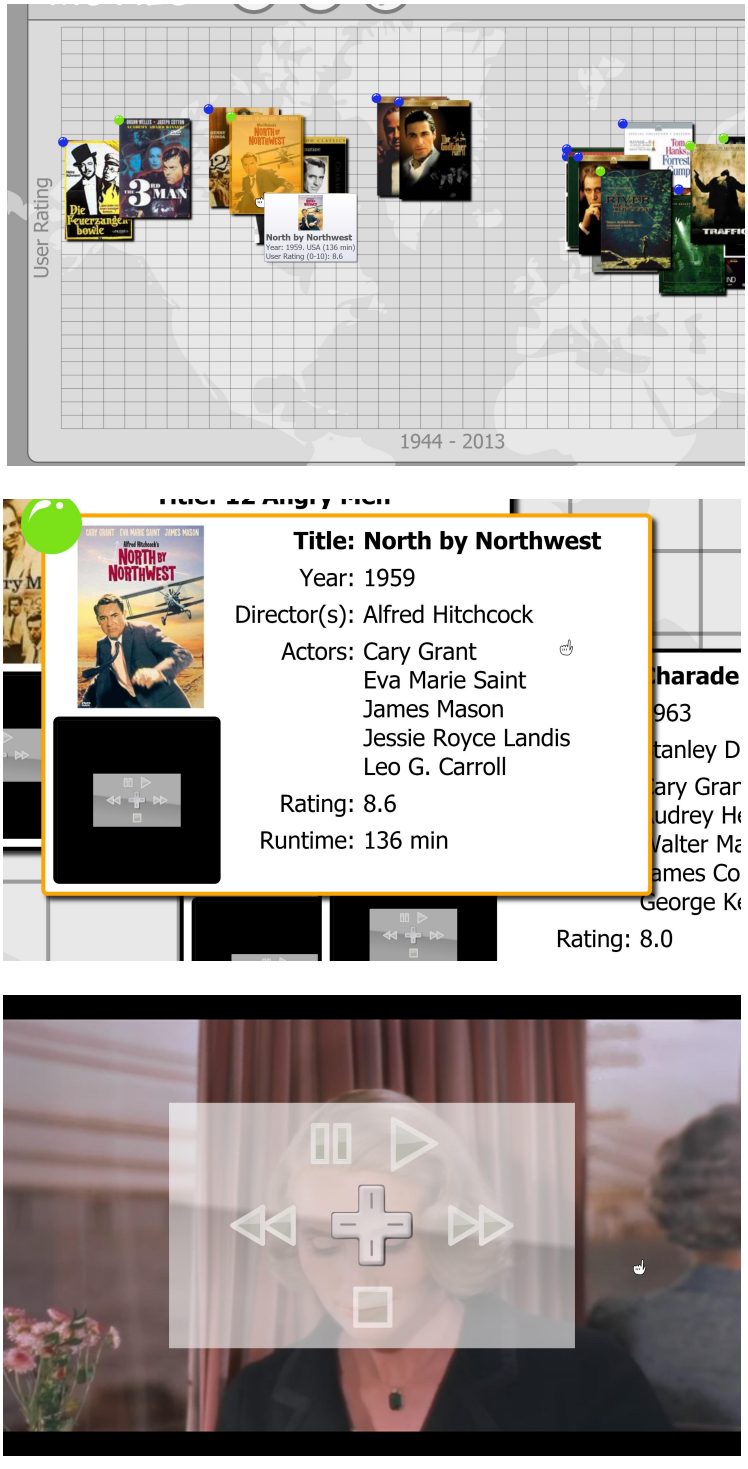

Figure 5: *Different representations of a movie. In overview, only the poster is shown (top). Zooming in reveals related meta-data, as well as the movie (middle) which can be watched in full-screen without having to leave the context (bottom) (Source: screenshots of the iTV prototype).*

information such as the director, actors and genre, and additional functionality, which - in the illustrated case - is watching the full-length movie.

The information uncovered after representation-change is, however, not limited to media content or text. The names of the director and actors can be objects of interest itself, so zooming into any of the names would reveal their birthplace, biography, picture, etc. without leaving the context of the movie. Similarly, zooming onto the embedded movie would provide the user with all the functionality of a video player. A typical WIMP approach to the same scenario would include searching in an often arbitrary hierarchy to find the movie file, then use this file's name (or difficult to retrieve meta data) to query additional information via a web browser and finally start a desktop application capable of viewing movies just to browse to the very same file - the one right in front of the user's eyes - again, using the file open dialogue.

Since semantic zooming is a crucial aspect of ZOIL, its implementation in the framework is a similarly important one and has been designed to be strictly modular and object-oriented. First, each information type chooses a semantic zooming mechanism, the reason for which will be discussed shortly. When the rendered size of a component changes, it notifies the linked semantic zoom mechanism which will determine the appropriate action to take, for example switching from one representation to another. As mentioned before, semantic zoom mechanism is pluggable. While the only one that is implemented at the point of writing is a system using different representation layers, other approaches could in some cases be preferable, such as modelling a single, monolithic transformation that is directly linked to the object's distance to the viewport, thus interpolating between the different views smoothly. It should be noted that both the animations and semantic layers are modelled by each information type itself, as opposed to a single instance that knows of each possible transition. Furthermore, taking the "Design Guidelines for Zooming in ZOIL" [Jet07] into consideration, each representation change can specify an animation that will morph one semantic layer into another. If the component designer refrains from doing so, an automatic fade transition is generated.

When using semantic zooming with layers, working with WPF imposes an important design consequence: each layer uses its own information space and coordinate system, an implementation that was necessary to uphold the idea of an infinite landscape, the reason for which will be discussed in 2.4.4. Therefore the item itself (as depicted in the information landscape), as well as each of its layers have a width and height that can be completely independent in value and aspect ratio, yet still work together. Once the rationale is understood this method allows for great flexibility when designing components, yet imposes an initial conceptual hurdle to designers new to the framework.

2.4.3 Fixed-shape Zooming

As described in greater detail in 2.4.1, and as it is usually the case in ZUIs, the framework scales the entire interface during the zoom operation, as opposed to a strictly object-oriented design where each component would decide how to size itself based on the global zoom level. Due to both, the complexity of aforementioned design, and the significant performance overhead it would impose, simply scaling the entire screen was the most reasonable implementation. However, as Jetter points out, scaling every element according to the rules of a strict, geometric zoom, might not always be the best solution [Jet07]. A label, for example, is only of meaning to the user as long as its text is legible. Borders lose their visual appeal when they become overly thick, as do buttons, scroll-bars and most other basic input elements. While semantic zooming eases the impact of that problem by switching representation when certain elements seem inappropriate, there are scenarios where a uniform scale undoubtedly becomes awkward.

Furnas and Zhang [FZ98] introduced a new method called ”simple fixed-shape zooming”, where the shape of an object remains constant during the zoom process, yet the scale does not strictly correlate with the geometric zoom. An example is fixed-size zooming, which, as the name implies, neutralises the normal magnification change entirely, thus keeping an object at the same width and height. Another special case is the so-called ”anti-zooming”, which inverses the object’s scale direction. When several of the aforementioned techniques are used interchangeably (depending on the object dimensions) the process is called ”compound fixed-shape zooming”. Since semantic zooming may alter the shape during a representation change, a third concept called ”piecewise fixed-shape zooming” is presented, which allows to do so for a finite number of times and assumes a static shape in between.

The framework supports simple, compound, as well as piecewise fixed-shape zooming. Each time the global zoom level changes, all framework elements that subscribe to the fixed-shape zooming mechanism are notified of their new size. Based on the knowledge of their original parameters in an unscaled environment they can calculate their final display dimensions, taking a multiplier value (called the ”zoom modifier”) passed by the component designer into consideration: zero is used for fixed-size, negative values for varying degrees of anti-zooming, while positive values increase or diminish the effect of geometric zooming. The framework further enhances the fixed-shape zooming idea by allowing it to affect not only size, but both width and height independently, border thickness, margin, or any other property, even colour. This approach still retains most of the performance benefits as it has to handle a relatively small number of subscribers, in comparison to the overall amount of framework elements, while retaining many of the advantages of an object-oriented model.

Figure 6: *This chart visually explains the various different zoom modifier settings. The dark rectangle represents an information item.*

Figure 6 shows examples of various different zoom modifier settings and their effect on size. Figure 7 demonstrates simple fixed-size zooming on sizing adorners (depicted as orange circles) of an image, allowing them to be easily clicked when in overview, but preventing them from obscuring the image when zooming in.

2.4.4 The Information Landscape

> *"The totally disorganised have hope."*
>
> – Jef Raskin, The Humane Interface

The most radical change in zoomable user interfaces, compared to the desktop paradigm, was the way information is organised and visually represented. As humans, we're confronted with a vast amount of information on a daily basis, thus Donelson argues that we're already very adept at "organi[sing] our own collections of information" [Don78]. Observing that "[t]his information is most often organi[s]ed spatially" he went on to create and evaluate the concept of an information landscape, originally called the "data surface", that tries to take advantage of our perceptual and memory

Figure 7: *Fixed-shape zooming in action: note how the orange circles around the image (used for scaling and rotating) do not change in size after zooming in (Source: screenshots of the iTV prototype).*

system. Although no details on the nature of the evaluation is given, he finally states that ”user response [...] has confirmed spatial management of information to be an outstanding concept”.

Jef Raskin calls his realisation of the information plane the ”ZoomWorld” [Ras00] and metaphorically describes it as flying over a landscape and occasionally ”diving down” to look at particular items. He mentions labels that should be able to be given to both, individual objects and entire groups, so that when looking for a certain item in the landscape we aren't forced to remember paths or names: following visual cues or certain landmarks that we remember as related due to our spatial memory are an alternative retrieval mechanism that can substantially speed up search. This becomes an especially attractive alternative when considering that correctly labelling and placing an information item in a hierarchical file structure is a cognitively difficult task, even for trained experts, made worse by the fact that the purpose of a folder might change significantly over time [Lan88, JT07, Eng08b]. In comparison to ZUIs, Raskin describes the desktop metaphor as maze-like system of rooms with doors bearing short labels: the only way to find out what's behind is to step through.

The framework defines the information landscape as an object in itself that is placed somewhere in an interface. What might sound counter-intuitive at first provides several design advantages:

- Interface parts that need to be superimposed onto the information landscape, such as an overview (see chapter 2.4.9), the history (see chapter 2.4.7) or feedback messages (see chapter 2.4.8) can simply be placed on a higher or the same level in the interface hierarchy (the ”logical tree”, see chapter 2.4.5) without having to employ fixed-size zooming to keep the element at a constant size (see chapter 2.4.3).
- The framework becomes ”pluggable”. In fact, all it takes to incorporate the ZOIL

framework is to add the information landscape object and everything that is further down the hierarchy automatically becomes zoomable without the need of custom-designed components.

- Multiple information landscapes are possible. While it might not make sense to have several landscapes at the same time from a theoretical standpoint, it offers programmers an easy way to incorporate an alternative view onto a different data tree or sub-tree. An example might be a version system, where the second landscape object is used to display a past revision, with the main landscape reflecting the current state, thus allowing for easy comparison.

The information landscape is, by definition, infinite. As noted in chapter 2.4.2, a work-around has to be used to overcome technical restrictions preventing infinity. Since every element in WPF has a certain dimension, the landscape object is no exception to that rule. If a fixed width and height is assumed, that value continuously decreases for objects further down the hierarchy. This results in two problems: first, as values get smaller, double precision will inevitably be insufficient to distinguish two positions. Second, each information item is sized according to the available space during the phase in which the layout is calculated. Since zoom uses a transformation that occurs after the internal layout pass, a scroll bar, for example, that is defined system-wide as being 10 point wide will completely obscure a control with less than that size. The issue is illustrated with a photo from the iTV prototype in figure 8.

The framework solves the resolution problem by creating a new coordinate system for each information item. A component can choose to specify an internal width and height that is independent from its own, outer dimensions, but greatly affects the scale of subsequent elements in its sub-tree. Furthermore, this value can be changed with each semantic zooming representation, allowing for progressively increasing resolution as the need arises (see chapter 2.4.2). For component designers, the greatest advantage is that layouts can be created in an external designer program for a specific component size and taken into the framework afterwards without modification; it is guaranteed to look exactly the same.

2.4.5 The Element Tree

When first introducing Jazz, Bederson et al. emphasise the importance of scene graphs, as one of their eight requirements for a ZUI library demands support for a large amount of scene objects so that ”rendering and interaction performance doesn't degrade with complex scenes”[15] [BMG00]. Jazz and Piccolo subdivide the scene by using scene graph nodes and structure them in a tree. The nodes itself have no visual

[15]Apart from the fifth, as discussed in chapter 2.5.2, all eight requirements are fulfilled by the ZOIL framework.

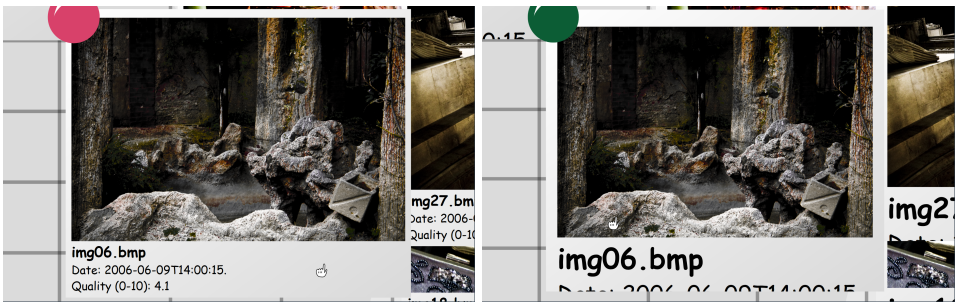

Figure 8: *Although the photo has been given a size of 100 to 120 points by the visualisation, its internal resolution is 1000 to 1200 points to ensure a good picture resolution. In the example below, the image has only been given half its internal resolution, with everything else unchanged. The border around the image is now larger and the text's relative font size is doubled (Source: screenshots of the iTV prototype).*

appearance, but can be used to easily apply an operation, such as affine transformations or culling, and properties like transparency to all child elements.

The ZOIL framework builds upon a similar concept, WPF's element tree [Mic08c], and extends it by certain ZUI characteristics. The element tree is conceptualised by two other trees. When adding controls to an interface, for example a grid containing buttons, the logical tree tracks the high level structure, connecting the grid directly with its children, the buttons. A button, however, is more than just one class: it is typically composed by a border, background and a label. Consequently, the visual tree supports access to the entire visual structure of an interface.

Using both tree mechanisms, the framework iterates every component to enforce various mechanisms:

- Since components are encapsulated using various different coordinate systems, the final rendered size of each element is calculated by a set of matrices.
- Using hit test and bounding rectangle methods provided by the visual tree, viewport intersection is calculated. A component can be either fully or partially contained, or off-screen.
- Information about the rendered size and viewport intersection is stored with each component and can be used for culling and disabling various CPU and memory intensive operations, such as animations.
- The semantic zoom mechanism of every element is notified of any changes in the displayed width and height, using the information to calculate the proper component representation.
- Lastly, all subscribers to fixed-shape zooming are updated.

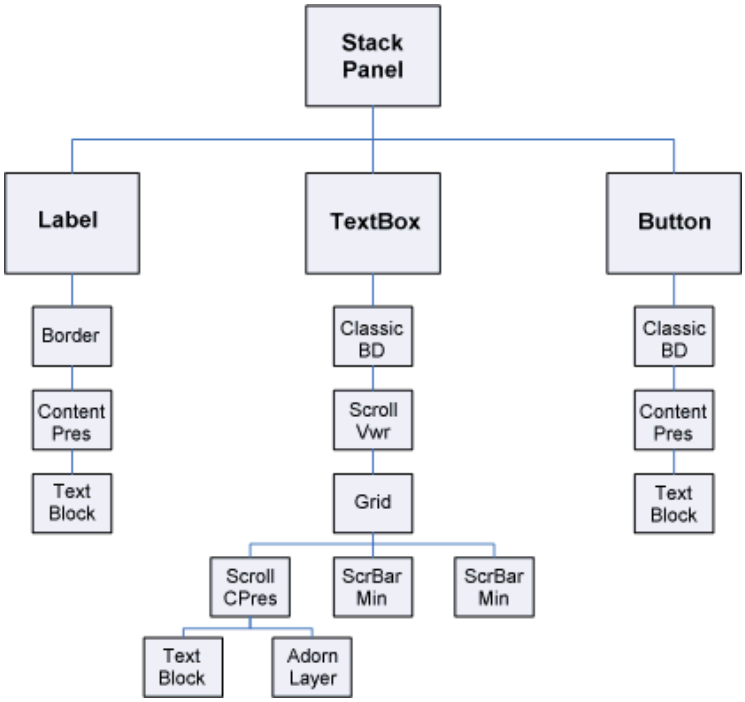

Figure 9: *An example of a visual tree. The elements highlighted in bold face are part of the logical tree (Source: MSDN).*

Apart from using the element tree for aforementioned internal operations it is often used in other places, both in the framework as well as in demos and prototypes. Therefore, a comprehensive set of functions for common tasks is provided, such as finding ancestor elements of a certain type, transforming a point from its local coordinate system into a different one, hit-testing at a certain position for picking operations, or calculating the bounding rectangle and/or intersection of a visual item with the viewport.

2.4.6 Portals, Visualisations and Data

So far, various techniques to provide an intuitive navigation have been discussed. There is, however, still a need for a coherent structure that ties the information items - which are still cluttered in the zoomable landscape - together. To be able to display qualitative and quantitative relations of objects, the very basis for analytic exploration, König proposes a system of ”frames” [Kön06]. A frame, which in the meantime got to be called a ”portal” and will henceforth be referred to as such, is basically a non-exclusive container for items of a certain domain of interest that is itself embedded in the information landscape. The organisation of these items can be changed freely using modular ”plug-in visualisations” [Jet07]. To give an example, while browsing pictures, a user could draw a portal around his/her favourite set and arrange them

geographically by choosing a map visualisation, thus creating and ad-hoc structure. Since portals can be nested, the user could further refine the selection by only taking pictures from England into the next portal. What usually takes a complex, spatial query, can be done using simple direct-manipulative techniques. A similar scenario is illustrated in figure 10; portals as used in the iTV prototype are shown in figure 11.

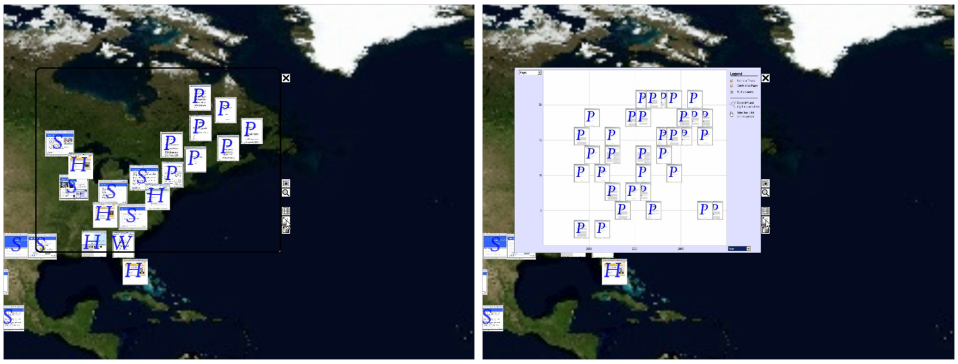

Figure 10: *Nested portals: after zooming into a portal showing a spatial organisation of information (left), a new portal is created using a scatterplot as its visualisation (Source: [JKGR08]).*

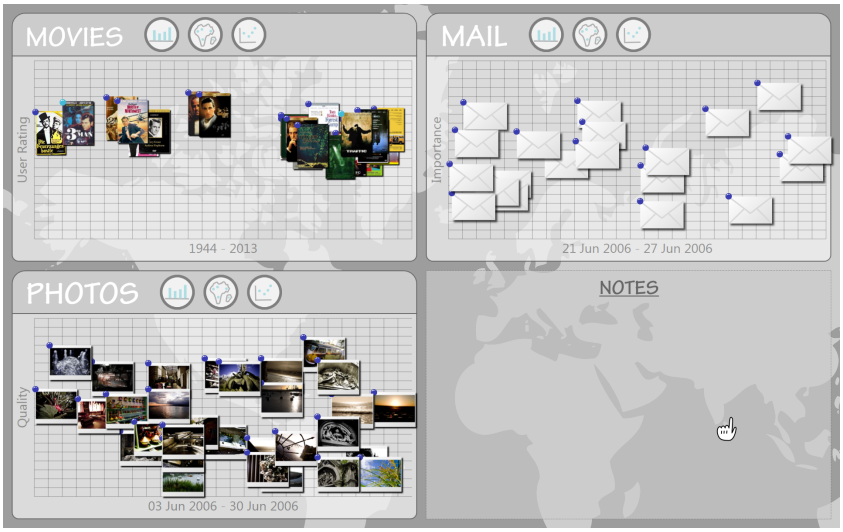

Figure 11: *Start screen of the iTV prototype. Four portals have been created to categorise the information landscape: a movie portal, a mail portal, a picture portal and a notes portal (Source: screenshots of the iTV prototype).*

On the technical side, portals are conceptually similar to an already existing concept in WPF (called "ItemsControl"). Closely modelled after Jetter's description of a hypothetical implementation [Jet07] (the entire reference model is shown in figure 12), portals offer an interface that pipe both information items and input events to a pluggable instance, the visualisation, without having any inherent knowledge of the same. It is a task exclusive to the visualisation to size and position the objects in its available

space. The component layout, including semantic zooming, is still done by each item itself, yet takes context and constraints imposed by the portal and visualisation into consideration.

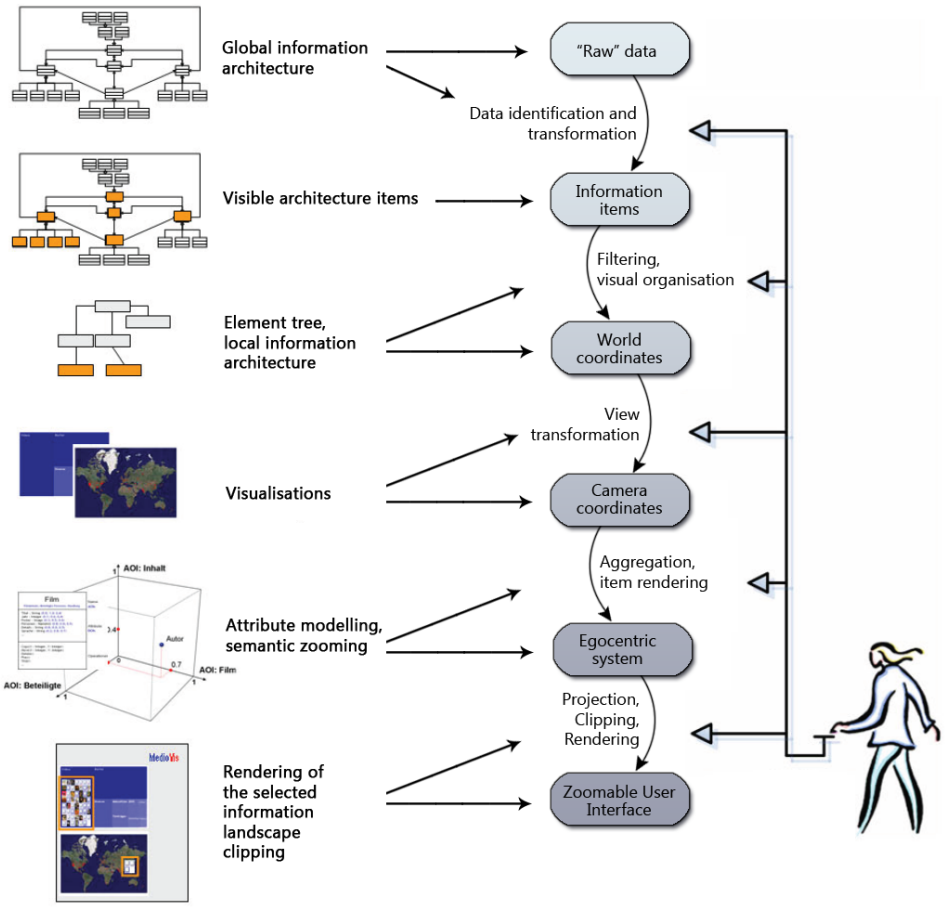

Figure 12: *The ZOIL reference model (Source: [Jet07], translated and slightly adapted).*

The framework uses a coherent system of data contexts that partly stem from WPF implementations. The information landscape typically has the entire, global data set as its context. By organising certain regions of the landscape into portals, each portal receives a subset of the underlying data context (or it might opt to link to an external source, like an Internet image search) which is then used to generate visualisations (which usually rely on meta-information to properly organise information items). In the case of the picture-browsing scenario, each picture might have a title, image source, photographer, and related information. By zooming in closely enough, the embedded photographer information could reveal even more information, such as his/her name, profile, birth place, and so on. Analogous to the element tree concept from chapter 2.4.5, the data context presents information in a tree structure that is used to generate the interface. Filtering of nested information visualisations conse-

quently becomes a straightforward process: leaving out certain items in the piping process automatically prevents them from appearing in subsequent portals.

Although, at this point, the ZOIL framework does not include an implementation of a more complex back-end architecture, a case study for a viable candidate will be discussed later in section 5.4.

2.4.7 History

> *"People make errors routinely."*
>
> – D. A. Norman, The Design of Everyday Things

The international standard ISO 9241-110 ("Dialogue principles") lists error tolerance as one of the ergonomic requirements. Even though it's neither a central topic for ZOIL nor this work, no framework can be complete without having given the prevention and consequence of human error a thought. While a huge potential source of mistakes is already circumvented by a strict prevention of modalities [Ras00] in ZOIL, it does not eliminate the need for a way to undo. In the traditional desktop environment, virtually every program features an undo operation, typically found in the edit-menu or accessed by a keyboard shortcut. This approach comes with several flaws:

- The operation has to be implemented on an application level, there is no global undo supported by the operating system. Thus, it might be prone to programming bugs, poor design choices, and is generally more susceptible to an incoherent user experience.
- Applications rarely decide to save their history beyond their running time, meaning that exiting a program, or worse yet, an unexpected crash may compromise hours of work (or at least render the user incapable of comparing the present state to revisions in the past).
- The scope of an undo operation is typically very narrow. Text editors, for example, usually only provide a history for text changes, whereas settings and the user interface - which are not immune to error - are unaffected.

The presented framework aims to provide a unified undo experience. It is based on König's proposition of a continuous history using a video-like metaphor, where user interaction forms a temporal stream over time, including changes in navigation, the user interface and information items [Kön06]. The framework uses a globally accessible class that keeps track of all user interaction and archives it using the aforementioned three categories, along with a timestamp. While it does not feature an intelligent system that incrementally writes it to a physical hard disk drive yet, it

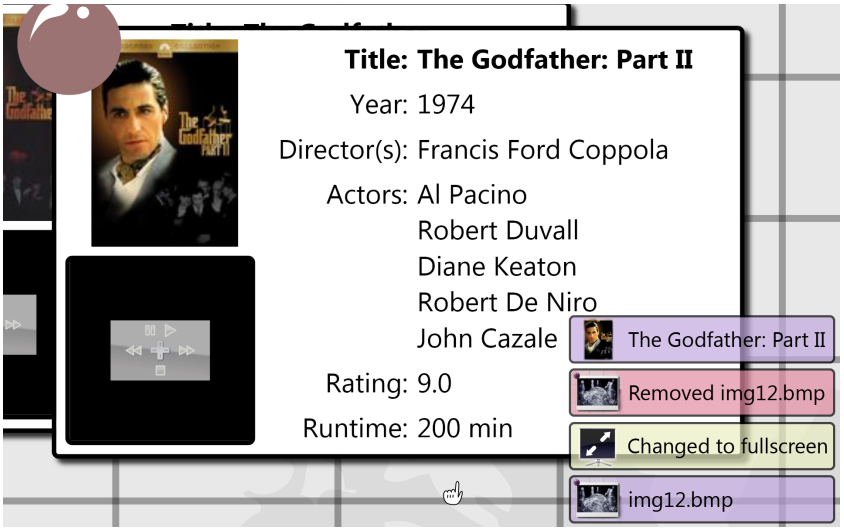

Figure 13: *Mock-up for a hypothetical history implementation. Each of the three categories is colour-coded: blue for changes in navigation, yellow for changes on an interface level and red for changes to an object. Reverting to a certain state in history would be done by clicking on the corresponding item in the stack (Source: edited screenshot of the iTV prototype).*

provides easy to use methods to dump the history on demand which can be used, for example, on shut-down or when an exception occurs. Similar to the CanonCat [Ras00], the environment can then be restored to the exact same state as before, including zoom and position in the information landscape. See figure 13 for an example of how a stack-based interface for the history implementation might look like.

2.4.8 Feedback

"Sites that keep quiet leave users guessing. Often, they guess wrong."
– Jakob Nielsen

Arguably one of the most demanded usability criteria is that of feedback. It is included in Nielsen's "Top-10 Application Design Mistakes" [Nie08], Shneiderman's "Eight Golden Rules of Interface Design" [Shn98] and Don Norman's design principles [Nor04], amongst others. Feedback is essentially about how well the system is communicating its current state as well as making clear how a user's actions have been interpreted. Therefore it is a vital aspect that had to be considered during the framework development.

An inspiration for good design regarding feedback as a response to user interaction was Adobe's Photoshop Lightroom, as shown in figure 14. Whenever an action is triggered that does not provide immediate and self-explanatory feedback itself, such as changing the visualisation from a single picture to the grid view, Lightroom shows

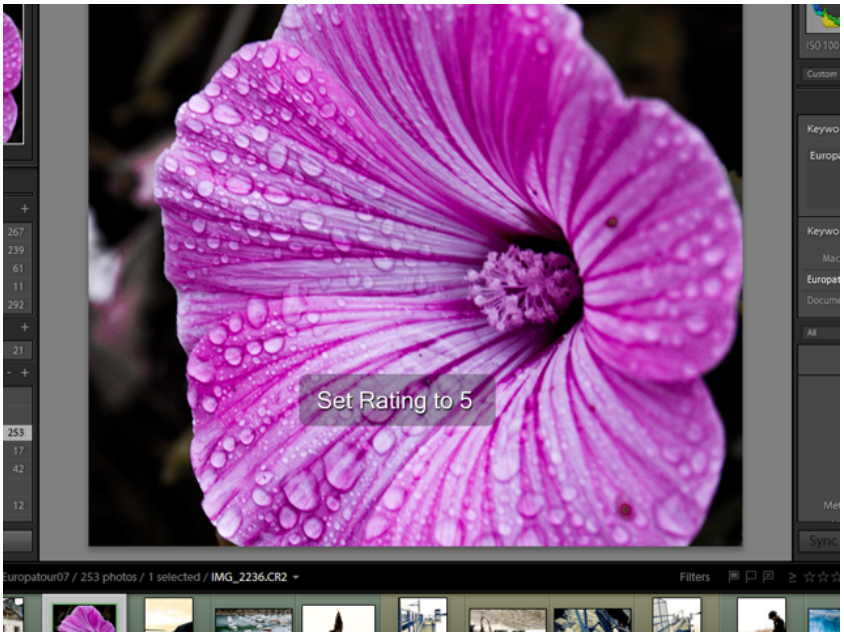

Figure 14: *The Adobe Photoshop Lightroom interface, which served as an inspiration for the framework implementation: a black, semi-transparent rectangle gives feedback to every operation (Source: screenshot of Adobe Photoshop Lightroom).*

a message in the middle of the screen for a short period of time. The notification is non-modal, as opposed to a dialogue box that forces another interaction step for it to disappear. Moreover, Lightroom does not only provide feedback and undo operations for changes to pictures and their meta-data, but also changes to the interface itself. It was therefore influential for the design of the framework's history implementation (see chapter 2.4.7) as well.

Apart from providing consistent feedback to operations, the framework offers an easy way to put any component of the interface into a working state whenever it is about to do something time-consuming. An important guideline for developing environments using the ZOIL metaphor is to use multi-threading wherever possible to prevent an operation from forcing the entire ZUI into a non-responsive state. Therefore a uniform visual indicator is used to show the parts of the landscape that are currently blocked from interaction. An example for a portal that is currently calculating the layout of its visualisation and is thus showing a loader is given in figure 15.

One last use of feedback that shall be discussed is concerning zooming. The ZOIL paradigm aims at more than mere exploration, therefore navigating to parts of an interface that are not populated by information items yet sometimes becomes necessary to extend the information landscape [Kön06]. When, in such a situation, the user is confronted with nothing but a statically coloured screen, the term "desert-fog" (see chapter 2.4.1) truly applies: it's even impossible to tell whether the system is currently zooming in or out or moving at all. The framework therefore introduces

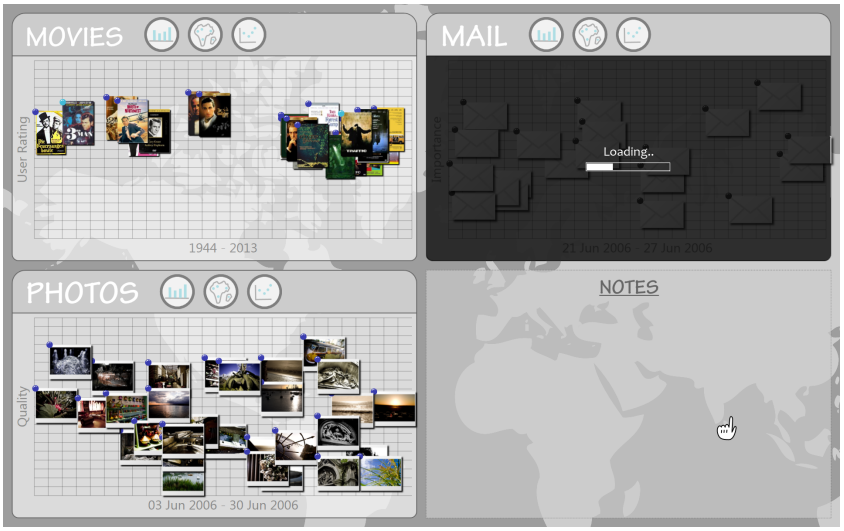

Figure 15: *Start-screen of the iTV prototype. The mail portal in the upper right is currently calculating its layout after the user triggered a visualisation change. The progress bar was added for illustration purposes, but is not yet technically realised (Source: screenshot of an experimental version of the iTV prototype).*

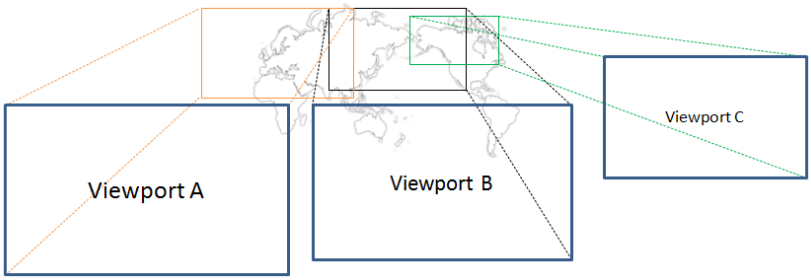

Figure 16: *A world map is chosen for the information landscape's background. After various steps of zooming and panning, three different viewports (on the entire landscape) show a distinct clipping of the background that can be used as "landmarks" for orientation.*

a technique called "parallax zooming" which uses a second layer featuring a clearly visible shape, geometrically zooming according to the foreground zoom level, yet decreased by a constant factor (see figure 16 for an illustration). The background, apart from being decorative, consequently becomes both a means of orientation and clearly indicates any zooming or panning regardless of desert-fog.

2.4.9 Overview

While ZOIL excels particularly in retaining the context of a document since all functionality can be accessed without the need to switch applications, when navigating a landscape containing thousands of information items a user might still be in need of an overview. Since zooming out all the way back to a point where the entire structure

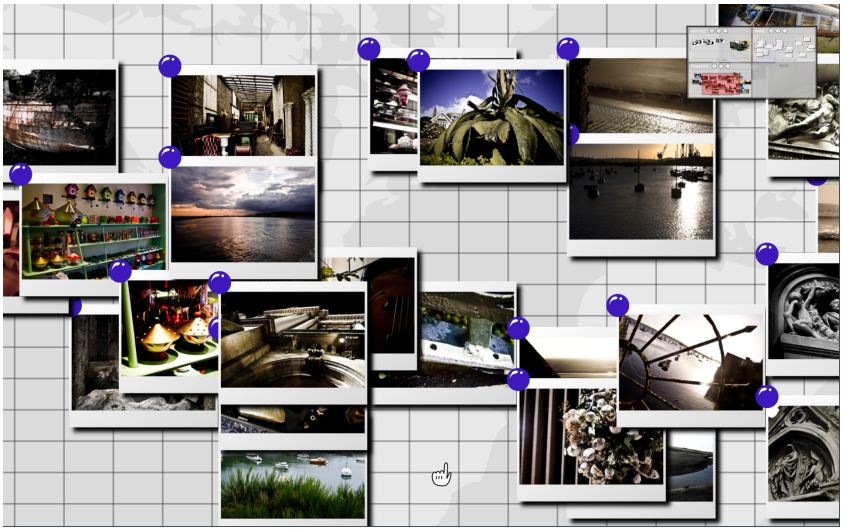

Figure 17: *When zoomed closely onto a portal, the overview as shown in the right top corner can prove useful. The coloured rectangle reveals the currently active viewport (Source: screenshot of an experimental version of the iTV prototype).*

is visible again can be cumbersome, especially when the document is deeply nested within the landscape, different solutions have been proposed. König [Kön06] suggests a navigation technique that can be activated by say, pressing a certain keyboard button, and will quickly take the user back to the top-level view of the information landscape. After releasing the button, the interface automatically navigates back. A real-world analogy would be jumping while being surrounded by a crowd of people to get a quick glimpse at one's location.

A different technique reviewed by Plaisant et al. [PCS95] is the use a single coordinated pair, also known as "overview-detail". It is typically implemented by reserving a small screen space for a global view, showing a rectangle denoting the detail view position. The concept was first introduced Donelson as a navigational aid for his "data surface" [Don78]. By adding an intermediate view, this idea can be extended to a tiled multilevel browser.

By using a combination of directed zoom onto the top-level of the information landscape and the history (as discussed in chapter 2.4.7), König's variant can easily be implemented. To reasonably support all mentioned variants, the concept of cameras had to be introduced to the framework. Cameras act like every other visual object, having a width and height. If they were to be placed in the information landscape they'd even zoom as expected. Once given a virtual landscape position they will render whatever is depicted in that location - just like the main viewport does - thus they are not restricted to showing an overview. This makes them a viable solution to show feedback of actions that affect items that are not currently visible. Moreover, cameras can be infinitely nested. A restriction with the current implementation is that they do

not interpret input events, meaning that it provides no functions that allow camera interaction, such as panning the screen using the overview. Furthermore, cameras are required to show the same component representation as the main viewport. Reasons for these limitations as well as approaches to work around them will be further discussed in chapter 2.5.2. See figure 17 for an example of a camera displaying an overview.

2.4.10 Direct Manipulation

”Direct Manipulation” is a term that was originally coined by Ben Shneiderman and generally refers to a system with three properties [Shn97]:

1. Continuous representation of the object of interest.
2. Physical actions or labelled button presses instead of complex syntax.
3. Rapid incremental reversible operations whose impact on the object of interest is immediately visible.

Put into simpler terms, direct manipulation is about changing the ”programming” of an object by visual means, such as moving icons, clicking buttons or performing drag & drop operations. It tries to abstract the interaction with data, away from ”hidden operations” or ”command names to learn” onto a set of graphical operations that matches how a user thinks about a problem.

Direct manipulation interfaces bring forth various virtues, such as increased learnability which can usually be done most efficiently by mere demonstration, but experts do also profit from the ability to rapidly perform wide range of tasks [Shn87]. Providing instant feedback, users can tell whether a set of action is furthering their goal and reverse their actions if it doesn’t to change the direction of their activity. Moreover, direct manipulation usually obliterates the need for error messages.

Hutchins et al. further explain direct manipulation on a cognitive level by introducing two aspects of directness, one of which is ”engagement”, the ”distance between one’s thoughts and the physical requirements of the system” [HHN85]. Direct engagement occurs whenever users experience immediate interaction with the objects in a domain, making them feel involved rather then giving them the feeling of communicating with an intermediary. Being an object-oriented paradigm, or, what in terms of Hutchins et al. would be described as a system built upon the model-world metaphor, the use of direct manipulation becomes essential for ZOIL as it helps bridging the gap (better known as the ”Gulf of Execution” and ”Gulf of Evaluation” [ND86]) between the information landscape, the action a user can perform, and thus ultimately the user’s goals.

Figure 18: *The left picture shows the iTV prototype in use with a Wiimote. To the right, the same interface is displayed on a multi-touch table.*

While one can not truly ”implement” direct manipulation as it is largely this aforementioned feeling of directness that accounts for it, the ZOIL framework offers a two technique that simplify its use on a component and application level. Firstly, there is an in-built support for dragging and dropping information items. As this is an operation that is overly complicated by WPF due to the requirement for inter-operation with older Windows APIs, the ZOIL framework abstracts this functionality so that it can simply be attached to an information item. Whenever a drag or drop is initiated, the corresponding events will be triggered, delivering information about the items in question and their data context. Furthermore, a small visual indicator displays a miniature version of the information item which can optionally be changed when hovering certain objects, giving immediate feedback about the operation that is about to be triggered. This is actually a use of the previously mentioned camera implementation (see chapter 2.4.9), thus if a movie player was to be dragged, the representation would continue streaming the media as well. An example of a drag & drop operation in progress can be seen in figure 30.

Secondly, there are various implementations of what in WPF is called an ”adorner”. These are layers that, when included at a certain position in an element hierarchy, add certain functionality or visual indicators to its logical children. To illustrate, one of these adorners (typically placed at the root of a component) enriches elements by drawing small circles at each corner which, when dragged, allow it to be scaled and rotated. It was previously shown in figure 7.

2.4.11 Styles and Themes

”Attractive things work better.”
– D. A. Norman, Emotional Design

Especially during the design of prototypes for evaluation, design is often given a secondary role. It is, after all, the functionality that is to be tested. The truth, however,

is that there is indeed a correlation between appearance and usability [Nor04]. The stress on visceral design becomes even greater when a prototype is designed for demonstration purposes or end-user applications and environments.

Consequently, an adequate support for themes became yet another benchmark for the framework. Since XAML, with its various design tools, allows both designers and programmers to work simultaneously, the framework aims for a clear separation of visuals, which are exclusively handled using pure XAML templates, and logic, coded in C#. Moreover, templates are not tightly coupled to controls, thus different designs can easily be replaced even during run-time, which greatly simplifies testing design alternatives. To support consistent use of font-size, colours and various other properties, styles are employed, mapping these values to a set of identifiers. As an example, the RGB code of a colour that is used throughout the interface could be given the name ”ForegroundColour” once, so that a change in the colour is reflected in every instance where the style is referenced. Figure 19 shows an example for the theme-mechanism as employed in the iTV prototype.

2.4.12 Input Abstraction

Typically, in the desktop world input is assumed to be mouse and keyboard. Although attention to multi-modal interaction is certainly on the rise, primarily, but not limited to the increased popularity of multi-touch devices such as the Apple iPhone[16] and Microsoft Surface[17], at the time of writing none of the more widespread ZUI frameworks support these without additional effort by the application developer.

Even though WPF, too, doesn't come with built-in support for more complex interaction such as multi-touch, its input architecture provides enough flexibility to still utilise it in own implementations. Since all elements in WPF are part of a tree, input events can be routed using either the ”bubbling” (notifying the element that caused the event first, then its parent, and so on) or ”tunnelling” (starting at the element tree's root and working towards the source element) strategy [Mic08a] (see figure 20).

The ZOIL framework design builds upon WPF by implementing various more generic events, such as ”pointing”, along with a multitude of input devices that raise these. By convention, one event originates at the component and bubbles towards the information landscape, while a corresponding ”preview” counterpart travels the opposite way. These can optionally suppress the regular event from being triggered by marking it as handled, a mechanism that is useful in cases where an event is substituted by another, carrying more precise meta-data. As an example, a multi-touch preview event registered in the information landscape might have identified the gesture as a

[16] http://www.apple.com/iphone/
[17] http://www.microsoft.com/surface/index.html

Figure 19: *The already familiar start screen of the iTV prototype is shown on top. After the press of a button, part of the layout (such as portal colours and background image) are changed during run-time by simply exchanging themes (below) (Source: screenshots of the iTV prototype).*

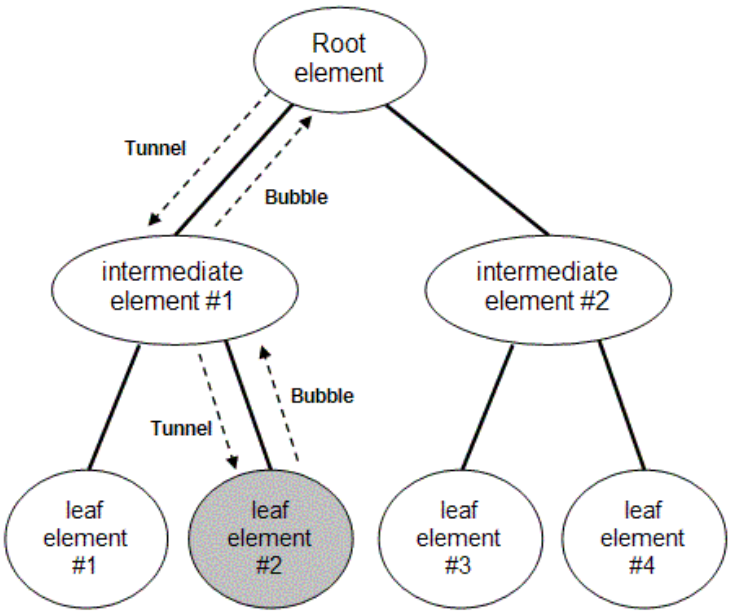

Figure 20: *The bubbling and tunnelling concept in WPF (Source: MSDN).*

scaling gesture (where two fingers are pointing at the edges of a component). Rather than sending the raw input information, the landscape replaces the original by a scale event with information about the delta between the current and last state.

By default, the framework already provides an input configuration for each device, thus regular demos and prototypes typically do not have to deal with input at all, other than specifying what devices to register. Custom input handling can be introduced at application, as well as component level to ensure maximum flexibility.

2.5 Limitations

This chapter outlines all known trade-offs and difficulties that do not have a generally viable or satisfying solution for a framework implementation, yet. While certainly not universally complete, these are the only technical design problems encountered in scenarios in which the ZOIL framework has been used as a foundation for so far.

2.5.1 Browser Integration

Arguably the biggest limitation of the framework is its lack of support for integrating browsers, since they have become more and more important as ”cloud computing” rises in popularity [Eng08b]. Even though WPF offers a very simple way to embed a browser frame, it is a non-native component that does not use WPF’s own rendering pipeline, but falls back to traditional GDI. Consequently, it is not applicable for most of

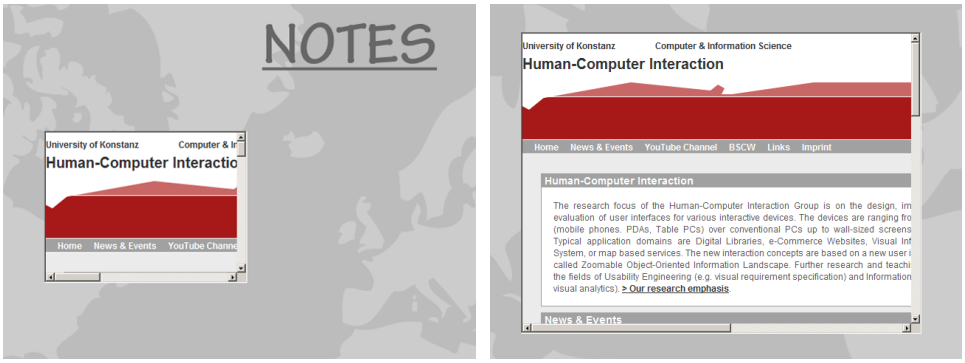

Figure 21: *The left shows the iTV notes section with a browser object displaying a website. After zooming, the browser frame increases in size, whereas its content does not scale (Source: screenshots of the iTV prototype).*

its in-built features such as transformations, which the presented framework heavily relies on. When a browser frame is added to the information landscape, apart from behaving sluggishly, only its dimensions scale during zooming; the content, however, remains untouched. Figure 21 illustrates the problem.

There is, however, an open source effort to create an off-screen surface using Windows interoperability and use it as a render target for the browser [jmo07]. The resulting texture is then transformed and displayed in the WPF application, whereas input is routed back to the original browser application. While slow and resource-intensive in its current stadium, this approach has still room for improvements on both, the framework's, as well as the control's side.

2.5.2 Multiple Representations

The fifth requirement for ZUIs as listed by Bederson et al. demands multiple representations for objects, so that they can be "rendered differently in different contexts" [BMG00]. Mainly in order to provide an efficient and simple-to-use prototyping environment, the ZOIL framework uses its visual hierarchy to map logical relations as well (see chapter 2.4.5), as opposed to using a scene graph-like concept where the tree merely links to instances of an interface element. Although intuitive, the approach doesn't fare well in two cases:

- An instance of a component appears multiple times in the information landscape. WPF limits each element to have a single place in its element tree, thus the only feasible way to have an information item occur multiple times in the same landscape is to duplicate it, manually taking care of synchronisation.
- An instance of a component has to be rendered differently by two views. An example would be an overview: as there is only one instance, we can not simply have two different semantic representations at the same time. Even though

the two views might look at the same item from different distances, their appearance is identical.

The first case, while certainly complicating some cases, doesn't actually pose a significant problem in the overall framework design as information items are meant to be generated from data sources and therefore get their data context from a single instance (see chapters 2.4.6 and 5.4 for more information). Thus, there are no inconsistencies by having several component-instances, even though they represent the same object.

The latter problem could be circumvented by calling the components rendering function twice, passing information about the context so it can adapt its representation. WPF, however, uses a concept called "retained mode" for rendering [Mic08d]. While in classic "immediate mode" the application is responsible for invalidating and repainting parts of the screen, retained mode offers increased usability (and, in most cases, performance) by delegating the task of handling repaint requests to the underlying system. The application, in turn, models the visual appearance by defining a set of serialised drawing data. Thus, WPF offers no direct control over rendering. Propositions to work around that issue will be discussed in chapter 5.5.

While obviously limiting the flexibility and capability of cameras, being unable to render multiple representation complicates the implementation of a visualisation technique first introduced by Bier et al., the so-called "magic lenses" [BSP+94]. Magic lenses render specific regions of the screen differently, supporting tasks as simple as magnifying parts of the landscape to changing layout and alignment of various information items. However, as König points out that ZOIL's concept of portals and visualisations are an analogy to magic lenses [Kön06] and these do not suffer from the aforementioned difficulty as separate instances are used for rendering. Consequently, magic lenses are less of a problem as most of their applications can be implemented as visualisations, to the same effect.

3 Comparison to Other Frameworks

3.1 State of the Art Analysis

An extensive state of the art analysis concerning historical projects such as SDMS, Pad and Pad++, as well as various ZUI techniques has already been given in König's master thesis [Kön06]. This work will therefore focus on the latest incarnations of toolkits and frameworks for building zoomable user interfaces.

Piccolo is the most recent version of the toolkit initially known as Pad [BGM04] and is built using Java, yet a port for .NET 1.1 is available as well. Although Piccolo is, at the

time of writing, about to be superseded by Piccolo2D [Goo08] this document focuses on the more establish predecessor. Piccolo.Java uses Java2D for rendering and runs on various platforms including Windows, Linux and Solaris. Piccolo.NET is based on GDI+ and is purely written in C#. There also exists a version of Piccolo.NET for PDAs called PockedPiccolo.NET, using the Compact .NET Framework. It is the longest running effort of a ZUI toolkit and undoubtedly the most widely spread. Moreover, due to its focus on fundamental ZUI-related features such as a scene graph implementation, it is highly versatile (by making few assumptions) and likely the best candidate for use with ZOIL. Therefore it will be analysed more thoroughly in the following sections.

Another popular and established toolkit is ZVTM[18] [Pie05], formerly a project developed in the Xerox Research Centre Europe under the name XVTM[19]. Other than Piccolo, ZVTM aims to provide an application programmer with ”building blocks for implementing complex [...] interface components” and thus comes with many prebuilt visualisation and navigation components. It is grounded on the metaphor of virtual spaces, theoretically infinite universes, that are observed by cameras and populated by geometric objects called glyphs. ZVTM implements a wide range of features such as input event handling, fish-eye lenses, rate-based scrolling, speed-dependent automatic zooming, semantic pointing, and more, while taking care of low-level graphical operations like clipping and animation management. Moreover, ZVTM offers hardware acceleration for rendering using various different methods, such as Java Volatile Images. Lacking a scene graph implementation comparable to the one in Piccolo, however, performance quickly becomes a problem in complex scenes, which might also be the reason why there is no support for anti-aliasing or other mechanisms devised to improve rendering quality. Furthermore, judging from its documentation and demo applications, ZVTM seems to have no understanding of media types other than basic image formats, nor does it have a browser component or other means of simplifying Internet communication. Consequently, it would still require substantial effort to use ZVTM as a foundation for a ZOIL implementation.

ORRIL[20] on the other hand takes a highly theoretical approach, aiming to aid the design and creation of ZUIs [Med04]. This is accomplished by abstracting a ZUI into four different types: objects (information items in the landscape), regions (three-dimensional areas), interface logic (representing transforms) and relations (mappings between regions, objects and interface logic). The exact process will not be outlined here, and it is only listed in this analysis as ORRIL has given birth to a practical framework called Nutmeg.

Nutmeg is an ”experimental rapid prototyping tool [...] for creating medium to high fidelity ZUIs” [Ben05]. Its aim is to set itself apart from low-level framework that re-

[18] Zoomable Visual Transformation Machine
[19] Xerox Visual Transformation Machine
[20] Objects, Regions, Relations and Interface Logic

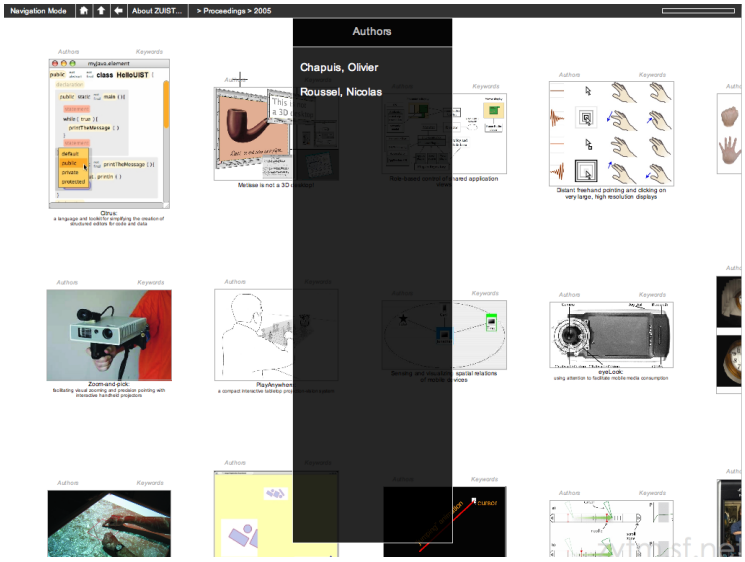

Figure 22: *A screenshot showing ZUIST, a multi-scale interface for navigating in 20 years of papers published at ACM UIST and implemented using ZVTM. Publications are browsed via zooming (Source: `http://zvtm.sourceforge.net/`).*

quire expert knowledge by introducing a specifically tailored mark-up language called ZIML[21] to generate ZUIs, much like a web browser creates websites from HTML[22], and not unlike XAML used by the ZOIL framework. Interface and programming logic is then added using a range of scripting languages. ZIML integrates a wide range of different data types (including various image and audio formats) and works on any platform supporting Java applets or applications. What sounds like a promising toolkit seems to be defunct in practise: on Bennett's official website[23] Nutmeg is listed as a "software artifact"; no download link is provided.

Eagle Mode is a purely practical project and thus is not backed up by any scientific publications. It claims to be "an advanced solution for a futuristic style of man-machine communication, in which the user can visit almost everything simply by zooming in" [Ham08]. It uses "panels" to structure its interface, which, translated into ZOIL terminology, refers to a hierarchy of visualisations. Eagle Mode already integrates a file manager, file viewers for common file types, as well as various games, but can be extended by using a portable C++ API[24]. At the time of writing there was no documentation for scenarios other than installation and navigation, however. From what can be seen, it seems to solely support file system navigation, whereas ZOIL heavily relies on content taken from various web services. Furthermore, it only supports the Linux operating system, thus is it definitely not a viable candidate.

[21] Zoomable Interface Mark-up Language
[22] Hypertext Mark-up Language
[23] `http://www.stressbunny.com/mike/research.html`
[24] Application Programming Interface

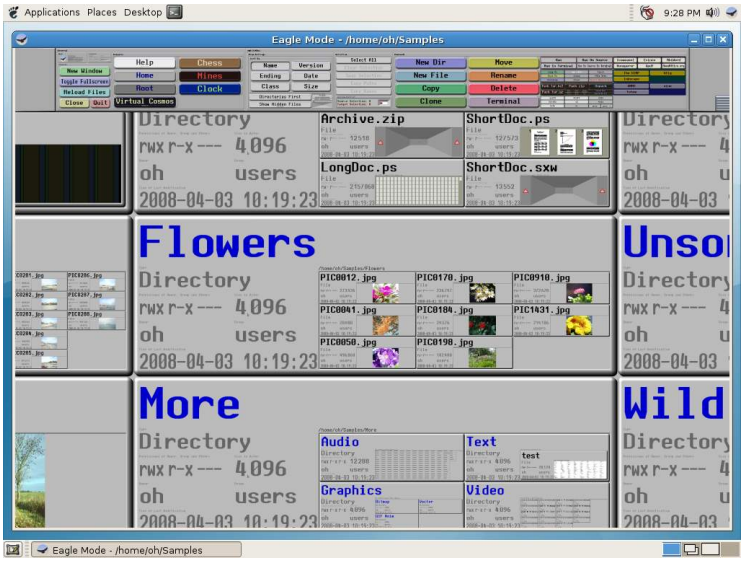

Figure 23: *A screenshot showing Eagle Mode's file manager running on Linux. The top panel exposes various functionality while the zoomable panel below is used to explore the file system (Source: `http://eaglemode.sourceforge.net/`).*

While this list does not claim to be complete, it certainly represents the sparse landscape of ZUI frameworks, and none of the aforementioned developments were deemed suitable for a ZOIL implementation. The following section picks the toolkit that comes closest to the requirements imposed by ZOIL and intends to clarify why yet another implementation was inevitable. Moreover, it aims to be a mostly technical comparison that should serve as a roadmap for the ZOIL framework as to where improvements are still necessary. It is not, in any way, a qualitative comparison meant to identify the superior framework.

3.2 Piccolo

Both Piccolo and the presented framework claim to support the creation of zoomable user interfaces, yet they differ greatly in terms of their primary focus, implementation philosophy and structure. Thus the first step will be to provide a proper outline of what can be compared, before looking at the actual comparison in the subsequent sections.

3.2.1 Establishing a Common Ground

As stated by its manual, "[a] primary characteristic of Piccolo.NET is that it is designed to support zoomable information spaces, although any particular applications may or

may not take advantage of this feature", thus focusing on "features such as zooming and multiple representation" and "the ability to create, manipulate, and render object-oriented graphics" [HCI08]. It aims to provide a reasonable real-time performance using "a tuned run-time system to render the scene graph as quickly as possible to support interactive applications". Piccolo's namespace structure reveals the following categories:

- Piccolo's *core* mostly provides cameras for rendering a scene, canvases for managing the scene graph and the basic node class which denotes an object that is part of the scene.
- Activities *feature* animations that can be assembled to form entire storyboards, interpolating various properties such as colours or transformations.
- The *event* namespaces simplifies implementation of forms, navigation as well as direct manipulative interaction such as drag & drop.
- The *node* namespaces is a collection of basic, pre-built node derivations, such as images or pathes.
- Lastly, the *util* namespace is a collection of miscellaneous classes, for example math helpers for matrices, serialisation mechanisms, rendering and debugging helpers.

Whereas the presented framework is built to only support the ZOIL paradigm, Piccolo aims to be a general framework for innovative interface design with zoomable information spaces in mind, although applications may choose not to take advantage of that feature at all [HCI08]. As many of Piccolo's core features have already become a part of the WPF platform in comparable ways, like the element tree (see section 2.4.5) as an alternative to Piccolo's scene graph implementation, as well as support for animations and vector graphics, the ZOIL framework aims to provide an asset of components, visualisations, input devices and visual themes instead; neither of which are inherently part of Piccolo.

As a consequence, the focus of both frameworks differs greatly. Using the specifications and documentations combined with general aims for framework projects and zoomable user interfaces, the following is a list of generalised criteria that shall be used as a common ground for the subsequent comparison.

- **Ease of use:** How big is the effort to create an interface, custom component and layouts?
- **Display and rendering quality:** How well does anti-aliasing perform in various situations? Does the interface scale nicely?

- **Performance:** How do the rendering times behave when loading massive amount of images, videos or interface elements? How much memory is used?
- **Documentation:** How well is the API documented?
- **Features:** How mature is the repertoire of non-traditional concepts?

Again, it is stressed that a new implementation of Piccolo is currently in development and promises to provide better performance since it is built entirely on Java2D. At the point of writing, however, the framework version that was officially maintained by the HCIL group of the University of Maryland is still the most widespread and best documented version, and is thus the one used for comparison.

3.2.2 Comparison: Ease of use

The first step shall be to set up a basic information landscape containing a single item and attach a mouse for basic input. In the case of Piccolo, a regular .NET 1.1 form has to be created that derives from a special PForm class. At the root of every application there is a canvas that maintains a tree of nodes deriving from the PNode class, consequently the next step is to add a simple rectangle node, colour it and add it to the canvas. Lastly, an event handler is attached to that node to handle directed zooming. The bare scene set-up code is therefore as follows:

```
// Create a node (320x240 points, red)
PNode node = PPath.CreateRectangle(0, 0, 320, 240);
Canvas.Camera.AnimateViewToCenterBounds(
    e.PickedNode.GlobalBounds, true, 500);
```

Free zooming as well as panning (both using the mouse) are enabled by default and must be actively suppressed if they are not wanted.

The ZOIL framework requires the user to manually attach input devices but provides a default configuration for each. The red rectangle is declared in XAML and flagged as a zoom target, which concludes this example. The demo's procedural part is therefore reduced to a single line of C#, while the equivalent ZOIL program is accomplished with following markup:

```
<Components:ZInformationLandscape x:Name="Landscape">
    <Rectangle Width="320" Height="240" Fill="Red"
     Components:ZInformationLandscape.ZoomTarget="True"/>
</Components:ZInformationLandscape>
```

And the accompanying procedural code:

```
ZOILFramework.RegisterInputDevice(
    Landscape, ZInputDevice.Mouse);
```

At this stage, both frameworks provide an equally simple entry. Piccolo, as well as the ZOIL framework, do not require a any manual initialisation and integrate seamlessly into existing, IDE-generated[25] Windows forms and WPF code, respectively.

As a consequence of building heavily upon the existing .NET framework as a foundation, the ZOIL framework does however gain the advantage of utilising the more modern Windows Presentation Foundation, part of .NET 3.0 and above. This becomes especially apparent in the ease of use aspect as WPF supports declarative interface modelling (which becomes increasingly frequent with modern languages). The use of declarative XAML allows for an easier interface, control and component creation using WYSIWYG[26] editors. In the case of Piccolo, custom controls have to be assembled, positioned, coloured and transformed entirely in code as there is no editor with a native understanding of the PNode class controls have to derive from. We shall soon see the advantage thereof in the next code sample.

Another important aspect is embedding already existing controls (such as buttons) to the applications. Piccolo limits the allowed controls in its scene graph to classes deriving from PNode. Consequently, native .NET controls have to be wrapped using a special PControl construct. Internally, PControl renders the control to an image which can then be scaled and rotated, thus they can only be interacted with when they are shown in their original size. Piccolo works around that problem by a special input handler that takes the camera to an appropriate position whenever a control is clicked. This obviously still imposes a severe limitation: if controls were to be used to, for example, provide filtering for data in a visualisation, clicking on an input element designed to adjust a filter criterion would first zoom onto it, leaving the context and making it impossible to tell the effects of the update. The following code is necessary to add a single, click-able button to a Piccolo application:

```
// Create a button Button button = new Button();
button.Text = "Hello";
button.BackColor = SystemColors.Control;
// Wrap the button in a PControl and
// add it to the scene graph
PControl control = new PControl(button);
control.SetBounds(0, 0, 100, 50);
// Add the control to the root layer
Canvas.Layer.AddChild(control);
Canvas.Camera.AddInputEventListener(
```

[25] Integrated Development Environment
[26] What you see is what you get

```
        new PControlEventHandler());
```

The ZOIL framework, on the other hand, does not require the landscape to only contain special controls as long as they are native to WPF[27]. The equivalent code for above sample is thus (only XAML, the code remains unchanged from the first sample):

```
<Components:ZInformationLandscape>
    <Button Width="100" Height="50">Hello</Button>
</Components:ZInformationLandscape>
```

Summarised, while both frameworks offer a roughly equally comprehensive entry, the ZOIL framework is clearly more up to date with recent development and thus leverages the advantages of modern APIs for easier creation of more complex content. Furthermore, the ZOIL framework simplifies the usage of existing controls greatly by allowing them to be directly embedded into the information landscape. As the framework was built with rapid prototyping in mind, this was one of the main reasons why using Piccolo was not a viable option.

3.2.3 Comparison: Display and rendering quality

Piccolo.NET uses GDI+ for rendering and tweaks its performance by adjusting rendering settings depending on the application's state, which can be "interacting" (during, for example, panning, which is not animated) or "animating" (while zooming, for example). The lowest common denominator is taken if both states are active at the same time. Affected rendering settings are interpolation mode, smoothing mode, text rendering hints, compositing quality (for images) and pixel offset mode, but they're only accessible by three preset modes, low, medium and high quality. Quality settings affect the application globally.

The ZOIL framework does not use any custom rendering mechanism as WPF already features an Element Tree (see chapter 2.4.5) that handles geometry clipping, dirty region updates, interpolation, anti-aliasing, transformation and similar computer graphics concepts [Mic08d]. Default WPF methods can be used to change rendering behaviour for certain parts of the tree, as well as for the entire application, which includes interpolation quality and caching hints. The framework, however, does not require automatic switching between rendering settings dependant on the application state as all rendering tasks are delegated to a low-level DirectX implementation that heavily utilises the GPU[28]. Thus there is no noticeable slowdown on tier-2 hardware (supporting at least DirectX 9.0) [Mic08e].

[27] Non-native controls (such as a browser) are rendered, but they can not be rotated and scaled. This limitation is explained in more detail in chapter 2.5.1.

[28] Graphical Processing Unit

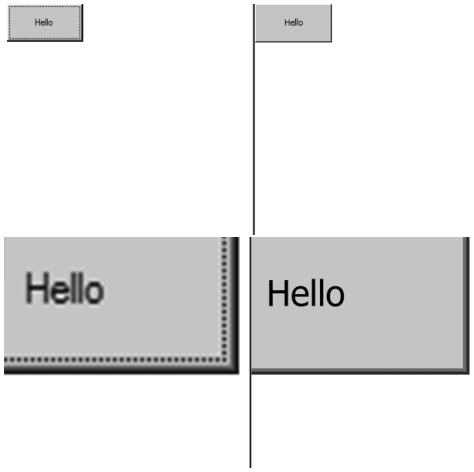

Figure 24: *A simple button demo, both in Piccolo (left) and the ZOIL framework (right) before (above) and after (below) zooming in.*

The fundamental difference between both approaches is that Piccolo is fundamentally pixel based, whereas ”WPF uses vector graphics as its rendering data format” [Mic08d]. While this also has implications for device independence, the following focuses solely on the quality aspect. Figure 24 shows the button-example from chapter 3.2.2 as displayed by Piccolo (to the left) and the ZOIL framework (to the right), both unscaled (above) and zoomed in (below). As Piccolo interpolates the image the control has been rendered to, the smudged out pixels become visible when viewed close-up. The button in the ZOIL framework is internally defined using vectors, thus quality doesn't degrade after a change of scale or rotation.

During rasterisation, anti-aliasing smudges lines that fall between device pixels, thus straight lines can have varying intensities of colour. Applications using the ZOIL framework can opt to align certain elements to the device pixel grid which greatly improves the quality for rendering shapes, for example grids, as seen in figure 25. The grid as rendered by Piccolo (left) has several lines that appear darker, while others are thicker due to being smoothed out.

As Piccolo, with its default settings, reduces the rendering quality during interaction, the visual appearance of an application can change drastically once a zoom or similar action is initiated. This is especially noticeable with ”dirty regions”[29] during a drag operations that leave a trail of low-quality renderings along the way, as shown in figure 26.

All in all, even when using Piccolo's high quality rendering, the ZOIL framework delivers improved rendering quality due to a native vector and GPU support.

[29]Dirty regions are used to invalidate portions of the screen that need to be redrawn, used to increase rendering performance by not having to refresh the entire screen.

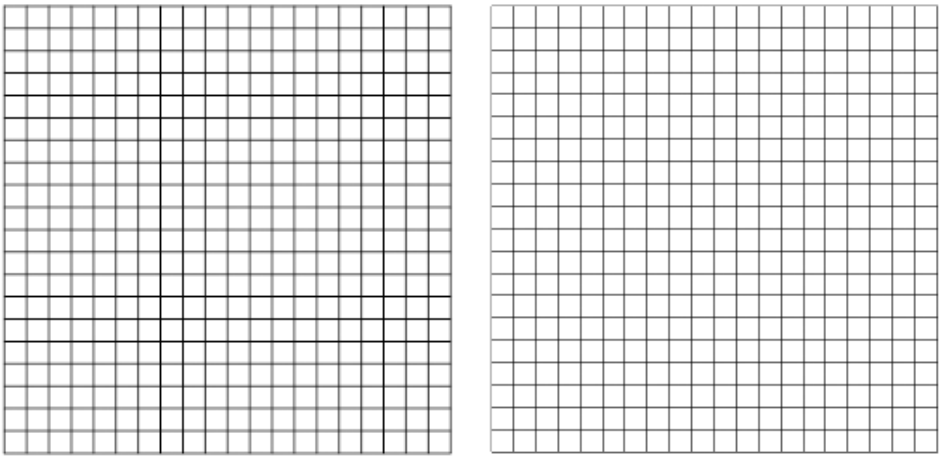

Figure 25: *A grid rendered by Piccolo (left) and the ZOIL framework (right). Be sure to watch the image at 100% of its size to see the difference as the reader-interpolated image suffers from the same issue.*

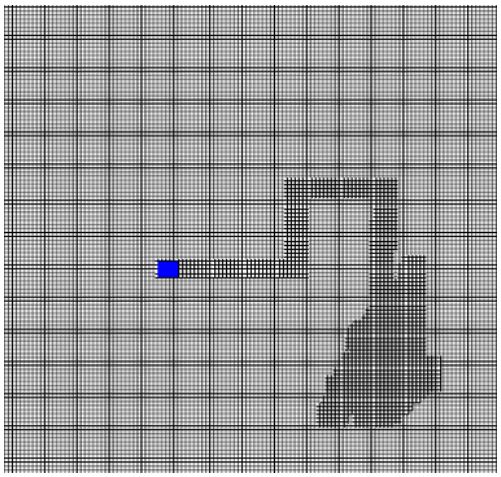

Figure 26: *As the blue rectangle is dragged over the grid it leaves a trail of aliased "dirty regions" (Source: Screenshot of the "GridExample" demo from the "Piccolo Features" demo suite).*

3.2.4 Comparison: Performance

As already pointed out in chapter 2.2, framerates have to be at least ten frames per second for animations to be considered smooth. When filtering, scaling and displaying large quantities of media data this can quickly become a limiting criteria. A required for ZUI platform is consequently that ”interaction performance doesn't degrade with complex scenes” [BMG00].

This section tries to compare Piccolo's performance with that of the ZOIL framework on a very basic level. Tests in three categories are conducted: controls, paths and images. For each category a number of representative elements are added, which is buttons for controls, a rectangles for paths and 1382 to 922 pixel, uncompressed bitmaps (24 bit per pixel) for images.

Both frameworks are compiled in release mode, using a 640 to 480 pixel window. The source code used for comparison is included in appendix D. The machine used for all tests is an Amilo M3438, running on 1.6GHz (single core) with 1024MB RAM and 256MB VRAM, operated by Windows XP, with .NET3.5 SP1 and DirectX 9.0c installed. Memory usage is measured using Process Explorer[30]; framerates are measured by the respective framework.

The test-procedure is the same for all categories. In a regular interval of ten seconds, a set number of items is be generated (100 buttons, 100 rectangles and 1 image respectively). After 5 seconds of each creation cycle a random element is taken as a zoom target for directed zooming. CPU and memory usage, as well as the framerate is measured in intervals of one second and plotted onto a graph. The results can be looked up in figure 27.

Although the tests were performed various times for reliability they have to be interpreted carefully: the test scenarios are oversimplified, various quality settings and performance tweaks are not taken into account and the results are only shown for one particular set of hardware. While there is no ground for the assumption that one framework is generally faster as the other, there is at least a comparison for one particular case that is not too unrealistic.

3.2.5 Comparison: Documentation

Piccolo's documentation is mainly provided in the form of an HTTP class reference, categorised by namespaces, but is also downloadable in form of a compiled HTML file. It is unquestionably thorough and comes with a description for every object in the framework. Since it is only useful for looking up how to use certain objects or methods there is an abundance of demos and full-featured applications with inline comments

[30] http://technet.microsoft.com/en-us/sysinternals/bb896653.aspx

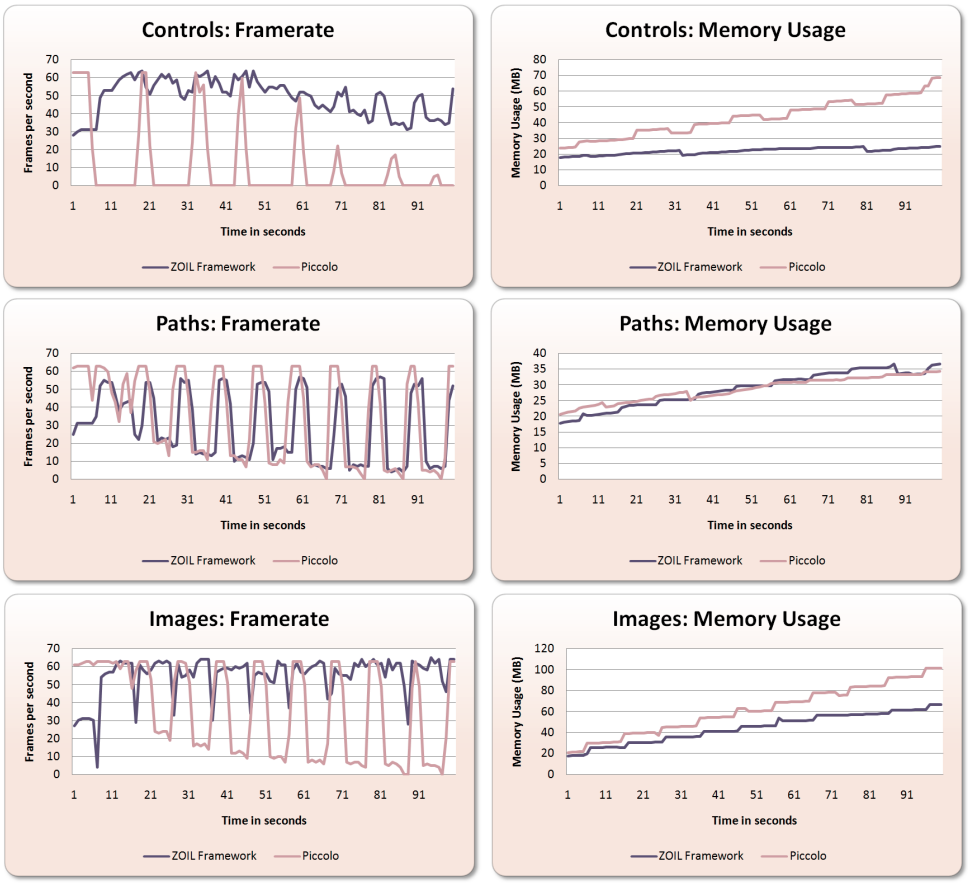

Figure 27: *Frames per second (left column) and memory usage (right column) of Piccolo and the ZOIL framework in three different categories: creation and rendering of* ***controls*** *(top row),* ***vector graphics*** *(middle row) and* ***high-resolution images*** *(last row).*

that are meant to replace an additional step by step guide. It clearly succeeds, as there is hardly a single feature that is not practically demonstrated in Piccolo's demo suite.

The ZOIL framework takes a somewhat different approach, emphasising a Wiki-style documentation (see chapter B) that is designed to primarily convey the design philosophy, as well as the high-level structure behind the overall design. It also includes a section on how to get started with the framework, featuring guides with code samples, and various tutorials on how to extend and contribute to the framework development. There is also a number of demo applications, plus an elaborate prototype (presented in chapter 4) that comes with the framework (see appendix A). Reference documentation is done inline, as comments are parsed and displayed in tool-tips in most modern IDEs and is therefore easiest to access.

Even though the ZOIL framework documentation is backed up by a Wiki that focuses and more than just usage - and thus arguably more versatile - Piccolo, with its myriad of helpful demo applications is undoubtedly more mature. The lesson learnt from using Piccolo is certainly that demos do an astoundingly good job at providing both, an easy entry as well as a solid feature overview and should therefore get reasonable attention in future framework development.

3.2.6 Comparison: Features

Making a head to head comparison of all features and their respective implementations in both frameworks would be far beyond the scope of this thesis, the section is therefore limited to a quick overview over certain aspects that exist in one toolkit, but are missing or comparably non-existent in the other.

Even though Piccolo comes with an abundance of helpful implementations, most of them have already found their way into the more recent versions of the .NET framework, such as storyboards (called activities in Piccolo), a scene graph (called the element tree), clipping, occlusion detection, paths and a powerful layout engine. Where Piccolo truly shines is its handling of multiple representations, allowing for a clean and efficient camera concept as well as powerful magic lenses.

The ZOIL framework sets itself apart by offering a multitude of pre-built high-level components that already come with a visual style, which can be modified without touching the framework code itself. Moreover, many of the concepts of the ZOIL paradigm, as well as regular interfaces, are already tightly coupled with the framework, such as semantic zooming, portals, visualisations, history, direct manipulation. Zooming and panning is already linked to an animation system providing basic physics, easing and a distance-based interpolation and can be utilised by various different input devices.

3.2.7 Summary

By now it should be apparent that the ZOIL framework and Piccolo[31] are not easily compared as much of the implementation work done by Piccolo is already integrated as part of the .NET 3.0 framework, complicated by the fact that Piccolo is not a ZUI-only framework with a much more generalised set of features.

Taking only common ground into consideration, however, the presented framework lags behind in only few aspects as the previous comparisons demonstrate. By leveraging modern hardware, as well as a novel programming framework built with media-rich scenarios in mind, more work can be spent on providing ease-of-use for developers while still providing a competitive performance and rendering quality. This is especially rewarding, considering the fact that Piccolo is still the most widely spread library for ZUIs.

4 Use-case: iTV

This chapter is devoted to presenting the prototype created for the Euro iTV 2008 conference in Salzburg [JESR08]. It shall be noted that the prototype does not, nor is it meant to reflect the entirety of the ZOIL metaphor: many trade-offs and simplifications had to be made to devise a fully running and interactive system that could be operated by anyone without a lot of explaining. It does, however, prove the capabilities of the presented framework, as well as provide a visual specification and basis for tests and future development.

Concepts that stray from those presented in chapter 2.4, or are entirely new, will not be discussed further than the rationale of their implementation; while they might turn out to be lasting ideas, they were merely meant as temporary solutions. It is recommended to watch the introduction video (appendix C) prior to reading this chapter.

4.1 The Scenario

In ”The Humane Interface”, Raskin envisions a zoomable user interface elaborate enough to replace the traditional desktop along with its hierarchical file system [Ras00]. The iTV prototype is meant to be a first step towards this goal: using the ZOIL paradigm as a personal information management environment. As the Euro iTV 2008 conference had a special emphasis on ”changing TV contexts and new entertainment environments” with an emphasis on ”user experiences and needs” [iTV07], the personal information management scenario was taken to a television-like setting in order

[31] The same is true for any other framework mentioned in chapter 3.1

”to demonstrate ZOIL’s benefit for the ITV domain” [JESR08]. Moreover, a ”realistic hardware setting” is simulated by using a 30” HD display and a ”Nintendo Wiimote as remote control and pointing device”.

4.2 Technical Details

The iTV prototype is designed for a specific set of hardware that was used for presentation, more specifically a shuttle PC with a 3.0 GHz dual-core CPU, 2 GB of RAM and an NVidia GeForce 8500 GT graphics processor, accompanied by a 30” Apple Cinema HD display [JESR08]. Regarding input devices, a Nintendo Wiimote and an Apple wireless keyboard were chosen for maximum portability.

Although the prototype is not limited to be used by that specific set of hardware, it was mostly devised bearing these input, resolution as well as performance constraints in mind and therefore offers the best experience with aforementioned set-up.

4.3 Data Sources

Three different sources for populating the information landscape were used, statically linked to the prototype as XML[32] files. Two of them, in turn, reference local files that are not included in the project repository (appendix A) due to their size.

The image data consists of 30 uncompressed images, each 1382 to 922 pixel, 24 bit per pixel, adding up to a total of almost 110 megabytes of pictures that are present at all time. For the map visualisations, two high-resolution images (8192 to 4096 and 4096 to 2048) are used as background. Moreover, poster images of varying resolution are used as movie thumbnails, as well as the feature-length streams of all 26 movie information items, adding up to a total of 47.2 GB of DVD-quality video files.

The remaining graphics (background, portal frames and visualisation buttons) are statically linked vector paths and part of the ZOIL framework’s default theme.

4.4 A Guided Tour

The first screen that will be presented after starting the prototype up displays four portals, as previously shown in figure 11. The data sources (see chapter 4.3) generate information items for three of the portals, movies on the top left, pictures on the bottom left and e-mails on the top right side of the screen. The last, visually different portal acts similar to a clipboard and will be explained at a later point.

[32] ”Extensible Markup Language”

Each portal has a label, as well as three buttons on its top border. The buttons can be used to switch between different visualisations, with the scatterplot being the default visualisation for each portal. Once the timeline or map visualisation is chosen, a different legend will be drawn and the information items in that portal will be rearranged accordingly, as seen in figure 28.

To demonstrate direct manipulative capabilities, the clipboard portal to the lower right can be used to freely arrange information items from all of the other portals by simply dragging it from one of the portals (which can be done with any semantic representation) and dropping it onto the canvas. During that operation, a translucent representation of the item will be drawn to preview the effect after it is dropped. Furthermore, text annotations can be created by simply starting to type text and moving it to the designated position afterwards.

Whenever an item is to be dragged onto the clipboard while it is not visible, the dock - a sidebar-like panel to the right of the screen - can be used, which automatically appears once a drag & drop operation is initiated. It temporarily saves a small number of information items, so that they can be properly arranged at a later point without interrupting the workflow. The dock also contains an item similar to the ”recycle bin” of the desktop metaphor. Items from the notes section or from the dock itself can by removed by dragging them onto this special slot.

All information items feature different semantic representations. For e-mail, the small mail icon is replaced by the mail content as well as a form for replying, while pictures and movies gain additional meta-information after zooming in. Movies furthermore contain a player that streams the feature-length movie, along with a full-screen mode activated by a simple directed zoom, as demonstrated in the earlier figure 5.

To help recognising certain items before zooming in, detailed tool-tips were designed that show a compact meta information summary, superseding the need to constantly zoom back and forth in order to get a better overview. They are triggered by hovering items with the pointer for a short time. This becomes especially helpful when browsing items that are difficult to represent uniquely using icons, such as e-mails. In order to provide feedback to the action of pointing at information items, a visual highlight, as well a tactile feedback is given. The latter is accomplished by subtle vibrations of the Wiimote whenever the cursor hovers a new element.

The iTV prototype was set up for the entire length of the Euro iTV 2008 conference in Salzburg. For two days, researchers, as well as industry representatives from various fields of interests were free to experiment with the demo. Even though it featured only a small subset of the functionality a full ZOIL implementation would have, the feedback was generally positive and hardly a minute passed where the booth would be unattended. Most surprisingly, the statement that the presented concept was meant to be a full desktop replacement was not challenged a single time.

Figure 28: *Using the portal buttons, visualisations can be switched. The left image shows photos stacked by the date they were taken. After choosing the map visualisation, their geographic origin is revealed (below). (Source: screenshots of the iTV prototype).*

Figure 29: *Using drag & drop, content can be put and freely arranged in the notes portal. The notes have been created simply by typing (Source: screenshot of the iTV prototype).*

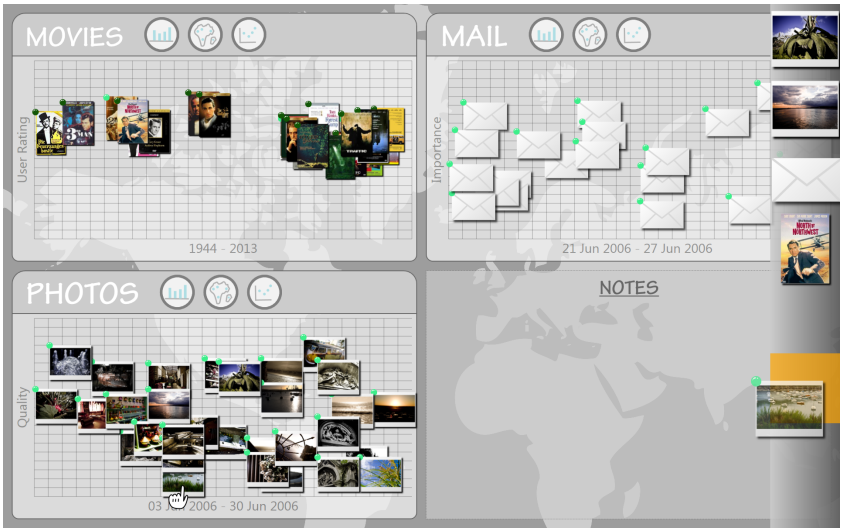

Figure 30: *The dock is used to temporarily hold a number of items for later rearranging (Source: screenshot of the iTV prototype).*

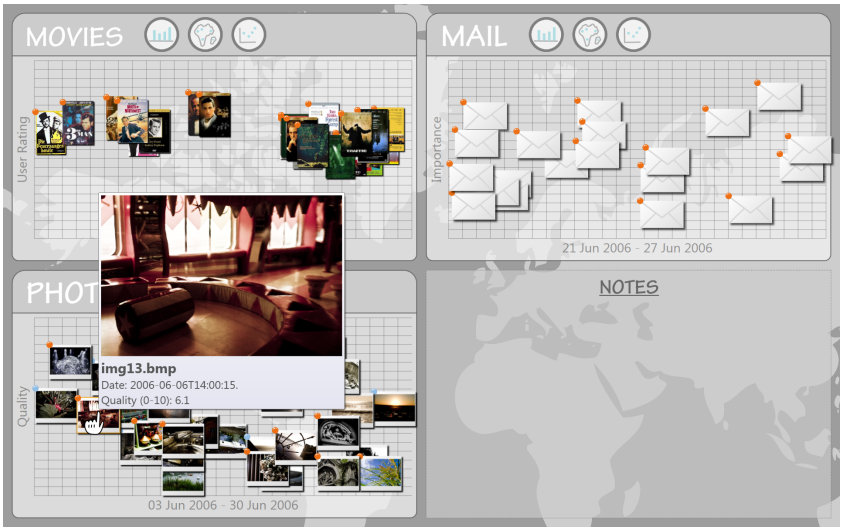

Figure 31: *Hovering an information item reveals a tool-tip, surfacing meta information without having to zoom in (Source: screenshot of the iTV prototype).*

5 ZOIL Framework 2.0

5.1 The Vision

The ZOIL framework has been put to extensive use in the past, ranging from feature demos that were partly used to test or demonstrate multi-touch functionality to the more elaborate iTV prototype and various spin-offs thereof. More recently, the framework has been offered to aid practical implementations of visual information seeking systems in a lecture of the same name, making it the first hands-on by students that were not involved in the development, two results of which are shown in figure 32 and 33.

A lot of design choices have been slightly adapted or proven during these various use-cases. Some, however, have not and are outdated or overly complicated to use. Additionally, new requirements have been identified as the framework becomes of interest by various parties with different aims and interests. This chapter shall highlight various case studies and formulates some of the more immediate requirements and is meant as a roadmap for future framework revisions.

5.2 ZOIL in a Browser

Along with their new framework for interface design, Microsoft has also released a new platform for web development called Silverlight [Mic08b]. Silverlight is a cross-browser and cross-platform implementation of a subset of the .NET framework and features a combination of declarative languages, mostly for modelling the front-end,

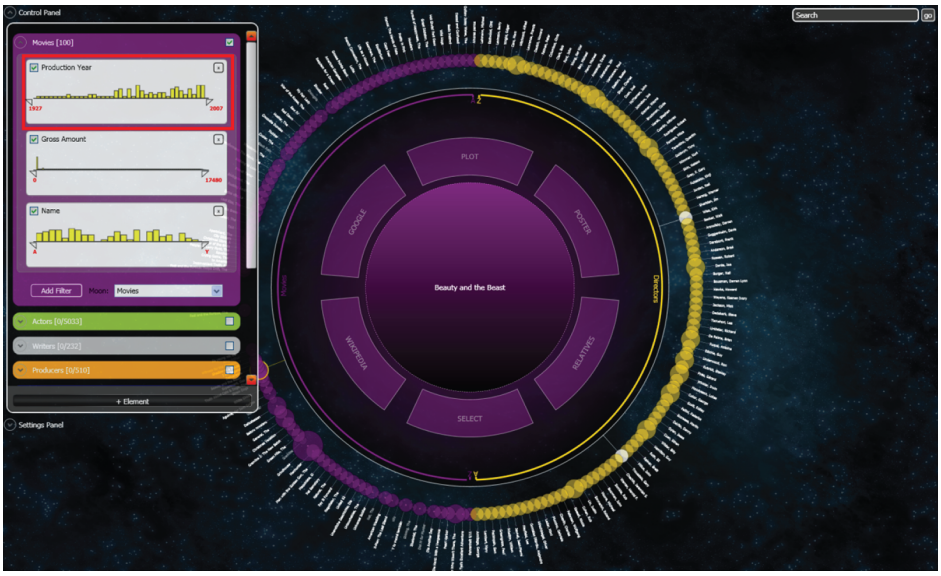

Figure 32: *A ZOIL visualisation for social relationships. Attributes can be browsed by zooming and panning the information landscape (Source: [HF08]).*

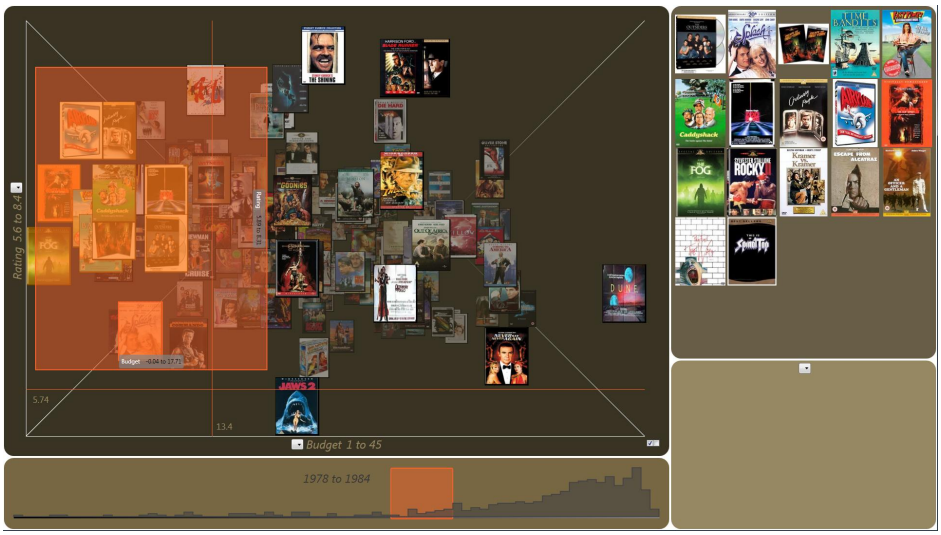

Figure 33: *"Timewarp" is a ZOIL visualisation that organises movie data in a 3D scatterplot, with time being on the Z-axis. The selection panel in the upper right corner embeds a nested information landscape, thus movie meta-data can be browsed by zooming (Source: [ES08]).*

and procedural programming. Since it builds upon the same foundation as WPF, it consequently uses many of its features.

Not only would it greatly simplify distribution of ZOIL case-studies and demonstrators, a web-based solution would allow for easier cross-platform capability, such as running ZOIL on a Mac or even mobile devices. This capability has already been tested in recent prototypes that are trying to add Silverlight support without compromising the existing ZOIL architecture. However, Silverlight has been extremely stripped down in functionality in order to reduce its size to a level suitable for a browser component. A lot of the current ZOIL components use features that are only available in WPF, so the framework is likely to be adjusted as not to sacrifice these. Newer revisions might therefore use a separate project for the core and input devices, with core functionality rewritten as to be able to compile both, WPF and Silverlight. Two additional projects could house the rest, specific to the platform. Consequently, each ZOIL component, control and visualisation will have to be written twice, for WPF and for Silverlight. This supports leveraging features of WPF, such as hardware acceleration and more advanced in-built controls, while maintaining compatibility with Silverlight.

5.3 Semantic Zooming Reworked

Even though semantic zooming was made a lot easier to use by allowing it to be defined entirely in declarative code (see chapter 2.4.2), handling sizing and positioning has proven to be a sometimes counter-intuitive process that takes a bit of trial & error to get the layout right. Newer case-studies try to automate that process entirely, stretching its content to fit the outer coordinate system. This behaviour can be controlled, for example to uniformly scale the component without distorting the aspect ratio. Visualisations can still apply outer dimensions of arbitrary aspect ratios without disturbing the information item's correct display.

5.4 Back-End for Complex Information Spaces

What was initially suspected as one of the ZOIL framework's biggest shortcomings, namely the lack of any back-end whatsoever, has had few limitations as the iTV prototype demonstrates. The more recent project called "The Timewarp", however, featured a data source of tens of thousands movies (as compared to 26 in the iTV prototype) that were not only zoomed, but filtered and synchronised over various visualisations [ES08]. This project is thus used as a testing ground for a new back-end implementation designed to simplify data-related tasks.

There were two main sources of inspiration regarding its concept, Yahoo! Pipes[33]

[33] http://pipes.yahoo.com/

and Django[34]. The first is a ”composition tool to aggregate, manipulate, and mash-up content from around the web”. Essentially, Pipes provides various pluggable building blocks that can be assembled using a graphical interface. Each of these feature input and output ports, in turn, which can be connected with each other provided they are compatible. This allows to create complex data pipelines, for example aggregating news feed from various websites and filter them for a set of specific keywords.

The second inspiration, Django, is a high-level framework for web development written in Python. Even though Django is perfectly capable of handling complex SQL queries, it abstracts it behind models, fetching data whenever it becomes necessary (so-called ”lazy” evaluation) and caching results. A user defines classes with properties that are automatically translated into relational tables behind the scenes. Whenever object instances are used, Django passively handles the underlying data connection.

The new back-end concept uses a stripped down combination of both approaches. Pipelines can be defined using a set of ”data expressions”, which define one input and one output port. In early tests there were two implementations of a data expression: data source and data filter. Data sources aggregate input with information fetched from a specific source, such as a local or remote XML, an enumerable list of objects or an SQL database. Data filters, in turn, manipulate their input by - for example - adding or removing certain attributes, searching for terms, or sorting and grouping items. The resulting pipes can then be used as a source for a WPF data binding, automatically handling notifications whenever the bound collection changes. Each expression is lazy and provides its own cache, thus updates don’t necessarily force the entire chain to be re-evaluated.

5.5 Multiple Representations

As already discussed in chapter 2.5.2, due to rendering in retained mode, a suitable implementation for cameras and magic lenses becomes challenging. Generally, there are two possible solutions:

Consistent with the way portals and visualisations handle multiple rendering of the same information items, cameras and magic lenses could work with their own copy of the element tree. For large information landscapes, introducing a second camera using this solution would pose a substantial overhead in both processing power and memory, as every element has to be allocated and rendered twice. Especially when both views allow manipulation of the landscape, synchronising the respective trees would be a burden.

The second possibility is to check for each component whether it needs to be drawn in two or more representations and create a second instance of the same. The need for

[34] http://www.djangoproject.com/

multiple instances in the same tree has not gone unnoticed by the WPF community, and there is already an elaborate approach to create, what has come to be called ”conceptual children” [WPF08].

5.6 Interface Overlay

Although the purist approach to ZOIL would be a navigation that takes place entirely on the information landscape, using only objects as interface elements, ”helpers” such as the iTV dock have proven great support for otherwise difficult to manage tasks and therefore suggest that indeed, in the long run, more elements might be introduced that are overlaying the information landscape (although, as demonstration by the ZUI presentation tool ZuiPrezi [35], these might still be zoomable). While it is simple to implement these, interaction with items from the information landscape has to be done manually. Moreover, choosing and updating semantic representations of components is handled by the information landscape, so once an instance of a component is created outside of the landscape's scope semantic zoom is defunct (the dock in the iTV prototype has to manually update the object it contains to work around that problem). If this scenario is to become commonplace, there is an apparent need for a better integration and a communication layer.

5.7 Local Zooming

There are certain scenarios where one might not want the information landscape to zoom globally. A typical example would be a scatterplot: once zoomed onto an information item, the axis and dimension information is out of focus, making exploration and comparison more difficult. A zoom that is limited to a certain region within the portal would - while sacrificing screen space - alleviate the problem. Technically this is already feasible using nested information landscapes. There should, however, be a solution that is more developer friendly, requiring only a single parameter to be set or region to be defined. Implementation of this attribute would be analogue to the way directed zoom targets are flagged.

[35] http://zuiprezi.com/

6 Conclusion

This work has briefly discussed ZOIL, a novel interface paradigm designed to be a replacement for the still predominant desktop metaphor, and introduced the theoretical and practical foundation of a WPF framework devised to be a foundation for prototypes and environments, as well as to implement ZOIL as a whole.

Not only does the ZOIL framework reflect all fundamental concepts off its underlying conceptual framework, it also provides enough in-depth implementations, pre-built components and visualisations, input device implementations and ease-of-use to allow elaborate demonstrators such as the iTV prototype to be written with relatively little effort. Moreover, its architecture ensures that developers can create new content without having to understand every single detail, as logical blocks are devised with as few interconnections and dependencies as possible.

Although the ZOIL framework takes a relatively unique position in the landscape of existing ZUI frameworks, its rich library of media integration, rendering quality and performance and developer-friendly nature certainly make it stand out enough to justify the effort that went into creating it. Building upon modern programming environments such as WPF and Silverlight, it profits from a foundation that is still under active development and will likely become both platform independent, as well as available in a version that will allow it to build browser applications for ZOIL as well.

Even though there are still limitations, none of them are impossible to solve and many work-arounds are already developed by third parties. And while there is certainly a lot more work ahead to reach its ultimate goal of becoming a desktop replacement, the ZOIL framework marks a first milestone on its way to technical realisation and another step towards the ”new computing”.

Acknowledgements

The closing words are thankfully dedicated to everyone contributing to the framework's development, as it has clearly grown beyond a one-man project over the recent months. Especially, I want to thank:

Hans-Christian Jetter, for coding support and proof-reading.

Sören Schubert, for his help with the iTV prototype and demonstration video.

Jane Mumford, for her splendid vector graphic themes, Gris and Verte.

A Framework Source

The framework's source, as well as its various demo applications and the iTV prototype are all maintained in the ZOIL repository of the HCI workgroup in the University of Konstanz:

`https://hci.uni-konstanz.de/svn/repos/zoil`

By the time of writing, access for both reading and writing is restricted and requires an account. Note that the framework is still undergoing heavy development and is far from a stage where it commits itself to backwards compatibility. Therefore, various demos might not compile properly due to changes in the API. If the interest lies not in the framework itself but in its applications, it is recommended to check out tagged revisions that are used to save important milestones and are thus best suited for demonstration purposes.

B Framework Documentation

Apart from inline documentation and demos, the HCI workgroup of the University of Konstanz maintains a Wiki where the framework's main documentation can be found:

`http://hci.uni-konstanz.de/permaedia/wiki/index.php/Hauptseite`

The article's name is "ZOIL framework"; viewing does not require an account, although the Wiki itself is password protected.

C ZOIL Framework Demonstration Video

The linked video shows various applications using the ZOIL framework.

`http://hci.uni-konstanz.de/permaedia/Videos/permaedia.avi`

The first section shows the iTV prototype setup as used during the conference (see chapter 4.2 for technical details). The second scene features the same application on a multi-touch table. In the third part the iTV demo is shown on two 67" HD displays, adding up to a total resolution of 3840 to 1080 pixels. Lastly, an information seeking system is shown that uses the ZOIL framework for zooming and panning.

The demo is also available as a (lower quality) video stream on the Permaedia project website:

`http://hci.uni-konstanz.de/index.php?a=research&b=projects&c=15957171`

D Comparison Demo Source

The source used to compare control, vector and image performance is made available as demos in the project repository (see appendix A) with the names ”ZOILPerformanceDemo” and ”PiccoloPerformanceDemo”.

References

[BB99] BEDERSON, Benjamin B. ; BOLTMAN, Angela: Does Animation Help Users Build Mental Maps of Spatial Information? In: *INFOVIS '99: Proceedings of the 1999 IEEE Symposium on Information Visualization*. Washington, DC, USA : IEEE Computer Society, 1999, S. 28

[Ben05] BENNETT, Michael: *A Framework for the Rapid Prototyping of Zoomable User Interfaces*, Department of Computer Science, University College Dublin, Diplomarbeit, 2005

[BGM04] BEDERSON, Benjamin B. ; GROSJEAN, J. ; MEYER, J.: Toolkit Design for Interactive Structured Graphics. In: *IEEE Transactions on Software Engineering* 30 (2004), Nr. 8, 535-546. `http://www.cs.umd.edu/hcil/jazz/learn/publications.shtml`

[BMG00] BEDERSON, Benjamin B. ; MEYER, Jon ; GOOD, Lance: Jazz: an extensible zoomable user interface graphics toolkit in Java. In: *UIST '00: Proceedings of the 13th annual ACM symposium on User interface software and technology*. New York, NY, USA : ACM Press, 2000, 171-180

[BSP+94] BIER, Eric A. ; STONE, Maureen C. ; PIER, Ken ; FISHKIN, Ken ; BAUDEL, Thomas ; CONWAY, Matt ; BUXTON, William ; DEROSE, Tony: Toolglass and magic lenses: the see-through interface. In: *CHI '94: Conference companion on Human factors in computing systems*. New York, NY, USA : ACM, 1994. – ISBN 0–89791–651–4, S. 445–446

[Don78] DONELSON, William C.: Spatial management of information. In: *SIGGRAPH '78: Proceedings of the 5th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM Press, 1978, 203-209

[Eng08a] ENGL, Andreas: *Bachelor Project: A Framework for an Infnitely Zoomable Information Landscape*. 2008

[Eng08b] ENGL, Andreas: *Term paper: From the Office Container to the Personal Information Cloud*. 2008

[ES08] ENGL, Andreas ; SCHUBERT, Sören: *Term paper: The Timewarp*. 2008

[FB95] FURNAS, George W. ; BEDERSON, Benjamin B.: Space-scale diagrams: understanding multiscale interfaces. In: *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA : ACM Press/Addison-Wesley Publishing Co., 1995. – ISBN 0–201–84705–1, S. 234–241

[FZ98] FURNAS, George W. ; ZHANG, Xiaolong: MuSE: a multiscale editor. In: *UIST '98: Proceedings of the 11th annual ACM symposium on User interface software and technology*. New York, NY, USA : ACM Press, 1998, 107-116

[Ger06] GERKEN, Jens: *Orientierung und Navigation in zoombaren Benutzerschnittstellen unter besonderer Berücksichtigung kognitations-psychologischer Erkenntnisse*. 2006

[Goo08] GOOGLE: *Piccolo2D: Structured 2D Graphics Framework*. `http://code.google.com/p/piccolo2d/`, 2008

[Ham08] HAMANN, Oliver: *Eagle Mode*. `http://eaglemode.sourceforge.net/`, 2008

[HCI08] HCIL, University of M.: *Piccolo Manual*, 2008

[HF08] HUBER, Stephan ; FRANTZEN, Benjamin: *Term paper: Visualisierung sozialer Zusammenhänge in der IMDb*. 2008

[HHN85] HUTCHINS, Edwin L. ; HOLLAN, James D. ; NORMAN, Donald A.: Direct Manipulation Interfaces. In: *Human-Computer Interaction* 1 (1985), Nr. 4, 311–338. `http://dx.doi.org/10.1207/s15327051hci0104_2`

[iTV07] EURO ITV.2008: Sixth European Interactive TV Conference, Salzburg, Austria, July 4th and 5th, 2008. In: *Comput. Entertain.* 5 (2007), Nr. 4, 1–5. `http://doi.acm.org/10.1145/1324198.1324211`. – ISSN 1544–3574

[JESR08] JETTER, Hans-Christian ; ENGL, Andreas ; SCHUBERT, Sören ; REITERER, Harald: Zooming not Zapping: Demonstrating the ZOIL User Interface Paradigm for ITV Applications. In: *Adjunct Proceedings of European Interactive TV Conference, Salzburg, Austria, July 3-4, 2008*, 2008. – Demonstration Session

[Jet07] JETTER, Hans-Christian: *Informationsarchitektur und Informationsvisualisierung für die Post-WIMP Ära*, Fachbereich für Informatik und Informationswissenschaft, Universität Konstanz, Diplomarbeit, 2007

[JKGR08] JETTER, Hans-Christian ; KÖNIG, Werner A. ; GERKEN, Jens ; REITERER, Harald: ZOIL - A Cross-Platform User Interface Paradigm for Personal Information Management. In: *Personal Information Management 2008: The disappearing desktop (a CHI 2008 Workshop), April 5-6, 2008, Florence, Italy*, 2008

[jmo07] JMORRIL, raygun: *WPF Win32 Interop Render Control*. `http://www.codeplex.com/WPFWin32Renderer/Release/ProjectReleases.aspx?ReleaseId=6198`, 2007

[JT07] JONES, W. ; TEEVAN, J.: *Personal Information Management*. Seattle, WA : University of Washington Press, 2007

[Jul02] JUL, Susanne: Predictive targeted movement in electronic spaces. In: *CHI '02: CHI '02 extended abstracts on Human factors in computing systems*. New York, NY, USA : ACM, 2002. – ISBN 1–58113–454–1, S. 626–627

[Jul03] JUL, Susanne: "This is a lot easier!": constrained movement speeds navigation. In: *CHI '03: CHI '03 extended abstracts on Human factors in computing systems*. New York, NY, USA : ACM, 2003. – ISBN 1–58113–637–4, S. 776–777

[KC07] KAPTELININ, V. ; CZERWINSKI, M.: *Beyond the Desktop Metaphor: Designing Integrated Digital Work Environments*. Cambridge, Mass. : MIT Press, 2007

[Kön06] KÖNIG, Werner A.: *Referenzmodell und Machbarkeitsstudie für ein neues Zoomable User Interface Paradigma*, Fachbereich für Informatik und Informationswissenschaft, Universität Konstanz, Diplomarbeit, 2006

[Lan88] LANSDALE, M.: The Psychology of Personal Information Management. In: *Applied Ergonomics* 19 (1988), March, Nr. 1, S. 55–66

[Med04] MEDIA, Mike B.: *ORRIL: A Simple Building Blocks Approach to Zoomable User Interfaces*. `citeseer.ist.psu.edu/725614.html`, 2004

[Mic08a] MICROSOFT: *Input Overview*. `http://msdn.microsoft.com/en-us/library/ms754010.aspx`, 2008

[Mic08b] MICROSOFT: *Silverlight Overview*. `http://msdn.microsoft.com/en-us/library/bb404708(VS.95).aspx`, 2008

[Mic08c] MICROSOFT: *Trees in WPF*. `http://msdn.microsoft.com/en-us/library/ms753391.aspx`, 2008

[Mic08d] MICROSOFT: *Windows Presentation Foundation Graphics Rendering Overview*. `http://msdn.microsoft.com/en-us/library/ms748373.aspx`, 2008

[Mic08e] MICROSOFT: *Windows Presentation Foundation Graphics Rendering Tiers*. `http://msdn.microsoft.com/en-us/library/ms742196.aspx`, 2008

[ND86] NORMAN, Donald A. ; DRAPER, Stephen W.: *User Centered System Design; New Perspectives on Human-Computer Interaction*. Hillsdale, NJ, USA : L. Erlbaum Associates Inc., 1986. – ISBN 0898597811

[Nie08] NIELSEN, J.: *Top-10 Application-Design Mistakes*. http://www.useit.com/alertbox/application-mistakes.html, 2008

[Nor04] NORMAN, D. A.: *Emotional Design*. 387 Park Avenue South, NY : Basic Books, 2004

[PCS95] PLAISANT, C. ; CARR, D. ; SHNEIDERMAN, B.: Image-Browser Taxonomy and Guidelines for Designers. In: *IEEE Software* 12 (1995), Nr. 2, 21-32. http://portal.acm.org/citation.cfm?id=624606.625443&dl=GUIDE&dl=GUIDE

[PF93] PERLIN, Ken ; FOX, David: Pad: an alternative approach to the computer interface. In: *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM Press, 1993, 57-64

[Pie05] PIETRIGA, Emmanuel: A Toolkit for Addressing HCI Issues in Visual Language Environments. (2005), 145-152. http://doi.ieeecomputersociety.org/10.1109/VLHCC.2005.11

[Ras00] RASKIN, Jef: *The Humane Interface: New Directions for Designing Interactive Systems*. Reading, Mass. : Addison-Wesley, 2000

[Shn87] SHNEIDERMAN, B.: Direct manipulation: A step beyond programming languages. (1987), S. 461–467. ISBN 0–934613–24–9

[Shn97] SHNEIDERMAN, Ben: Direct manipulation for comprehensible, predictable and controllable user interfaces. In: *IUI '97: Proceedings of the 2nd international conference on Intelligent user interfaces*. New York, NY, USA : ACM, 1997. – ISBN 0–89791–839–8, S. 33–39

[Shn98] SHNEIDERMAN, B.: *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Menlo Park, CA : Addison Wesley, 1998

[WPF08] WPF, Dr.: *Conceptual Children: A powerful new concept in WPF*. http://www.codeproject.com/KB/WPF/ConceptualChildren.aspx, 2008

List of Figures