Die approbierte Originalversion dieser Diplom-/Masterarbeit ist an der Hauptbibliothek der Technischen Universität Wien aufgestellt (http://www.ub.tuwien.ac.at).

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology (http://www.ub.tuwien.ac.at/englweb/).

**MASTERARBEIT**

# Evaluierung von Fisheye View-Listen als Auswahl- und Drag & Drop-Ziel

zur Erlangung des akademischen Grades
Diplomingenieur
(Dipl.-Ing.)

Ausgeführt am
Institut für Rechnergestützte Automation
Forschungsgruppe Industrial Software

der Technischen Universität Wien

unter der Anleitung von
Univ.-Prof. Dipl.-Ing. Dr. techn. Thomas Grechenig
und
Dipl.-Ing. Martin Tomitsch

durch
Christoph Wimmer, Bakk. techn.
Anzengrubergasse 4 / 16
A-1050 Wien, Austria

Wien, am 8. Oktober 2007

# Eidesstattliche Erklärung

Ich erkläre an Eides statt, daß ich die vorliegende Arbeit selbständig und ohne fremde Hilfe verfaßt, andere als die angegebenen Quellen nicht benützt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Wien, am 8. Oktober 2007

# Abstract

This diploma thesis explores the applicability and usability of fisheye view lists for visualizing large amounts of information within limited screen space. Fisheye views provide a balance between the global overview of large data structures and local detail within these structures. Therefore fisheye views allow the visualization of large amounts of data on a single screen without scrollbars, providing the user with the necessary details to interact efficiently with the data on screen without sacrificing the representation of the global context the user is operating in.

The goal of this diploma thesis was to evaluate the usability of fisheye view lists for use in desktop user interfaces. Usability tests were conducted to evaluate the usability and user acceptance of fisheye view lists in comparison to lists with scrollbars and treeviews.

This document presents origin and basic concepts of fisheye views and continues with an overview of related work and prior research in this area. Consequently the implementation of a fisheye view list as part of this thesis and the method employed in the usability tests is described. Finally the usability test results are presented and discussed.

# Zusammenfassung

Diese Diplomarbeit untersucht die Anwendbarkeit und Usability von Fisheye View Listen zur Visualisierung großer Datenmengen bei beschränktem Platz am Bildschirm. Fisheye Views bieten eine Balance zwischen globalem Überblick über große Datenstrukturen und lokalen Details innerhalb dieser Strukturen. Dementsprechend ermöglichen Fisheye Views die Visualisierung großer Datenmengen auf einem Bildschirm ohne Scrollbalken und bieten dem Anwender die notwendigen Details, um mit den Daten am Bildschirm effizient interagieren zu können ohne die Darstellung des globalen Kontexts zu vernachlässigen, in dem der Anwender operiert.

Das Ziel dieser Diplomarbeit war die Evaluierung der Usability von Fisheye View Listen zur Verwendung in Desktop User Interfaces. Es wurden Usability Tests durchgeführt, um die Usability und Anwenderakzeptanz von Fisheye View Listen im Vergleich zu Listen mit Scrollbalken und Baumdarstellungen zu evaluieren.

Diese Arbeit präsentiert Ursprung und grundlegende Konzepte von Fisheye Views und fährt mit einem Überblick über Related Work und verwandte Forschung auf diesem Gebiet fort. Weiters wird die Implementierung von Fisheye View Listen im Rahmen dieser Arbeit und die verwendete Methode für die Usability Tests beschrieben. Zum Abschluss werden die Ergebnisse der Usability Tests präsentiert und diskutiert.

# Inhaltsverzeichnis

# Kapitel 1

# Einleitung

Ein zentrales Problem des User Interface Designs ist die Auswahl von Daten und ihrer Darstellung auf dem beschränkten Platz, den Bildschirme bereitstellen. Trotz wachsender Bildschirmgrößen und -auflösungen erlauben uns Monitore nur den Blick auf einen kleinen Ausschnitt unserer Daten. Mit der zunehmenden Menge an Informationen, mit denen Computeranwender in den letzten Jahren konfrontiert waren, ist die Bedeutung von effizienter Informationsvisualisierung in User Interfaces weiter gewachsen.

Im Rahmen dieser Diplomarbeit wurde die Anwendbarkeit und Usability von Fisheye-Visualisierungsmethoden in Desktop User Interfaces untersucht. Fisheye-Visualisierungsmethoden bieten sich an, um große Datenmengen bei beschränktem Platz auf dem Bildschirm darzustellen. Sie bieten eine Balance zwischen globalem Überblick über eine große Datenstruktur und lokalen Details innerhalb dieser Datenstruktur. Dies wird erreicht, indem lokale Bereiche, die von hohem Interesse für den Betrachter sind, in allen Details dargestellt werden, während andere Bereiche von geringerem Interesse weniger detailreich dargestellt werden. Die detaillierte Darstellung von Bereichen von großem Interesse ist notwendig, um die Interaktion mit diesen Bereichen zu vereinfachen bzw. überhaupt zu ermöglichen. Gleichzeitig erlaubt die vereinfachte und abstrahierte Darstellung von Bereichen von geringerem Interesse

die Darstellung von großen Datenstrukturen auf kleinem Raum und gewährt dem Benutzer dadurch einen Überblick über den globalen Kontext. Fisheye Views bieten dem Benutzer die notwendigen Details um effizient mit den Daten am Bildschirm interagieren zu können ohne die Darstellung des globalen Kontexts zu vernachlässigen in welchem der Benutzer operiert.

Im Rahmen dieser Arbeit wurden Fisheye View Listen entwickelt. Dabei handelt es sich um Listen-Widgets, die sich einer visuellen Fisheye Verzerrung bedienen, um die Listenelemente kompakt am Bildschirm darzustellen. Zur Evaluierung der Anwendbarkeit und Benutzbarkeit von Fisheye Views in typischen Desktopapplikationen wurden Usability Tests durchgeführt. Auf Basis der entwickelten Fisheye View Listen wurde ein Testprogramm nach dem Vorbild verbreiteter Desktopapplikationen entwickelt, um Usability Tests zur Evaluierung der Benutzbarkeit und Benutzerakzeptanz von Fisheye View Listen durchzuführen.

## 1.1 Problemstellung

Diese Diplomarbeit untersucht die Anwendbarkeit und Benutzbarkeit von Fisheye-Visualisierungsmethoden für Desktopanwendungen. Der Fokus der Untersuchungen liegt dabei auf typischen Information Management und Information Retrieval Aufgaben, wie z.B. die Auswahl von einzelnen Elementen in einer langen Liste oder das Ablegen und Sortieren von Objekten.

Bei vielen Anwendungen ist das Ordnen und Sortieren von Daten eine "Begleiterscheinung", ein nebenläufiger Task, dessen Notwendigkeit sich aus dem Umgang mit dem Programm ergibt. Ein Beispiel hierfür wäre z.B. ein E-Mail-Client, dessen Hauptaufgabe darin besteht, dem Benutzer das Lesen und Schreiben von E-Mails zu ermöglichen. Das Sortieren von E-Mails kann als begleitender Task betrachtet werden, dessen Notwendigkeit sich aus der Menge von E-Mails ergibt, mit der Anwender konfrontiert sind. Diese Begleit-Tasks müssen möglichst nahtlos in den Workflow des Benutzers inte-

griert sein und müssen oft mit wenig Platz am Bildschirm auskommen, um die eigentliche Hauptaufgabe des Programms nicht negativ zu beeinträchtigen. Dementsprechend ist es notwendig, möglichst effiziente und platzsparende Möglichkeiten zur Visualisierung von umfangreichen Datenstrukturen zu finden und die Navigation in und Interaktion mit diesen Datenstrukturen zu verbessern.

Fisheye Views stellen eine mögliche Lösung für diese Problemstellung dar: Sie bieten eine Balance zwischen lokalen Details und globalem Kontext innerhalb einer Datenstruktur. Dies wird erreicht, indem lokale Bereiche von großem Interesse für den Benutzer detailreich dargestellt werden, während Details in anderen Bereichen ausgeblendet oder wegabstrahiert werden. Lokale Details von Bereichen von großem Interesse sind für den Benutzer wichtig, damit dieser Informationen richtig wahrnehmen und erkennen kann und um die Interaktion mit Objekten in diesen Bereichen sinnvoll zu ermöglichen. Der globale Kontext ist notwendig, um dem Benutzer zu vermitteln, in welchem Teil einer Datenstruktur er arbeitet und aus welchen anderen Teilen diese Struktur noch besteht.

Ein vorherrschendes Information Management Paradigma in heutigen Desktop User Interfaces basiert auf Hierarchien von Verzeichnissen, die weitere Unterverzeichnisse und Dateien enthalten. Verbreitete Visualisierungs- und Navigationsmethoden für diese Art von Daten sind z.B. Baumdarstellungen und Listen.

Traditionellen Methoden zur Visualisierung von und Navigation in Datenstrukturen, wie z.B. Baumdarstellungen oder Listen, gelingt zwar die Darstellung von lokalen Details, jedoch mangelt es oft einer adäquaten Darstellung des globalen Kontexts. Betrachtet man z.B. ein Filesystem mit einer breiten und tiefen Hierarchie von Verzeichnissen und Unterverzeichnissen, so wird eine Baumdarstellung dieser Struktur um ein Verzeichnis von momentanem Interesse zentriert sein. Nur dieses Verzeichnis und Verzeichnisse in unmittelbarer Nachbarschaft werden in dem beschränkten Platz auf dem Bildschirm ausreichend detailliert dargestellt, um mit ihnen arbeiten zu

können, während der Rest der Datenstruktur hinter Scrollbalken verschwindet. Baumdarstellungen bieten ausreichend lokale Details, um mit einigen wenigen Verzeichnissen effizient arbeiten zu können, es fehlt ihnen jedoch die Möglichkeit, große Datenstrukturen in ihrer Gesamtheit darzustellen und dem Benutzer den globalen Kontext zu vermitteln, in dem er operiert. Es besteht die Gefahr, dass die Benutzer die Orientierung verlieren.

Fisheye Views sind eine vielversprechende Visualisierungs- und Navigationstechnik zur Darstellung von großen Datenmengen auf einem Bildschirm, ohne Scrollbalken oder verschachtelte Hierarchien zu verwenden. Sie bieten dem Benutzer die notwendigen Details, um mit den Daten effizient interagieren zu können, ohne die Darstellung des globalen Kontexts zu vernachlässigen, in dem der Benutzer arbeitet.

## 1.2 Inhaltsübersicht

Kapitel 2 beschreibt den Ursprung und das grundsätzliche Prinzip hinter Fisheye Views in User Interfaces. Verwandte Bereiche aus dem Feld der Informationsvisualisierung werden kurz vorgestellt (Kapitel 2.1 und Kapitel 2.2) und die Grundidee hinter Fisheye Views wird präsentiert (Kapitel 2.3). Die Degree of Interest Funktion für Fisheye Views wird formalisiert (Kapitel 2.4) und die Darstellungs- und Anwendungsmöglichkeiten von Fisheye Views werden diskutiert (Kapitel 2.5).

Kapitel 3 setzt sich mit Fisheye Views in graphischen User Interfaces auseinander. Probleme etablierter User Interface Elemente werden diskutiert (Kapitel 3.1) und Vorteile von Fisheye Views für graphische User Interfaces werden erläutert (Kapitel 3.2). Weiters werden bekannte Usabilityprobleme von Fisheye Views in graphischen User Interfaces und mögliche Lösungsansätze präsentiert (Kapitel 3.3).

Kapitel 4 stellt Related Work vor: Die grundlegende Arbeit von Furnas wird vorgestellt (Kapitel 4.1). Fisheye Menus, welche großen Einfluss auf

diese Arbeit hatten, werden umfangreich präsentiert (Kapitel 4.2). Weiters werden Arbeiten aus weiteren Anwendungsgebieten von Fisheye Views diskutiert (Kapitel 4.3).

Kapitel 5 widmet sich Design und Implementierung von Fisheye View Listen im Rahmen dieser Arbeit. Designüberlegungen (Kapitel 5.1), verwendeter Algorithmus zur Berechnung der Fisheye Transformation (Kapitel 5.2) und implementierungsspezifische Details (Kapitel 5.3) werden beschrieben.

Kapitel 6 beschreibt die angewandte Methode der im Rahmen dieser Arbeit durchgeführten Usability Tests im Detail und präsentiert das für die Tests entwickelte Testprogramm (Kapitel 6.2), Testziele (Kapitel 6.1), eine Beschreibung der Testpersonen (Kapitel 6.3), Testszenarien (Kapitel 6.4), Testmaterial (Kapitel 6.5), Testdesign (Kapitel 6.6) und die Ergebnisse des Pilottests (Kapitel 6.7).

Kapitel 7 präsentiert die Ergebnisse der durchgeführten Usability Tests hinsichtlich Taskerfüllungszeiten (Kapitel 7.1), Korrektheit (Kapitel 7.2), Anwenderpräferenzen (Kapitel 7.3) und Interaktionsverhalten der Testpersonen (Kapitel 7.4).

In Kapitel 8 werden die Ergebnisse der Usability Tests schlussendlich diskutiert.

# Kapitel 2

# Fisheye Visualisierungs- und Navigationstechniken

Ein Hauptproblem, mit dem sich die Informationsvisualisierung seit Jahrzehnten beschäftigt, ist der beschränkte Platz zur Darstellung von Informationen und die effiziente Nutzung des zur Verfügung stehenden Platzes. Im Laufe der Jahre wurden zahlreiche Ansätze für die Lösung dieses Problems erarbeitet und kontinuierlich weiterentwickelt.

## 2.1 Zoomable Interfaces

Ein Ansatz zur Lösung dieses Problems mit einer langen Geschichte in der Informationsvisualisierung sind zoombare Interfaces. Dieser Ansatz folgt dem Beispiel einer Zoomlinse: Der Anwender beginnt mit einer globalen Ansicht der gesamten Struktur und kann auf lokale Details innerhalb der Struktur heranzoomen. Weiters kann der Anwender zu jedem Zeitpunkt wieder herauszoomen um zu sehen, wo innerhalb der globalen Struktur er soeben arbeitet. Hornbæk et al. (2002) definiert zoombare Interfaces anhand folgender Charakteristika: In einem zoombaren Interface sind Informationsobjekte in Raum

und Maßstab organisiert und ein Benutzer kann mit dem Informationsraum direkt mittels Zoomen und Schwenken interagieren.

Bei zoombaren Interfaces kann grundsätzlich zwischen geometrischem Zoom und semantischem Zoom unterschieden werden: Bei geometrischem Zoom ändert sich die Darstellungsgröße der Informationsobjekte linear. Wenn man näher an ein Objekt heranzoomt, so wird es größer dargestellt. Bei semantischem Zoom ändert sich nicht nur die Darstellungsgröße der Elemente, sondern auch die Auswahl und Struktur der Elemente. Ein klassisches Beispiel hierfür sind Landkarten: Während auf einer Weltkarte beispielsweise nur sehr große Städte bzw. Städte von großer Bedeutung wie z.B. Hauptstädte eingezeichnet sind, sind auf der Karte eines Landes üblicherweise auch sehr viel kleinere Städte eingezeichnet. Diese kleineren Städte werden in der "Zoomstufe" der Weltkarte gewissermaßen herausgefiltert, um die Darstellung übersichtlicher zu gestalten. Stark vereinfacht könnte man sagen, dass sich bei geometrischem Zoom ändert *wie* etwas dargestellt wird, während sich bei semantischem Zoom ändert *was* dargestellt wird.

Ein Problem von zoombaren Interfaces ist, dass lokale Details und globaler Kontext niemals gleichzeitig sichtbar sind, was die Orientierung für den Anwender erschwert. Ein möglicher Lösungsansatz hierfür besteht darin, sowohl Detailansicht als auch herausgezoomte Gesamtansicht z.B. mittels Splitviews gleichzeitig am Bildschirm darzustellen. Diese Darstellungsmethode zählt zu den Overview+Detail Visualisierungsmethoden. Jedoch hat auch dieser Ansatz den Nachteil, dass der Zusammenhang zwischen Detailansicht und Übersicht nicht unmittelbar verständlich ist. Eine Studie von Hornbæk et al. (2002) hat eine solche Overview+Detail Visualisierungsmethode anhand einer Landkarten-Applikation untersucht (siehe Abbildung 2.1) und mit einer Landkartenapplikation ohne Overview-Funktion verglichen und kam dabei zu folgendem Ergebnis: 80% der Testpersonen gaben an, dass sie das Overview+Detail Interface bevorzugen. Dennoch waren die Testpersonen bei der Erfüllung der Tasks mit dem Interface ohne Overview schneller. Die Studie kommt zu dem Schluss, dass zoombare Interfaces ohne Overview gewisse Vorteile gegenüber zoombaren Interfaces mit Overview haben können und

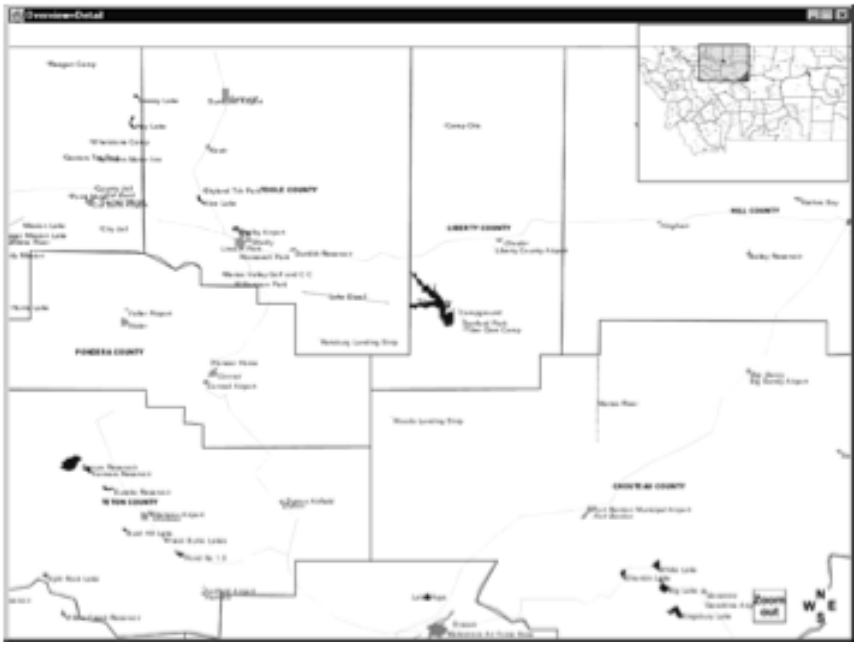

Abbildung 2.1: Zoombares Overview+Detail Interface zur Darstellung von Landkarten

dass die Erwartungen an die Nützlichkeit von Overviews in zoombaren Interfaces nicht gänzlich erfüllt wurden. Als ein möglicher Grund hierfür wird unter anderem vermutet, dass das Wechseln zwischen Overview und Detailansicht mit mentaler Anstrengung für die Testpersonen verbunden war. In Kaptelinin (1995) wurde unter anderem die Performance von Testpersonen beim Auffinden von Daten mittels einer zoombaren Map-Funktion mit und ohne Overview untersucht. Diese Studie kam zu einem ähnlichen Ergebnis wie Hornbæk et al. (2002): Bei der Task Completion Time konnte kein signifikanter Performanceunterschied zwischen dem Interface mit Overview und dem Interface ohne Overview festgestellt werden. Darüberhinaus gaben die Testpersonen in dieser Studie an, das Interface ohne Overview zu bevorzugen.

Beispiele für zoombare Interfaces sind unter anderem Sketchpad (Sutherland, 1964) (siehe Abbildung 2.2), Spatial Data Management System (Donelson, 1978) und Pad++ (Bederson & Hollan, 1994) (siehe Abbildung 2.3).

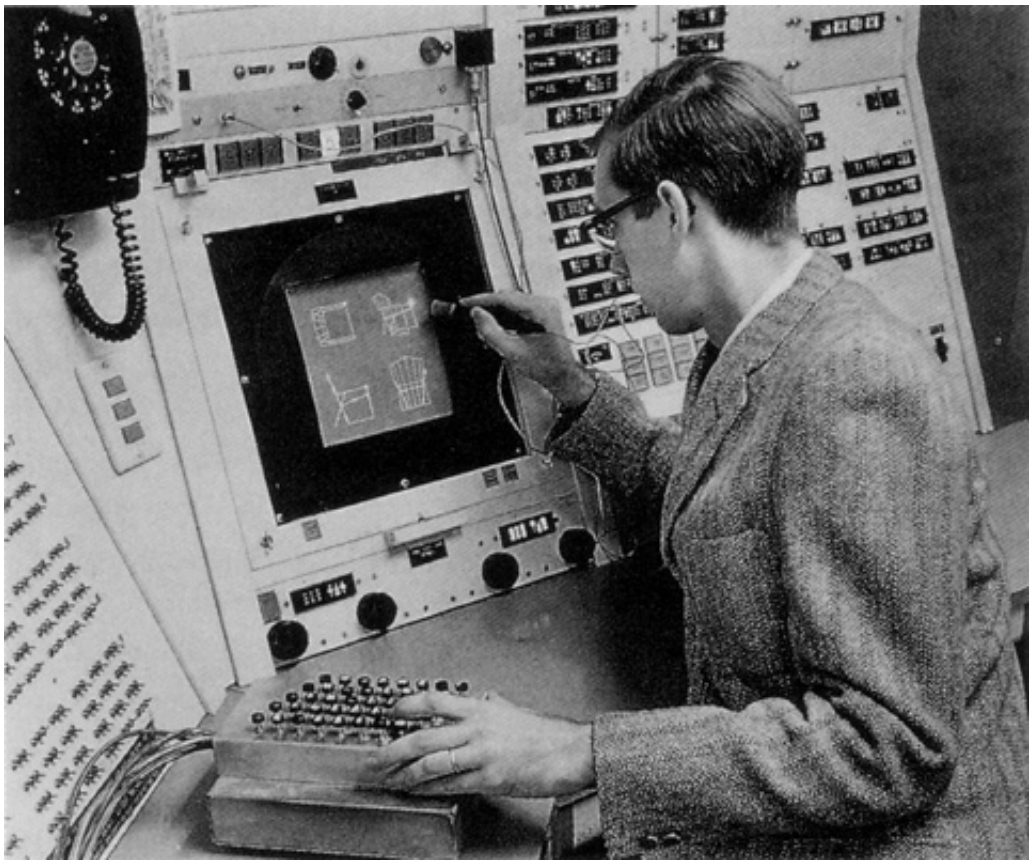

Abbildung 2.2: Sketchpad


Abbildung 2.3: Pad++ Filebrowser in drei unterschiedlichen Zoomstufen

## 2.2 Fokus+Kontext Visualisierungstechniken

Fisheye Visualisierungstechniken zielen darauf ab, eine Balance zwischen der Darstellung von lokalen Details und der Darstellung des globalen Kontexts in einer einzigen, dynamischen Ansicht zu finden. Sie zählen zur Kategorie der Fokus+Kontext Visualisierungstechniken. Fokus+Kontext Visualisierungstechniken liegt die Idee zu Grunde, dem Benutzer eine detaillierte Ansicht eines Teils der Datenstruktur zu bieten, während gleichzeitig auch die Datenstruktur in ihrer Gesamtheit dargestellt wird. Der Unterschied zu zoombaren Interfaces bzw. zu Overview+Detail Interfaces besteht darin, dass mehr als nur ein Maßstab gleichzeitig verwendet werden kann, dass diese unterschiedlichen Maßstäbe in einer einzigen dynamischen Ansicht kombiniert werden können und dass es dadurch zu einer Verzerrung der Darstellung kommt.

Die Grundidee hinter Fokus+Kontext Visualisierungstechniken geht von drei Prämissen aus (Card et al., 1999):

- Der Anwender benötigt sowohl einen globalen Überblick (Kontext) als auch detaillierte Informationen (Fokus) zur gleichen Zeit.
- Die Information, die in der globalen Übersicht benötigt wird, kann sich von der Information in der Detailansicht unterscheiden, d.h. globale Übersicht und Detailansicht können unterschiedlich detailliert sein.
- Diese beiden unterschiedlichen Arten von Information können in einer einzigen, dynamischen Ansicht kombiniert werden.

Daraus ergeben sich folgende Grundprinzipien für Fokus+Kontext Visualisierungstechniken:

- Bestimmte Bereiche werden äußerst detailliert dargestellt (Fokus).
- Ein Überblick über die gesamte Struktur mit geringerem Detailgrad muss erhalten bleiben (Kontext).

- Beide Darstellungen (Fokus+Kontext) müssen gleichzeitig sichtbar und in einer einzigen Ansicht kombinierbar sein.

Zu den typischen Fokus+Kontext Visualisierungstechniken gehören unter anderem Fisheye Views (Furnas, 1982, 1986), Polyfocal Displays (Kadmon & Shlomi, 1978), Bifocal Lens (Apperley et al., 1982), Document Lens (Robertson & Mackinlay, 1993), Perspective Wall (Mackinlay et al., 1991) und Hyperbolic Tree Browser (Lamping et al., 1995; Lamping & Rao, 1999) . Eine Übersicht über verschiedene Fokus+Kontext-Visualisierungstechniken bieten Leung & Apperley (1994) in *A Review and Taxonomy of Distortion-Oriented Presentation Techniques.*

## 2.3 Fisheye Views im Allgemeinen

Fisheye Views bieten eine Balance zwischen lokalen Details und globalem Kontext, indem sie sich der Analogie einer Fisheye Linse (wie aus der Photographie bekannt) bedienen. Eine solche Linse zeigt Orte im Zentrum im Detail, während entfernt gelegene Orte immer weniger und weniger detailliert dargestellt werden. Diese Analogie lässt sich insofern auf das Problem der Informationsvisualisierung anwenden, als Anwender typischerweise besonderes Interesse an einem bestimmten Teil einer Datenstruktur und nicht der gesamten Datenstruktur haben, z.B. weil sie auf der Suche nach bestimmten Informationen sind. Bei der Verwendung von Fisheye-Visualisierungstechniken gilt es, diese besonders interessanten Bereiche in das "Zentrum" der Fisheye View zu rücken.

Ein schönes Beispiel für die Grundidee hinter Fisheye Views, welches auch Furnas (1986) anführt, ist das Cover des Magazins *The New Yorker* vom 29. März 1976, *A View of the World from 9th Avenue* von Saul Steinberg (Abbildung 2.4). Diese Karrikatur illustriert humorvoll das typische Weltbild eines New Yorkers: Im Vordergrund sieht man die 9th Avenue und 10th Avenue detailreich dargestellt. Man erkennt Menschen, Autos, Läden, Ampeln, etc.

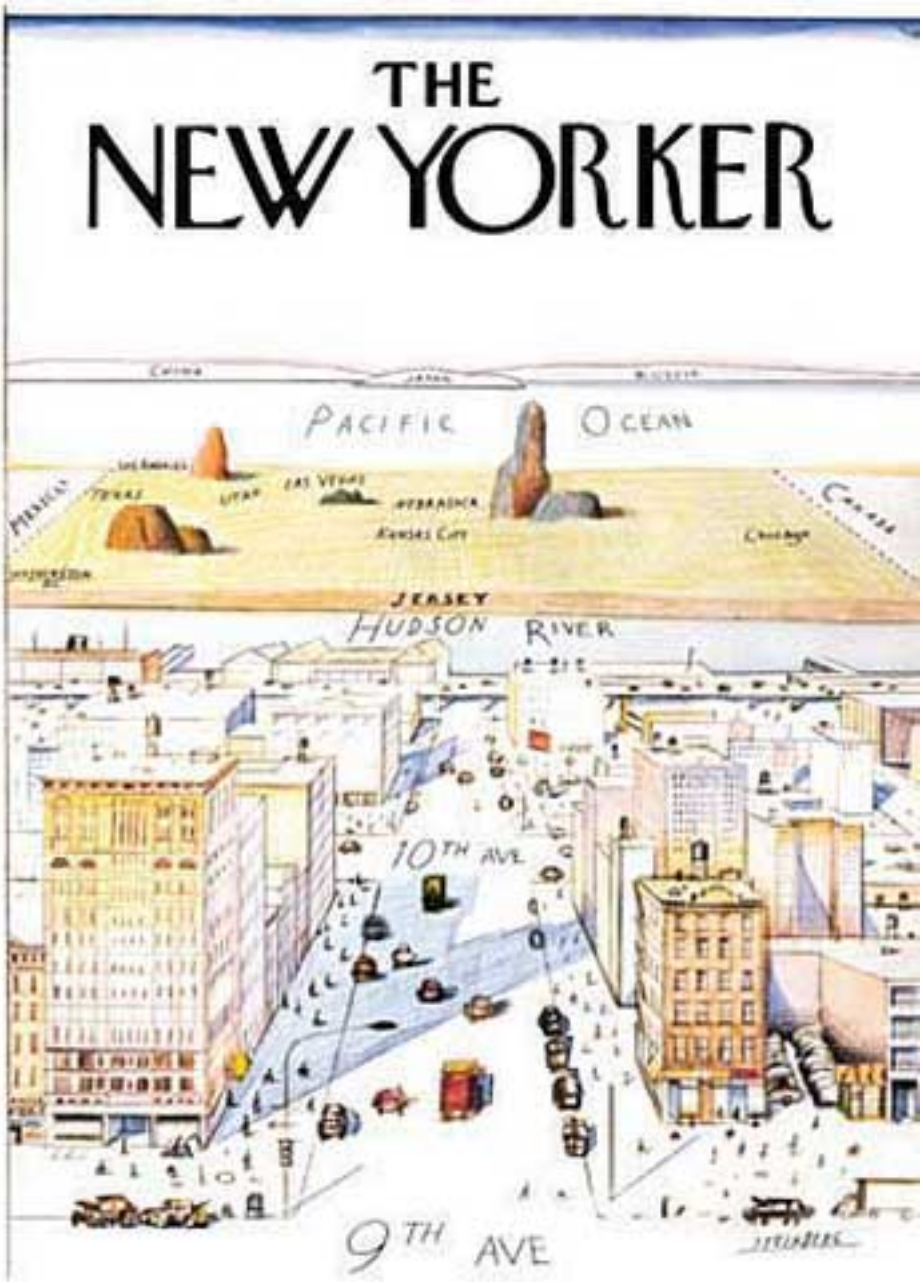


Abbildung 2.4: A View of the World from 9th Avenue, veröffentlicht in *The New Yorker*, 29. März 1976

Dahinter sieht man in der Ferne den Hudson River und New Jersey. Folgt man dem Blick weiter in Richtung Osten, so erkennt man diverse Städte und Bundesstaaten, sowie Mexiko linker- und Kanada rechterhand, bevor man beim Pazifischen Ozean angelangt. Jenseits des Pazifiks sieht man nur noch China, Japan und Russland als grobe Flecken am Horizont.

Die 9th Avenue und 10th Avenue entsprechen dem Fokusbereich in dieser Darstellung, man erkennt unterschiedliche Details und die Darstellung ist ausreichend detailliert, um sinnvoll mit seiner Umgebung interagieren zu können. Sucht man z.B. einen Briefkasten um einen Brief aufzugeben oder sucht nach seinem Auto, das irgendwo am Straßenrand geparkt ist, so ist dies bei dem dargestellten Detailgrad im "Fokusbereich" möglich. Je weiter man nach Osten blickt, desto gröber wird die "Auflösung" bzw. der "Level of Detail" der Darstellung. Zahlreiche Details und Informationen werden weggelassen oder abstrahiert. Dieser Bereich entspricht dem Kontext, er veranschaulicht das Wissen über die Welt im Ganzen. Die Darstellung ist zwar nur grob und nicht besonders detailgetreu, aber dennoch ist dieser Bereich wichtig, um New York als Stadt in einem größeren Kontext zu begreifen: New York als Teil der Vereinigten Staaten, umringt von seinen Nachbarstaaten und weit entfernt dem Rest der Welt. Obwohl die Darstellung nicht detailliert genug ist, um ein Postamt in Chicago oder Tokyo ausfindig zu machen, wird niemand bestreiten, dass ein grundlegendes Wissen über die Welt im Ganzen essentiell für jeden New Yorker Bürger ist.

*A View of the World from 9th Avenue* veranschaulicht aber nicht nur die Grundidee hinter Fisheye Views, es zeigt auch sehr schön, weshalb Fisheye Views zur Visualisierung von Informationen geeignet sind. Wie jeder guten Karrikatur liegt auch dieser gewiss ein Fünkchen Wahrheit zu Grunde. Auch wenn das Bild zumindest leise Kritik am beschränkten Weltbild vieler Menschen übt, läßt sich nicht von der Hand weisen, dass dieses Weltbild bis zu einem gewissen Grad exemplarisch für die Wahrnehmung des Menschen ist. Das Ausblenden und Wegabstrahieren von Details, die für den Menschen nicht von unmittelbarer Bedeutung sind, ist notwendig, um nicht von all den Informationen, die uns umgeben, förmlich erschlagen zu werden. Diesem

Grundsatz folgen auch Fokus+Kontext Visualisierungsmethoden.

Furnas (1986) hat in *Generalized Fisheye Views* natürlich auftretende Fisheye Views aus kognitionswissenschaftlicher Hinsicht untersucht. Diese Untersuchungen haben gezeigt, dass Menschen ihre Umwelt oftmals ähnlich einer Fisheye View betrachten. Im Rahmen eines Experiments wurden die Teilnehmer angewiesen, sich vorzustellen, dass das Kind eines neu hinzugezogenen Nachbarn sie bittet, ihm etwas über X zu erzählen (wobei X Staaten, Präsidenten, geschichtliche Ereignisse sein konnte). Die Aufgabe der Teilnehmer war es, 10 Beispiele der Kategorie X aufzuführen, von denen sie meinten, das Kind sollte darüber Bescheid wissen. Die Ergebnisse spiegelten ein Fisheye Subset wider: die von den Teilnehmern angegebenen Beispiele wiesen entweder eine hohe a-priori Bedeutung auf oder waren für die Teilnehmer in geographischer oder zeitlicher Hinsicht naheliegend. Ein weiteres Beispiel für natürlich auftretende Fisheye Views ist, dass bei Angestellten eine Fisheyeansicht der Managementstruktur ihrer Firma zu finden war: Die Angestellten wussten einerseits über direkte Vorgesetzte und Mitarbeiter innerhalb der Abteilung Bescheid, also Personen, die ihnen nahe stehen, andererseits waren in anderen Abteilungen nur leitende Angestellte in führenden Positionen bekannt, also Personen mit hoher a-priori Bedeutung innerhalb der Firma. Eine ähnliche Beobachtung ließ sich bei der redaktionellen Themenwahl in Zeitungen anstellen: Zeitungen enthalten typischerweise lokale Nachrichten aus der Umgebung des Erscheinungsortes, während Beiträge und Artikel zu Themen aus geographisch entfernten Regionen üblicherweise eine weitaus größere Bedeutung und Tragweite aufweisen.

Wie bereits erwähnt zielen Fisheye Views darauf ab, eine Balance zwischen der Darstellung von lokalen Details und der Darstellung des globalen Kontexts zu finden. Lokale Details von Bereichen, die für den Anwender von momentanem Interesse sind, sind notwendig, damit der Anwender Informationen in diesen Bereichen richtig wahrnehmen kann und um die Interaktion mit Objekten in diesen Bereichen sinnvoll zu ermöglichen. Der globale Kontext ist notwendig, damit der Anwender einen Überblick darüber bekommt, welche anderen Bereiche einer großen bzw. komplexen Struktur noch existie-

ren und wo diese zu finden sind. Der globale Kontext kann darüber hinaus auch noch notwendig sein, um die Bedeutung von lokalen Informationen zu vermitteln bzw. überhaupt erst begreiflich zu machen.

## 2.4 Formalisierung einer Degree of Interest-Funktion für Fisheye Views

Um das Fisheye Konzept auf Interface Design-Probleme anwenden zu können ist es notwendig, das Konzept zu formalisieren, wie Furnas (1986, 1982) das in *Generalized Fisheye Views* und *The Fisheye View: a new look at structured files* gemacht hat. Ausgangspunkt hierfür ist die Definition einer "Degree of Interest" (DOI)-Funktion. Diese geht von der Annahme aus, dass Anwender zu einem bestimmten Zeitpunkt nicht an allen Teilen einer Datenstruktur gleichermaßen interessiert sind, sondern größeres Interesse an bestimmten Teilen der Datenstruktur haben als an anderen. Die Gestaltung eines Interfaces sollte darauf abzielen, dem Anwender genau jene Teile anzuzeigen, an denen er am meisten interessiert ist. Das Ziel der DOI-Funktion ist die Formalisierung dieses Interesses an bestimmten Teilen einer Datenstruktur.

Furnas zerlegt die DOI-Funktion in eine a-priori und eine a-posteriori Komponente. Bei der a-priori Komponente handelt es sich um jenen Teil, der die inhärente Wichtigkeit eines Teils der Datenstruktur betrifft und weitestgehend von der gegebenen Interaktion des Anwenders mit der Datenstruktur unabhängig ist. Besonders große, wichtige, informationsreiche oder auch allgemeine Teile einer Datenstruktur haben eine höhere a-priori Wichtigkeit als andere Teile. Die a-posteriori Komponente ist jener Teil, der sehr stark und direkt von der gegebenen Interaktion des Anwenders mit der Datenstruktur abhängt. In Fisheye Views wie von Furnas vorgestellt handelt es sich um ein simples Modell, das davon ausgeht, dass sämtliche Interaktionen zu jedem Zeitpunkt auf genau einen Punkt fokussiert sind. Die interessensbedingte Komponente der Funktion ist demnach einfach als Distanz von diesem

Fokuspunkt modelliert.

Wie Furnas (1982) schreibt sind demnach folgende Eigenschaften für die Erzeugung einer Fisheye View notwendig:

- Ein Fokuspunkt '.'
- Die Distanz vom Fokuspunkt: D(.,x), wobei D(.,.)=0
- Level of Detail, Wichtigkeit, Auflösung: LOD(x)

Zunächst wird ein Fokuspunkt benötigt, der sich aus der Interaktion mit der Datenstruktur ergibt (z.B. aus der Position des Mauszeigers). Weiters benötigt man die Möglichkeit, eine Distanz zwischen diesem Fokuspunkt und jedem anderen Punkt innerhalb der Datenstruktur zu bestimmen. Diese Distanz kann z.B. die direkte, lineare Distanz zwischen einem Punkt und dem Fokuspunkt sein, oder auch eine strukturell definierte Distanz. Zu guter Letzt benötigt man eine Vorstellung von der Bedeutung und Gewichtung der Elemente der Datenstruktur (LOD).

Auf Basis dieser Komponenten lässt sich folgende allgemeine DOI-Funktion für Fisheye Views für einen Punkt x definieren:

$$DOI(x|.) = F(LOD(x), D(., x)) \tag{2.1}$$

wobei größere Zahlen ein größeres Interesse bedeuten. Das Interesse wächst dabei in Abhängigkeit vom ersten Argument (Level of Detail) und schrumpft in Abhängigkeit vom zweiten Argument (Distanz vom Fokuspunkt), d.h. das Interesse wächst mit zunehmender Wichtigkeit des Punktes und schwindet mit wachsender Distanz vom Fokuspunkt. In ihrer simpelsten, additiven Form ergibt sich daraus folgende Fisheye DOI-Funktion:

$$DOI(x|.) = LOD(x) - D(., x) \tag{2.2}$$

Dies geht von der Annahme aus, dass globales Interesse an einem Punkt mit schwindendem Detailgrad und der Distanz vom Fokuspunkt abnimmt.

Dementsprechend folgt daraus die Annahme, dass das Interesse an besonders bedeutenden Punkten und an nahe am Fokus gelegenen Punkten am größten ist.

## 2.5 Darstellungsmöglichkeiten und Anwendbarkeit von Fisheye Views

Grundsätzlich lässt sich die Anwendbarkeit von Fisheye Views in zwei Felder unterteilen: Fisheye Views können einerseits verwendet werden, um zu bestimmen, *wie* etwas dargestellt wird. Anwendungen in diesem Bereich kommen meist aus dem Informationsvisualisierungsumfeld und bedienen sich häufig einer visuellen Verzerrung zur Darstellung der Fisheye View. Andererseits können Fisheye Views verwendet werden, um zu bestimmen, *was* dargestellt wird. Anstatt nur zu fragen, wie der Kontext dargestellt wird, rückt die Frage in den Vordergrund, wieviel Kontext denn überhaupt in Betracht gezogen werden soll. Furnas (1986, 1982, 2006) konzentriert sich in seinen Arbeiten zum Thema Fisheye Views vor allem auf diesen Bereich.

Bei der Verwendung eines visuellen Verzerrungseffekts kann die DOI-Funktion z.B. dazu verwendet werden, um die Größe eines Elements in der Darstellung auf dem Bildschirm zu bestimmen: Elemente mit hohem Degree of Interest werden größer dargestellt, während Elemente mit geringem Degree of Interest kleiner dargestellt werden. Charakteristisch für visuell verzerrte Fisheye Views ist der fließende Übergang zwischen der starken Vergrößerung im Fokusbereich und der Verkleinerung im Kontextbereich. Beispiele hierfür sind z.B. der Vergrößerungseffekt im Mac OS X Dock, die Fisheye Menus von Bederson (2000) oder Graphical Fisheye Views of Graphs von Sarkar & Brown (1992).

Bei der Verwendung eines semantischen Filters zur Umsetzung einer Fisheye View werden nicht nur die Parameter der visuellen Repräsentation modifiziert, stattdessen wird die Auswahl und Struktur der dargestellten Daten

verändert. Bei einer Fokusänderung wird ein beobachtetes Objekt nicht einfach nur verkleinert, sondern die gesamte Erscheinung des Objekts kann sich verändern. Die Erscheinung ist dabei abhängig von der Bedeutung, die dem Objekt eingeräumt wird. Ein simpler Ansatz für die Umsetzung eines semantischen Filters in einer Fisheye View ist folgender: Man definiere einen Schwellenwert k. Ein Element x wird nur dann dargestellt, wenn sein Degree of Interest größer diesem Schwellenwert k ist. Ein Beispiel für einen solchen semantischen Filter findet sich in der Darstellung von Baumstrukturen mittels Fisheye Views in *Generalized Fisheye Views* (Furnas, 1986) oder in *Using fisheye for navigation on small displays* (Backvall et al., 2000).

Visuelle Verzerrungstechniken haben besonders bei fließender Interaktion mit einem System gewisse ästhetische Vorteile. Bei der Verwendung eines semantischen Filters in einem System, mit welchem ein Benutzer mit einer Maus interagiert, würde die Darstellung sich scheinbar sprunghaft ändern, während die Verwendung einer visuellen Verzerrung fließende Übergänge ermöglicht. Die Verwendung eines semantischen Filters ohne irgendeiner Form der visuellen Verzerrung bietet sich deshalb eher für Systeme an, in denen die Interaktion in diskreten Schritten erfolgt.

Furnas (2006) diskutiert in *A Fisheye Follow-up: Further Reflections on Focus + Context* die Trennung von Inhalt und Darstellung. Darin beschäftigt er sich mit dem "What" und "How", was dargestellt wird und wie es dargestellt wird, in Hinblick auf diverse Fokus+Kontext Visualisierungstechniken der letzten 20 Jahre. Unter anderem wird darin auch der Unterschied zwischen visueller Verzerrung und semantischem Filtern anhand des Beispiels in Abbildung 2.5 veranschaulicht.

Furnas (2006) argumentiert, dass auch visuelle Verzerrungstechniken eine inhärente Filterfunktion erfüllen: Auch wenn Informationen nicht direkt gefiltert, sondern nur in ihrer Darstellungsgröße skaliert werden, ergibt sich eine Filterfunktion auf Grund der nicht uneingeschränkten Auflösung der Übertragungsmedien (sei dies ein Bildschirm oder auch die Retina des menschlichen Auges). Sobald Elemente in ihrer Darstellung so klein skaliert werden,

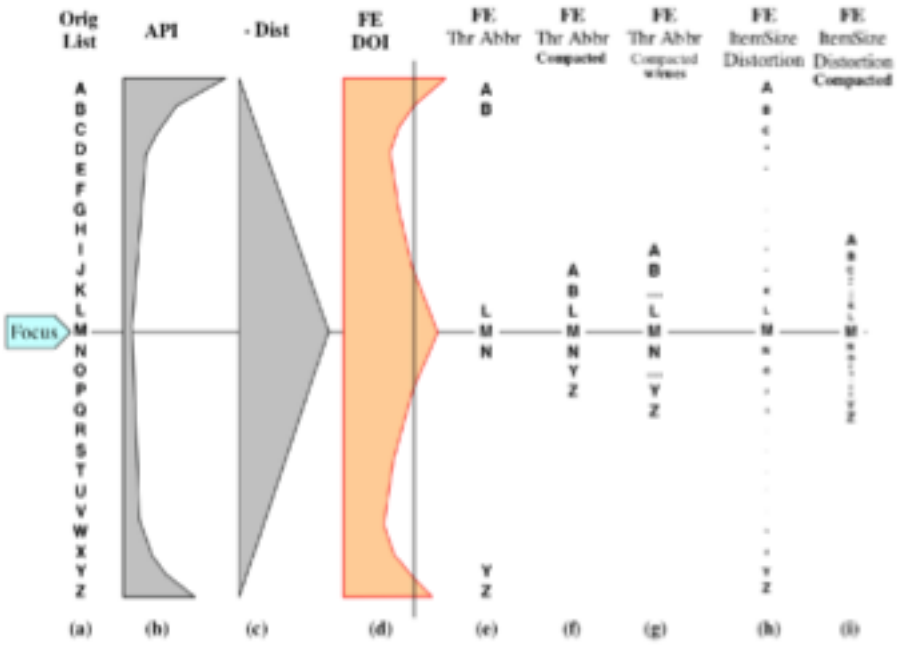


Abbildung 2.5: Filterung und Verzerrung in einer Fisheye View einer Liste. Figur (a) zeigt eine Liste des Alphabets. In Figur (b) wird eine a-priori Bedeutungsfunktion für diese Liste definiert. In diesem Fall wird Elementen am Anfang und am Ende der Liste eine größere Bedeutung bzw. Wichtigkeit als den Elementen in der Mitte der Liste beigemessen. Figur (c) veranschaulicht die Distanzfunktion vom Fokus, der auf Element "M" liegt. Figur (d) zeigt die DOI-Funktion über der Liste, basierend auf der additiven Kombination der a-priori Bedeutungsfunktion und der Distanzfunktion. Weiters ist über der DOI-Funktion ein Schwellenwert definiert, der in einem Fisheye DOI Subset resultiert. Figur (e) zeigt genau jenes Subset, indem Listenelemente, deren Degree of Interest unter dem Schwellenwert liegt, gefiltert werden. In Figur (f) wird die Darstellung dieses Subsets komprimiert um Platz zu sparen, jedoch geht dabei auch die Information über die Distanz zwischen den Listenelementen verloren. In Figur (g) wird diese verlorene Distanzinformation zumindest teilweise wiederhergestellt, indem das Symbol "..." an jenen Stellen eingefügt wird, wo Listenelemente ausgelassen werden. Figur (h) veranschaulicht die Anwendung einer visuellen Verzerrung auf die Liste; die Darstellungsgröße der Listenelemente variiert in direkter Abhängigkeit von ihrem Degree of Interest. In Figur (i) wurden die Listenelemente nach der Anwendung einer visuellen Verzerrung zusammengerückt, um eine einheitliche Dichte zu erreichen.

dass sie vom Benutzer nicht mehr wahrnehmbar bzw. erkennbar sind, werden diese Elemente gewissermaßen gefiltert.

In weiterer Folge kommt (Furnas, 2006) zu dem Schluss, dass der Wert der Fisheye Metapher nicht allein in der Analogie der visuellen Verzerrung einer Fisheye Linse zu finden ist, sondern in den Fisheye DOI Subsets, deren Anwendbarkeit sich über die Grenzen der Informationsvisualisierung hinaus erstreckt. Ein Beispiel hierfür ist die Anwendbarkeit auf Recommendersysteme, die nicht nur interessensmäßig naheliegende Produkte vorschlagen sollten, sondern auch Produkte von allgemein hoher Bedeutung. Ein Beispiel für die allgemeine Anwendbarkeit von Fisheye Views außerhalb der Informationsvisualisierung ist CoMem (Demian & Fruchter, 2006), ein System welches das Auffinden und Verstehen von Informationen in großen hierarchischen Datensammlungen unterstützen soll. In diesem Projekt wurden Fisheye DOI Subsets verwendet, um die Zusammengehörigkeit von Daten zu ermitteln.

# Kapitel 3

# Fisheye Views in Graphischen User Interfaces

Fisheye Visualisierungstechniken, wie überhaupt Informationsvisualisierungstechniken im Allgemeinen, zielen auf die Lösung eines althergebrachten Interfacedesignproblems ab: Während Anwender mit gewaltigen und noch immer wachsenden Daten- und Informationsmengen umgehen müssen, ist der Platz zur Darstellung dieser Datenmengen äußerst beschränkt, einerseits durch die Bildschirmgrößen, andererseits durch die Grenzen der menschlichen Wahrnehmung. Dieses Problem läuft auf die Entscheidung hinaus, welche Teile einer großen Datenstruktur dargestellt werden sollen und welche nicht (Furnas, 1982).

## 3.1 Probleme etablierter User Interface-Elemente

Typische User Interface-Elemente in heutigen Desktop-Umgebungen begnügen sich meist mit der Darstellung eines kleinen Ausschnitts einer großen Datenstruktur. Dementsprechend gelingt diesen GUI-Elementen zwar die Darstellung von lokalen Details, jedoch oft auf Kosten der Darstellung des globalen

Kontexts in dem der Anwender operiert. Beispiele hierfür sind z.B. klassische Listen und Baumdarstellungen, wie sie von der Mehrheit von graphischen User Interface Toolkits und Libraries unterstützt werden. Betrachtet man z.B. die Darstellung eines großen Filesystems mit tausenden Verzeichnissen verteilt auf mehrere Hierarchieebenen als Baum, wie sie heutzutage üblich ist, so ist diese Darstellung stets um ein Verzeichnis von momentanem Interesse für den Anwender zentriert und nur dieses Verzeichnis und Verzeichnisse in unmittelbarer Nachbarschaft werden detailliert dargestellt. Der Rest der Datenstruktur findet keinen Platz am Bildschirm und verschwindet hinter Scrollbalken. Zur Navigation innerhalb der Datenstruktur dienen meist Scrollbalken am rechten und unteren Rand des Fensters, mit denen der Benutzer den dargestellten Ausschnitt verändern kann. Insbesondere für die Navigation in zwei Dimensionen haben sich Scrollbalken als wenig effizient erwiesen (Kaptelinin, 1995), vermutlich weil der Scrollvorgang hierbei in eine horizontale und eine vertikale Komponente zerlegt ist.

Baumdarstellungen bieten ausreichend lokale Details, um mit einigen wenigen Verzeichnissen effizient arbeiten zu können, sie bieten jedoch keine Möglichkeit, große Datenstrukturen in ihrer Gesamtheit darzustellen und dem Anwender Informationen über den globalen Kontext, in dem er operiert, zu vermitteln. Die einzige Information darüber, wo sich der Benutzer innerhalb einer Datenstruktur befindet, ergibt sich meist aus der Position und Größe der Scrollbalken. Daraus ergibt sich die Gefahr, dass die Benutzer die Orientierung verlieren, und keinen Überblick darüber haben, in welchem Teil der Datenstruktur sie gerade arbeiten. Diese Gefahr besteht z.B. insbesondere dann, wenn Benutzer ihre Arbeit unterbrechen und später wieder zurückkehren. Darüberhinaus gestaltet sich die Interaktion mit Verzeichnissen, die momentan nicht angezeigt werden, erheblich schwieriger und zeitaufwändiger als die Interaktion mit Verzeichnissen, die gerade am Bildschirm angezeigt werden. Anwender müssen zuerst zu einem Verzeichnis navigieren, damit dieses am Bildschirm dargestellt wird, bevor sie damit interagieren können. Durch die mangelnde Vermittlung des globalen Kontexts wird das Auffinden von Ordnern noch weiter erschwert.

Analog verhält es sich mit Listen, deren Darstellung ebenfalls um ein Element zentriert ist. Wie auch bei den beschriebenen Baumdarstellungen ist auch bei Listen die Darstellung auf die Nachbarelemente eines momentanen Fokuselements beschränkt. Auch hier dienen üblicherweise Scrollbalken zur Navigation innerhalb der Liste und diese stellen meist den einzigen Indikator für die Position innerhalb der gesamten Datenstruktur dar. Ein anderes Beispiel, welches auch Furnas (1982) präsentiert, ist die Darstellung von langen Texten, deren Zeilen linear aufeinanderfolgend in einem Fenster mit einem Mechanismus zum Scrollen dargestellt werden.

Aus diesem Ansatz, der auf die Darstellung des globalen Kontexts wenig Rücksicht nimmt, ergeben sich gewisse Probleme: Einerseits ergibt sich die Bedeutung von lokalen Informationen oft erst aus dem globalen Kontext, andererseits wird die Orientierung und Navigation innerhalb der Datenstruktur durch mangelnde Informationen über den Kontext, in dem man arbeitet, erschwert. Schlimmstenfalls besteht das Risiko, dass Anwender die Orientierung verlieren und nicht mehr wissen, wo genau innerhalb einer Datenstruktur sie sich befinden. Dieses Risiko ist z.B. nach Pausen oder Unterbrechungen der Arbeit besonders gegeben.

Der Fokus dieser Arbeit liegt auf der Untersuchung der Anwendbarkeit von Fisheye Views für typische Desktopapplikationen, in denen Information Management-Aufgaben oftmals nur eine nebenläufige Rolle einnehmen. Beispiele hierfür wären z.B. E-Mail-Clients, Mediaplayer oder Webbrowser: Diesen Programmen ist gemein, dass die Verwaltung von Daten nicht die eigentliche Hauptaufgabe des Programms darstellt. E-Mail-Clients sollen in erster Linie das Lesen, Schreiben, Empfangen und Versenden von E-Mails ermöglichen. Die Notwendigkeit E-Mails zu verwalten und zu kategorisieren ergibt sich lediglich aus der zunehmenden Masse an E-Mails, mit der Anwender heutzutage konfrontiert sind. Betrachtet man einen typischen E-Mail-Client wie z.B. Apple Mail (Abbildung 3.1), so stellt man fest, dass nur ein kleiner Teil des Programmfensters zur Darstellung der Ordnerhierarchie verwendet wird. Der Großteil des Platzes im Programmfenster wird für andere Programmfunktionen beansprucht. Mediaplayer dienen vorrangig der

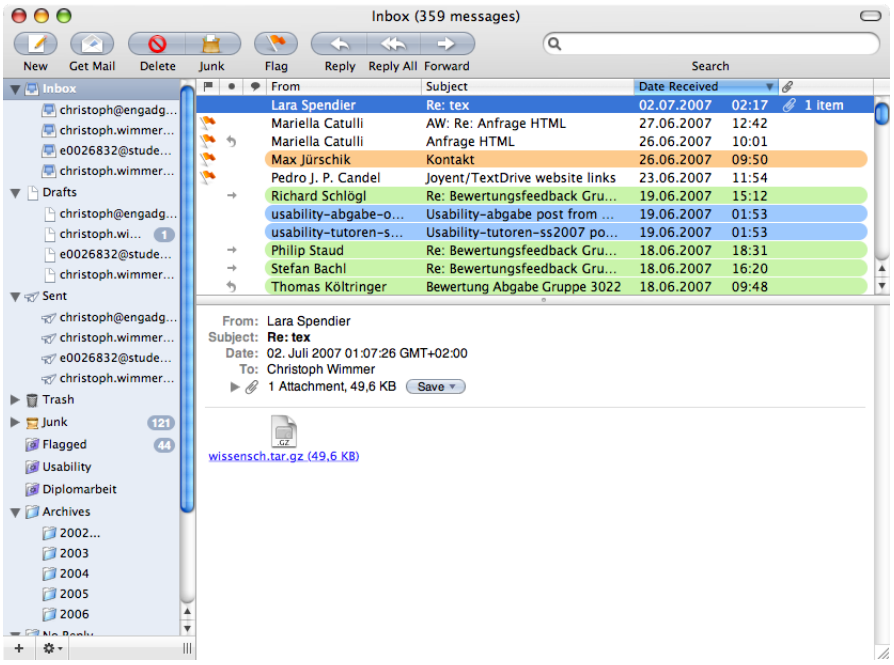


Abbildung 3.1: Apple Mail

Wiedergabe von Musik- und Videodateien, jedoch bieten viele dieser Programme heutzutage die zusätzliche Funktionalität, eine Medienbibliothek zu verwalten. Webbrowser ermöglichen es in erster Linie, im World Wide Web zu surfen, doch eine Vielzahl an Bookmarks macht eine Form der rudimentären Verwaltung dieser Bookmarks notwendig. Auch hier ist der Platz für die Verwaltung von Bookmarks meist stark beschränkt, wie am Beispiel von Mozilla Firefox (Abbildung 3.2) zu sehen ist. Bei allen diesen Anwendungen steht die Verwaltung von Daten im Hintergrund, es handelt sich gewissermaßen um eine "Begleiterscheinung" bzw. einen nebenläufigen Task, dessen Notwendigkeit sich aus dem Umgang mit dem Programm ergibt. Dementsprechend sollte dieser nebenläufige Task möglichst nahtlos und ohne zu stören in den Workflow des Anwenders integriert sein.

Der zur Verfügung stehende Platz am Bildschirm für diese begleitenden Verwaltungs- und Organisationstasks ist meist stark beschränkt. Daraus ergeben sich Einschränkungen hinsichtlich der Anwendbarkeit gewisser

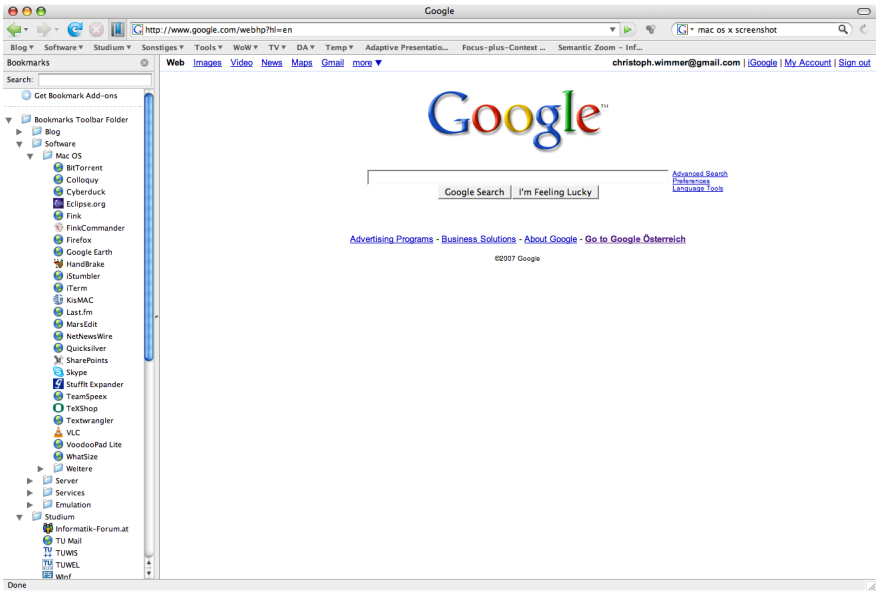


Abbildung 3.2: Mozilla Firefox

Visualisierungsmethoden. Betrachtet man z.B. den für die Darstellung von Ordnern verfügbaren Platz in verbreiteten Desktopapplikationen, wie z.B. E-Mail-Clients, so ist offensichtlich, dass beispielsweise ein Hyperbolic Tree Browser (Lamping & Rao, 1999) zur Visualisierung der Ordnerstruktur aufgrund der gegebenen Platzeinschränkungen nicht praktikabel implementierbar wäre. Eine Visualisierungstechnik, die sich stattdessen anbietet, ist eine Fisheye View, bzw. konkret die Verwendung einer eindimensionalen Fisheye-Verzerrung, wie sie von Bederson auf Menüs angewendet wurde (Bederson, 2000). Diese Methode ist gut geeignet für die Darstellung großer Datenstrukturen ohne Scrollbalken auf beschränktem Raum. Sie bietet einen Überblick über die gesamte Datenstruktur und zeigt lokale Details durch die Anwendung einer visuellen Fisheye-Verzerrung in einer Dimension. Die Darstellungsgröße von Listenelementen ist abhängig von ihrem Degree of Interest.

## 3.2 Vorteile von Fisheye Views für graphische User Interfaces

Fisheye Views versprechen eine Lösung der in Abschnitt 3.1 beschriebenen Probleme, indem sie sowohl lokale Details als auch globalen Kontext in einer einzigen dynamischen Ansicht vereinen. Den in Abschnitt 3.1 erwähnten, etablierten Methoden der Informationsdarstellung (Bäume, Listen, Textbereiche, etc.) liegt eine implizite, wenn auch etwas naive Degree of Interest-Funktion zu Grunde: Sämtliche Elemente, die sich innerhalb der halben Fensterhöhe um das zentrale Element befinden, sind ausreichend interessant um dargestellt zu werden, und alle anderen Elemente sind es nicht. Fisheye Views versuchen den tatsächlichen Interessen des Anwenders mit Hilfe einer ausgereifteren Degree of Interest-Funktion (wie in Abschnitt 2.4 beschrieben) zu entsprechen: Das Interesse an einem Element aus einer Datenstruktur ergibt sich aus der a-priori Wichtigkeit des Elements und der Distanz zwischen Element und einem Fokuspunkt, der sich aus der Interaktion mit der Datenstruktur ergibt. Durch die nahtlose Integration von Fokus und Kontext in einer einzigen dynamischen Ansicht wird darüber hinaus das Problem überwunden, dass nicht sofort ersichtlich ist, in welchem Zusammenhang Fokus und Kontext stehen.

Zur Umsetzung von Fisheye Views in graphischen User Interfaces bieten sich grundsätzlich sowohl visuelle Verzerrungstechniken als auch semantische Filterung an (siehe hierzu Abschnitt 2.5). Bei visueller Verzerrung ergibt sich die Darstellungsgröße von Elementen am Bildschirm aus dem Degree of Interest, während sich bei semantischer Filterung die Auswahl der Elemente und Struktur der Darstellung auf Basis der DOI-Funktion ändert. Wie bereits in Kapitel 2.5 dargelegt bieten sich visuelle Verzerrungstechniken für Systeme mit fließender Interaktion besonders an, um sprunghafte Änderungen in der Darstellung zu vermeiden.

Durch die Kombination von lokalen Details und globalem Kontext in einer einzigen, dynamischen Ansicht helfen Fisheye Views bei der Orientierung

in großen Datenstrukturen. Sie geben dem Anwender einen guten Überblick, wo innerhalb der Datenstruktur er sich befindet und welche anderen Teile der Datenstruktur noch existieren. Gleichzeitig erlaubt die detaillierte Darstellung des Fokusbereichs die effiziente Interaktion mit der Datenstruktur. Darüber hinaus kann die Darstellung des Kontexts von besonderer Bedeutung sein, wenn sich die eigentliche Bedeutung von Informationen erst aus dem Kontext ergibt.

Ein weiterer Vorteil von Fisheye Views zur Darstellung großer Datenstrukturen ist die Unterstützung, die sie dem Benutzer bei der Navigation bieten (Furnas, 2006). In (Furnas, 1997) wird festgestellt, dass eine Grundanforderung für die effiziente Navigation in großen Datenmengen die Möglichkeit ist, mit einer möglichst kleinen Anzahl an Schritten von einem beliebigen Punkt zu einem anderen beliebigen Punkt in der Datenstruktur zu gelangen, wobei jeder dieser Schritte aus einer kleinen Menge von möglichen Schritten wählbar sein muss. Elemente in der Nähe des Fokuspunkts werden detailliert dargestellt und lassen sich dementsprechend einfach und direkt ansteuern ("short-distance links"). Gleichzeitig bietet die Darstellung des Kontexts eine Auswahl an "long-distance links", welche es vereinfachen, mühelos zu einem weit entfernten Punkt in der Datenstruktur zu springen. Fisheye Views bieten die Möglichkeit der Darstellung einer Datenstruktur in ihrer Gesamtheit ohne der Verwendung von Scrollbalken. Daraus ergibt sich, dass der Benutzer schnell und einfach zu einem beliebigen Punkt in der Datenstruktur gelangen kann. Darüber hinaus erleichtert die Darstellung der Datenstruktur in ihrer Gesamtheit die Orientierung und verschafft dem Benutzer einen besseren Überblick, aus welchen Teilen die Datenstruktur besteht und wo diese zu finden sind.

## 3.3 Bekannte Usability-Probleme von Fisheye Views und mögliche Lösungsansätze

Ein Problem bei der Anwendung von Fisheye Views mit visueller Verzerrung in graphischen User Interfaces stellt der stark eingeschränkte Motorspace für die Auswahl von Elementen dar. Bereits kleine Mausbewegungen resultieren in einer Änderung des Fokus. Wie in (Bederson, 2000) beschrieben, muss bei traditionellen Darstellungsmethoden die Maus über die gesamte Höhe eines Elements bewegt werden, um zum nächsten Element zu gelangen. Bei der Verwendung einer Fisheye View mit visueller Verzerrung ist dies anders, hier muss die Maus nur über die Höhe des Elements mit der kleinsten Höhe bewegt werden, um zum nächsten Element zu wechseln. Die Fisheye Verzerrung hat nur Auswirkungen auf die Größe der Darstellung eines Elements, die Größe des Auswahlbereichs ist konstant und entspricht der Größe des kleinsten Elements. Wie Bederson (2000) beschreibt ist dies ein fundamentales Resultat des Fisheye Algorithmus, weil alle Elemente durch Mausbewegung im fixen Raum des GUI-Elements auswählbar sein müssen. Ist ein Fisheye View GUI-Element, das sich einer eindimensionalen, vertikalen visuellen Verzerrung bedient, z.B. 800 Pixel hoch und enthält 100 Elemente, so ist der vertikale Raum zur Auswahl eines Elements nur ca. 8 Pixel hoch, selbst wenn das Element aufgrund der Fisheye Verzerrung deutlich größer am Bildschirm dargestellt wird. Daraus folgt, dass Elemente in einer Fisheye View mit Visueller Verzerrung schwieriger auszuwählen sind, auch wenn sie groß und gut lesbar dargestellt werden. Fitts Law (Fitts, 1954) besagt, dass die Zeit, um ein Element auszuwählen, invers proportional zur Größe des Zielbereichs für die Auswahl ist. Daraus folgt, dass man für die Auswahl eines Elements in einer Fisheye View mit einem 8 Pixel hohen Zielbereich länger braucht als für die Auswahl eines Elements mit einer fixen Höhe von 16 Pixeln als Zielbereich in z.B. einer traditionellen Liste. Dadurch kann es für den Anwender insbesondere bei steigender Anzahl an Elementen in der Fisheye View schwierig werden, schnell, präzise und zuverlässig ein bestimmtes Element auszuwählen.

Um diesem Problem entgegenzuwirken hat Bederson (2000) bei Fisheye Menus einen sogenannten Focus Lock Modus implementiert: Sobald der Benutzer den Mauscursor in die rechte Hälfte des Menüs bewegt, wird der Focus Lock Modus aktiviert. Der Fokus bleibt in diesem Modus konstant auf einem Element und wenn der Benutzer den Mauszeiger außerhalb des bestehenden Fokusbereichs bewegt, so wird der Fokusbereich in die entsprechende Richtung erweitert. Eine Pilotstudie hat jedoch gezeigt, dass diese Implementierung eines Focus Lock Modus nicht intuitiv für Anwender verständlich ist. Für eine genauere Beschreibung des Focus Lock Modus und Fisheye Menus im Allgemeinen siehe auch Kapitel 4.2.

Einen anderen möglichen Lösungsansatz für das Problem des stark eingeschränkten Motorspace bei visuell verzerrten Fisheye Views stellt Semantic Pointing dar (Blanch et al., 2004). Bei Semantic Pointing handelt es sich um eine Interaktionstechnik, bei der die Zielauswahl in GUIs durch die Entkoppelung von Darstellungsgröße und Motorgröße verbessert und erleichtert werden soll. Unter Motorgröße ist hierbei jene Größe gemeint, die dem Anwender zur Verfügung steht, um ein Ziel am Bildschirm auszuwählen. Bei herkömmlichen GUIs sind Darstellungsgröße und Motorgröße üblicherweise direkt voneinander abhängig: Je größer ein GUI Element am Bildschirm dargestellt wird, desto größer ist auch der Bereich zur Auswahl dieses Elements und umgekehrt. Durch die Entkoppelung von Darstellungsgröße und Motorgröße wird es möglich, dass auch nur klein am Bildschirm dargestellte Elemente leicht auszuwählen sind. Die Entkoppelung von Darstellungsgröße und Motorgröße wird durch die Adaption von Control-Display Ratio in Abhängigkeit der Cursorposition erreicht. Für den Anwender kann sich Semantic Pointing beispielsweise so äußern, dass sich die Geschwindigkeit von Mausbewegungen abhängig von der Cursorposition am Bildschirm scheinbar ändert. Die Maus würde sich beispielsweise in GUI Elementen mit großer Motorgröße und kleiner Darstellungsgröße scheinbar langsamer bewegen. Bei visuell verzerrten Fisheye Views kommt eine Control-Display Ratio Adaptation eigentlich bereits zur Anwendung: Obwohl Elemente im Fokusbereich größer dargestellt werden, bleibt die Motorgröße aller Elemente unabhängig

von der Darstellungsgröße konstant - daraus ergibt sich überhaupt erst das Problem des verkleinerten Motorspace. Offen bleibt die Frage, wie sich Semantic Pointing auf visuell verzerrte Fisheye Views anwenden lässt, um diesem Problem entgegenzuwirken. Ein möglicher Ansatz besteht darin, den Motorspace aller Elemente zu vergrößern. Dies hat jedoch auch zur Folge, dass sich die Maus innerhalb der Fisheye View generell langsamer zu bewegen scheint und birgt das Risiko, gewisse Vorteile hinsichtlich Navigationsgeschwindigkeit zu entfernten Teilen der Datenstruktur hinfällig zu machen. Eine andere Möglichkeit in Anlehnung an den Focus Lock Modus der Fisheye Menus bestünde darin, zwischen einem Navigationsmodus, in dem man sich schnell von einem Ort zum andern bewegen kann, und einem Zielauswahl-Modus, der eine präzisere Kontrolle bietet, zu unterscheiden. Im Navigationsmodus wäre die Motorgröße der Elemente dementsprechend kleiner als im Zielauswahlmodus. Selbst bei der Umsetzung solcher unterschiedlicher Modi bleibt jedoch die Frage offen, wie man zwischen diesen effizient und für den Anwender intuitiv durchschaubar wechseln kann.

Ein weiteres Problem im Zusammenhang mit der Zielauswahl und Fokusänderung in visuell verzerrten Fisheye Views besteht darin, dass sich die Elemente innerhalb der Fisheye View scheinbar bewegen, während sich der Fokuspunkt ihnen nähert (Gutwin, 2002). Fisheye Views bedienen sich eines nichtlinearen Vergrößerungseffekts. Fokusbereiche werden stark vergrößert, während Kontextbereiche verkleinert dargestellt werden. Diese nichtlineare Vergrößerung führt jedoch zu Verzerrungen der Repräsentation der Daten, die Usabilityprobleme verursachen können. Ändert man in einer viusell verzerrten Fisheye View den Fokuspunkt, so scheint es, als würde das Ziel sich in die entgegengesetzte Richtung der Bewegung der "Vergrößerungslinse" bewegen, d.h. ein Ziel bewegt sich auf einen sich nähernden Mauszeiger zu und von einem sich entfernenden Mauszeiger weg. Diese Bewegung erschwert die Zielauswahl und gezielte Fokusänderung in Fisheye Views.

Gutwin (2002) schlägt zur Kompensation dieses Problems "Speed Coupled Flattening" vor. Hierbei wird die Geschwindigkeit und Beschleunigung des Mauszeigers analysiert, um den Zielauswahlvorgang einer Interaktion zu

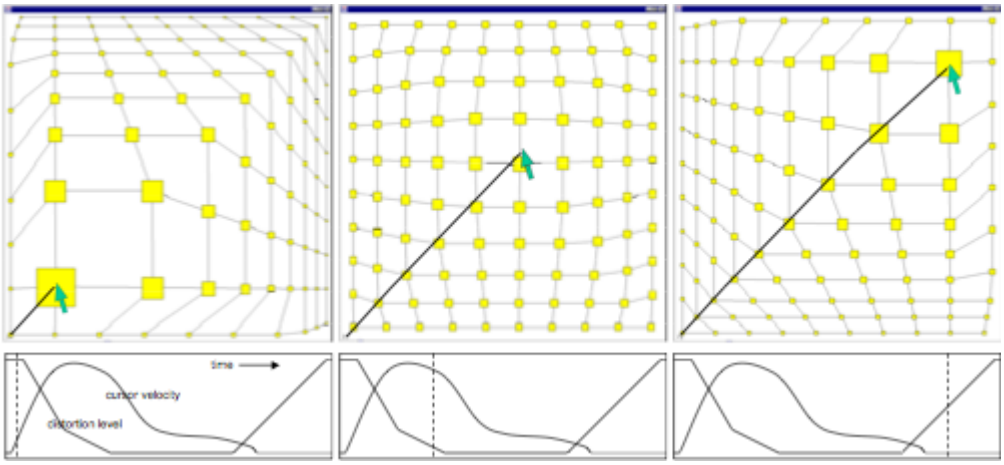


Abbildung 3.3: Speed Coupled Flattening. Die obere Reihe zeigt die Fisheye View und Zeigerpfad. Die untere Reihe zeigt Zeigergeschwindigkeit und Verzerrungsgrad. Die gepunktete Linie indiziert den mit dem Bild in der oberen Reihe korrespondierenden Zeitpunkt.

bestimmen. Wenn sich der Mauszeiger schnell bewegt oder beschleunigt, so ist anzunehmen, dass der Benutzer versucht zu einem neuen Fokuspunkt zu navigieren anstatt die lokalen Details genauer zu betrachten. Wird ein solcher Zielauswalvorgang festgestellt, dann wird die visuelle Verzerrung der Darstellung reduziert. Erst wenn die Bewegung langsamer wird, wird auch die visuelle Verzerrung langsam wieder zu ihrem ursprünglichen Ausmaß erhöht (siehe Abbildung 3.3). Im Rahmen eines Experiments auf Basis des Sarkar & Brown (1992) Fisheye zur Visualisierung von Graphen hat sich gezeigt, dass Speed Coupled Flattening die Zeit zur Auswahl eines Ziels und die Fehlerrate signifikant reduziert.

Darüber hinaus wurde festgestellt (Skopik & Gutwin, 2005), dass die visuelle Verzerrung in Fisheye Views es Anwendern erschwert, sich einzelne Elemente und deren Position innerhalb einer Datenstruktur zu merken. Durch visuelle Verzerrung ändern sich Position, Größe und lokale Muster von Elementen in einer Fisheye View in Abhängigkeit vom Fokuspunkt. Dies erschwert es Anwendern, sich an die Position von Elementen zu erinnern, weil es nicht länger möglich ist, sich die absolute Position von Elementen einzu-

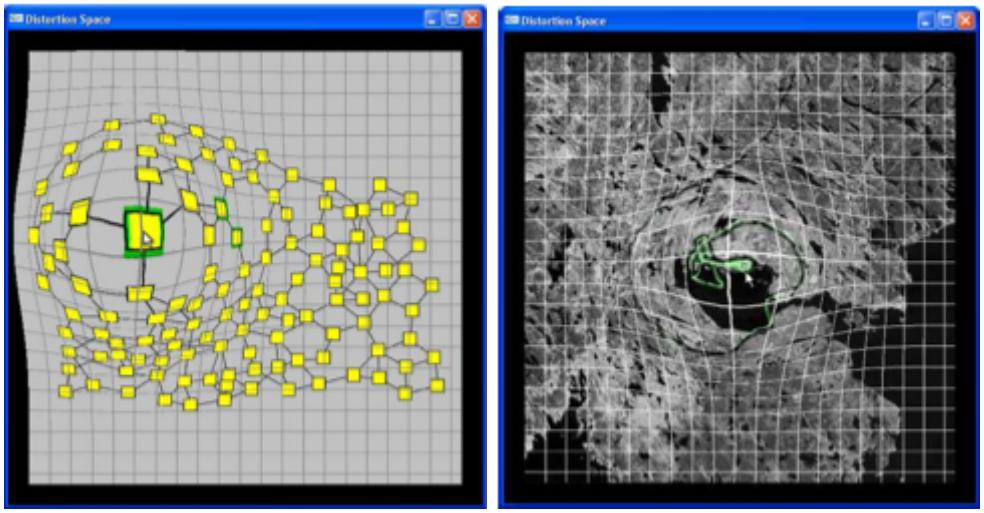


Abbildung 3.4: Visit Wear am Beispiel eines diskreten Datensets und eines kontinuierlichen Datensets. Besuchte Bereiche werden grün hervorgehoben.

prägen. Je stärker dabei die visuelle Verzerrung, desto schwieriger wird es, sich die Position von Elementen in der Datenstruktur zu merken.

Ein möglicher Lösungsansatz hierfür ist die Einführung von künstlichen Anhaltspunkten als Merkhilfe und zur Orientierung für den Anwender, wie z.B. Visit Wear (Skopik & Gutwin, 2005). Hierbei werden besuchte Elemente innerhalb der Fisheye View visuell hervorgehoben. Im Rahmen eines Experiments mit 16 Teilnehmern wurde der Effekt von Visit Wear auf diskrete und kontinuierliche Datenmengen untersucht (siehe Abbildung 3.4): Sowohl bei diskreten als auch kontinuierlichen Datensets war die Task Completion Time unter Verwendung von Visit Wear signifikant schneller als ohne. Darüber hinaus ließ sich bei Verwendung von Visit Wear ein signifikanter Effekt auf die Fehlerrate bei Identifikationsfehlern und Task Completion Fehlern feststellen. Sämtliche Testpersonen gaben an, Visit Wear für diskrete Datensets zu bevorzugen. Vier Testpersonen gaben an, Visit Wear für kontinuierliche Datensets nicht zu mögen, weil es Unruhe in die Darstellung der Daten bringt und eher verwirrend wirkt, während die restlichen zwölf Testpersonen auch für kontinuierliche Datensets Visit Wear den Vorzug gaben.

# Kapitel 4

# Related Work

Fisheye Views zur Visualisierung von Datenstrukturen haben eine lange Geschichte im Bereich des User Interface Designs, auch wenn sie bisher in verbreiteten kommerziellen User Interfaces nicht Fuß fassen konnten und nicht allgemein verbreitet sind. Bereits 1982 hat Furnas die Anwendbarkeit von Fisheye Views auf Interfacedesignprobleme diskutiert und eine Fisheye-DOI-Funktion formalisiert. Seither hat sich eine Vielzahl an Arbeiten tiefergehend mit der Thematik beschäftgit.

## 4.1 Generalized Fisheye Views

In *The Fisheye View: A New Look at Structured Files* formalisiert Furnas (1982) erstmals eine Fisheye-DOI-Funktion. Als Inspiration diente hierbei das Vorbild einer Fisheye Linse, welche das Zentrum im Detail darstellt, während Bereiche außerhalb des Zentrums immer weniger detailliert dargestellt werden, je weiter man sich vom Zentrum entfernt. Neben der Distanz vom Fokuspunkt bezieht Furnas auch die a-priori-Wichtigkeit von Informationen in seine DOI-Funktion mit ein. Die Fisheye-DOI-Funktion wird im Detail in Kapitel 2.4 dieser Arbeit vorgestellt.

In (Furnas, 1982) wird darüber hinaus die Anwendbarkeit der Fisheye-DOI-Funktion auf Interfacedesignprobleme hinsichtlich Baumstrukturen diskutiert. Furnas definiert die Komponenten der DOI-Funktion für Baumstrukturen folgendermaßen:

- Fokuspunkt: Ein Knoten in der Baumstruktur von momentanem Interesse
- Distanz vom Fokuspunkt: D(.,x) = d(.,x)
- Level of Detail: LOD(x) = -d(r,x), wobei r die Wurzel des Baums ist

Daraus ergibt sich folgende DOI-Funktion für Baumstrukturen bei simpler additiver Kombination der Komponenten:

$$DOI(x|.) = -(d(r,x) + d(.,x)) \tag{4.1}$$

Diese DOI-Funktion für Baumstrukturen lässt sich nun z.B. auf hierarchisch strukturierte sequentielle Dateien, wie z.B. strukturierten Source Code, Textoutlines oder Filehierarchie-Listings anwenden. Diese Fisheye-DOI-Funktion für Baumstrukturen wird auch in (Furnas, 1986) wieder aufgegriffen und anhand eines Beispielprogramms zur Darstellung von C-Programmcode illustriert (siehe Abbildung 4.1 und Abschnitt 4.2). Hierbei wird den einzelnen Zeilen Source Code ein DOI-Wert in Abhängigkeit von ihrer a-priori Wichtigkeit und der Distanz vom Fokus des Anwenders, in diesem Fall der momentan selektierten Zeile, zugewiesen. Die a-priori Wichtigkeit von Zeilen ergibt sich aus der hierarchischen Struktur der Datei, ersichtlich auch an der Einrückung der Zeilen. Umschließende Programmbefehle wie z.B. loop- und if-Statements haben dementsprechend eine höhere a-priori Wichtigkeit. Weiters wird Zeilen in unmittelbarer Nachbarschaft des Fokus ein höheres Interesse beigemessen. Zeilen mit einem DOI-Wert unter einer bestimmten Schwelle werden ausgeblendet und der dadurch gewonnene Platz wird dafür verwendet, um dem Anwender einen Überblick über den Fokus zu gewähren.


Abbildung 4.1: Flat-Window Ansicht eines C-Programmausschnitts. Am linken Rand befindet sich die Zeilennummerierung.


Abbildung 4.2: Fisheye View eines C-Programmausschnitts. Am linken Rand befindet sich die Zeilennummerierung, "..." indiziert ausgelassene Zeilen.

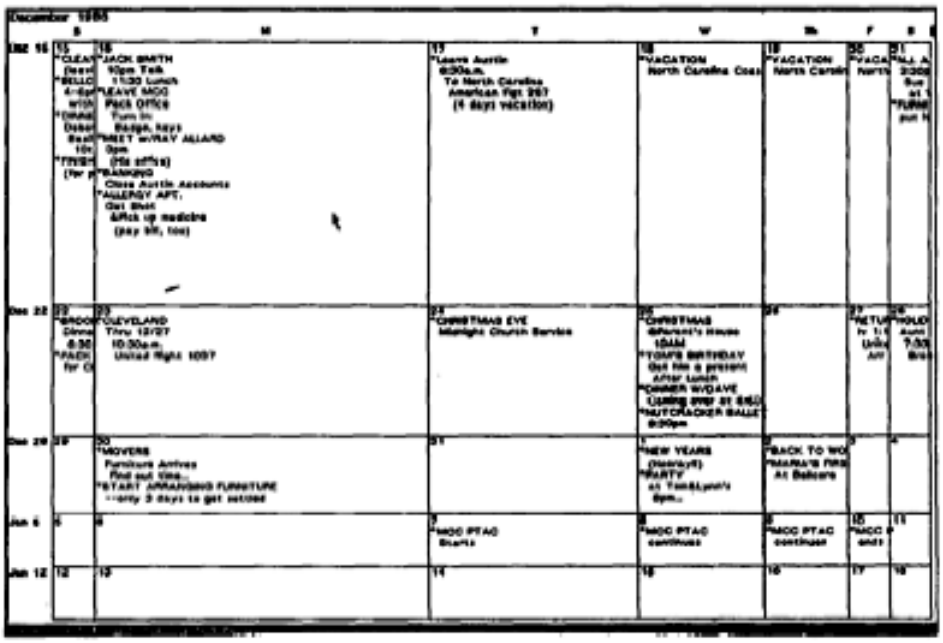

Abbildung 4.3: Fisheye Kalender

Darüber hinaus illustriert Furnas (1986) die allgemeine Anwendbarkeit des Fisheye View Konzepts auch anhand des Beispiels eines Fisheye Kalenders (siehe Abbildung 4.3)

Furnas (1986) hat auch natürlich auftretende Fisheye Views aus kognitionswissenschaftlicher Hinsicht untersucht und dabei festgestellt, dass viele natürlich auftretende Sichtweisen von Menschen Fisheye Charakteristika aufweisen. Für weitere Details hierzu siehe auch Kapitel 2.3.

Obwohl Furnas in seinen frühen Arbeiten (Furnas, 1982, 1986) konkret die Anwendbarkeit der Fisheye-DOI-Funktion auf Interfacedesignprobleme diskutiert, beschäftigt er sich wesentlich weniger mit der Frage der Informationsvisualisierung als viele spätere Arbeiten in diesem Bereich. Es steht nicht so sehr die Frage *wie* etwas dargestellt werden soll im Vordergrund, sondern *was* denn überhaupt dargestellt werden soll. Diesem Unterschied zwischen visueller Repräsentation und semantischem Filtern wird in (Furnas, 2006) hinsichtlich diverser Fokus+Kontext Visualisierungstechniken und Anwen-

dungen von Fisheye Views genauer auf den Grund gegangen. Furnas kommt hierbei zu dem Schluss, dass der Wert der Fisheye Metapher nicht allein in der Analogie der visuellen Verzerrung einer Fisheye Linse zu finden ist, sondern vielmehr in den Fisheye-DOI-Subsets, deren Anwendbarkeit über den Bereich der Informationsvisualisierung hinaus geht. Für weitere Details hierzu siehe auch Kapitel 2.5.

## 4.2 Fisheye Menus

Die Arbeit in Fisheye Menus (Bederson, 2000) steht dieser Arbeit besonders nahe. Bei Fisheye Menus (siehe Abbildung 4.4) handelt es sich um Dropdownmenüs, die sich einer eindimensionalen visuellen Fisheyeverzerrung bedienen, um Menüs mit vielen Einträgen kompakt am Bildschirm darzustellen. Fisheye Menus skalieren automatisch die Darstellungsgröße der Menüeinträge und erzeugen auf diese Weise einen Fokusbereich um den Mauszeiger. Dies ermöglicht die Darstellung eines großen Menüs mit vielen Menüeinträgen auf dem Bildschirm, ohne auf Scrollbalken, Scrolling Arrows oder Hierarchien zurückzugreifen.

Menüdesign ist ein in der Vergangenheit umfangreich untersuchter Bereich. Die Designtradeoffs beim Design von sorgfältig geplanten und einheitlich strukturierten Menüs sind allgemein bekannt (Norman, 1991). Die zunehmende Verbreitung von Menüs zur Auswahl von Datenelementen aus langen Listen mit nicht fest vorbestimmtem Inhalt war jedoch der ausschlaggebende Motivator für diese Arbeit. Ein Beispiel hierfür ist die Verwendung von Dropdownmenüs zur Auswahl von Bookmarks in verbreiteten Webbrowserprogrammen wie z.B. Microsoft Internet Explorer oder Mozilla Firefox. Solche Menüs zur Auswahl von Datenelementen unterscheiden sich von herkömmlichen, verbreiteten Menüs zur Auswahl von Funktionen und Befehlen darin, dass die Datenelemente im Menü häufigen Änderungen unterworfen sind und dass die Anzahl der Menüeinträge meist größer ist. Aus diesen häufigen Änderungen des Menüinhalts ergibt sich, dass die Größe


Abbildung 4.4: Fisheye Menu

des Menüs nicht exakt vorhersehbar ist. Darüber hinaus ist die Wahrscheinlichkeit größer, dass der Anwender mit Reihenfolge und Formulierung der Menüeinträge nicht vertraut ist.

Bei Fisheye Menus können sämtliche Menüeinträge in einem einzigen Menüfenster fixer Größe zu jedem Zeitpunkt am Bildschirm sichtbar dargestellt werden, und zwar unabhängig von der Anzahl der Menüeinträge. Dies wird ermöglicht, indem eine eindimensionale Fisheyeverzerrung auf die Menüeinträge angewandt wird. Menüeinträge in der Nähe des Mauszeigers werden in voller Größe dargestellt, während Menüeinträge in größerer Entfernung vom Mauszeiger sehr klein dargestellt werden. Bewegt man die Maus über das Menü, so werden die Menüeinträge dynamisch skaliert und ein Bereich mit groß dargestellten, lesbaren Menüeinträgen folgt dem Cursor. Durch die Darstellung des gesamten Menüs am Bildschirm wird dem Anwender die Orientierung erleichtert und der Anwender kann schnell und direkt zu weit entfernten Menüeinträgen springen.

Abbildung 4.5: Fisheye Menu im Focus Lock Modus: Der Fokusbereich wird nach oben hin erweitert.

Fisheye Menus bedienen sich alphabetischer Indizes auf der linken Seite des Menüs (siehe Abbildung 4.4), um den Anwender bei der Suche nach Menüeinträgen zu unterstützen. Durch die dynamische Skalierung der Darstellungsgröße der Menüeinträge ist es bei einer großen Anzahl von Einträgen möglich, dass viele Menüeinträge im Fisheyemenu unlesbar klein dargestellt werden. Die alphabetischen Indizes sollen es dem Anwender erleichtern, einen bestimmten Bereich zu identifizieren, in dem sich ein gewünschtes Auswahlziel befindet welches der Anwender sucht, selbst wenn der entsprechende Menüeintrag zu klein dargestellt wird um lesbar zu sein.

Um dem in Kapitel 3.3 beschriebenen Problem des radikal verringerten Motorspace bei der Auswahl von Elementen entgegenzuwirken bedient sich Bederson eines sogenannten Focus Lock Modus: Der Anwender bewegt den Mauszeiger in der linken Hälfte des Menüs, bis er sich dem Bereich nähert, in dem sich das Auswahlziel befindet. Solange sich der Mauszeiger in der linken Hälfte des Menüs befindet, verhält sich das Menü wie oben beschrieben:

Menüeinträge in unmittelbarer Nähe des Mauszeigers werden groß dargestellt und weiter entfernte Menüeinträge werden klein dargestellt. Sobald sich der Anwender in der Nähe des Zielmenüeintrags befindet bewegt er die Maus in die rechte Hälfte des Menüs, wodurch der Focus Lock Modus aktiviert wird. In diesem Modus wird der Fokus auf jenem Menüeintrag fixiert, über dem sich der Mauszeiger zum Zeitpunkt des Wechsels in die rechte Hälfte des Menüs befand. Wird der Mauszeiger nun außerhalb des bestehenden Fokusbereichs rund um den fixierten Fokuspunkt bewegt, so wird der Fokusbereich in die entsprechende Richtung erweitert (siehe Abbildung 4.5). Dies kann jedoch dazu führen, dass nicht mehr sämtliche Menüeinträge dargestellt werden können, weil Einträge im oberen und unteren Bereich durch die Erweiterung des Fokusbereichs quasi aus dem Menü "hinausgeschoben" werden. Will der Anwender den Focus Lock Modus wieder verlassen um zu einem anderen Bereich des Menüs zu navigieren, genügt es, die Maus wieder in die linke Hälfte des Menüs zu bewegen.

Zur Evaluierung der Fisheye Menus wurde eine Pilotstudie mit 10 Testpersonen durchgeführt. Im Rahmen dieser Studie wurden Fisheye Menus mit traditionellen Menüs, die Scrollbalken, Hierarchien oder Scrolling Arrow Buttons verwenden, verglichen. Ziel der Studie war es, die Präferenzen und Akzeptanz der Testpersonen herauszufinden. Die Testpersonen wurden gebeten, drei vorgegebene Menüeinträge auszuwählen. Die drei Menüeinträge wurden dabei so gewählt, dass sie sich jeweils am Anfang, in der Mitte und am Ende der Menüs befanden. Anschließend wurden die Testpersonen gebeten, die Menüs frei zu durchstöbern. Nach dem Test sollten die Testpersonen die vier unterschiedlichen Menüs auf einer neunteiligen semantisch-differentiellen Skala hinsichtlich sieben verschiedener Kriterien bewerten. Schlussendlich wurden die Testpersonen gebeten, die vier unterschiedlichen Menüs in der Reihenfolge ihrer Präferenzen sowohl für goal-directed Tasks als auch Browsingtasks zu reihen.

Die Pilotstudie kam zu dem Ergebnis, dass die durchschnittliche, subjektive Zufriedenheit der Testpersonen bei den hierarchischen Menüs am höchsten war, dicht gefolgt von Fisheye Menus auf Platz zwei und Menüs mit Scrollbal-

ken auf Platz drei. Menüs mit Scrolling Arrow Buttons schnitten mit Abstand am schlechtesten ab. Bei der Reihung der vier Menüarten nach persönlichen Präferenzen ergaben sich Unterschiede in der Reihung zwischen goal-directed Tasks und Browsingtasks: Für goal-directed Tasks konnte sich das hierarchische Menü knapp vor den Fisheye Menus behaupten, während Menüs mit Scrollbalken und Menüs mit Scrolling Arrow Buttons weit abgeschlagen auf Platz drei und vier landeten. Für Browsingtasks schnitten die Fisheye Menus am besten ab, gefolgt von Menüs mit Scrollbalken, hierarchischen Menüs und Menüs mit Scrolling Arrow Buttons. Zwei der Testpersonen gaben an, Fisheye Menus überhaupt nicht zu mögen, die anderen acht Testpersonen waren den Fisheye Menus grundsätzlich nicht abgeneigt. Die Studie zeigte, dass die meisten Testpersonen anfangs die Funktionsweise der alphabetischen Indizes nicht richtig verstanden. Nach kurzer Benutzung legte sich aber dieses Problem. Weiters verstanden die wenigsten Testpersonen die Funktionsweise des Focus Lock Modus, selbst nach längerer Benutzung der Fisheye Menus. Erst nach einer ausdrücklichen Erklärung der Funktionsweise wurde diese den Testpersonen verständlich. Bederson kommt zu dem Schluss, dass aufgrund der stark schwankenden Akzeptanz und Präferenzen hinsichtlich Fisheye Menus zwischen den einzelnen Testpersonen die Verwendung von Fisheye Menus in Anwendungsprogrammen auf jeden Fall optional sein sollte. Neben der Pilotstudie wurde darüber hinaus ein einfacher Expertentest durchgeführt, um die Zeitperformance der unterschiedlichen Menüs zu vergleichen. Die Aufgabe war hierbei, einen Menüeintrag möglichst schnell auszuwählen. Diese Aufgabe wurde im Rahmen des Tests mit jedem Menü zehnmal wiederholt. Hierbei erwies sich das hierarchische Menü als am schnellsten, gefolgt von Fisheye Menus, Menüs mit Scrollbalken und Menüs mit Scrolling Arrow Buttons.

Fisheye Menus fanden auch in (Huang & Quan, 2004) Anwendung: Hierbei wurden Fisheye Menus als Kontextmenüs mit einem Hyperbolic Tree Browser kombiniert, um die effiziente Nutzung des am Bildschirm zur Verfügung stehenden Platzes zu gewährleisten.

## 4.3 Weitere Anwendungsgebiete von Fisheye Views

In der Vergangenheit wurde das Prinzip von Fisheye Views auf unterschiedliche Problemstellungen angewandt. Beispiele für die eindimensionale Anwendung von Fisheye Views gibt es für Textdarstellung- und Bearbeitung, Listen, Bäume und Menüs (die Anwendung von Fisheye Views auf Menüs wurde bereits umfassend in Kapitel 4.2 präsentiert). Beispiele für die zweidimensionale Anwendung von Fisheye Views gibt es in der Graphenvisualisierung und zur Darstellung von Tabellen.

### 4.3.1 Fisheye Views zur Textdarstellung

Furnas (1982) hat sich bereits in *The Fisheye View: A New Look at Structured Files* mit der Anwendbarkeit von Fisheye Views auf strukturierte Texte in Form von Source Code beschäftigt (siehe hierzu auch Kapitel 4.1). Seit damals haben sich jedoch auch zahlreiche weitere Arbeiten mit dieser Thematik tiefergehend auseinandergesetzt: Die Arbeit von Jakobsen & Hornbæk (2006) folgt gewissermaßen direkt in Furnas' Fußstapfen und beschäftigt sich ebenfalls mit der Umsetzung einer Fisheye View zur Darstellung von Source Code. Im Rahmen der Arbeit wurde ein Editor unter Verwendung von Fisheye Visualisierungstechniken als Plugin für das Eclipse IDE entwickelt. Der Editor bedient sich des Overview+Detail Konzepts, im linken Teil des Fensters befindet sich eine Detailansicht, während im rechten Teil des Fensters ein Überblick über das Gesamte Dokument gegeben wird. Die Detailansicht selbst ist wiederum in einen Fokus- und einen Kontextbereich unterteilt, wobei der zur Verfügung stehende Platz gleichmäßig zwischen Fokus- und Kontextbereich aufgeteilt ist. Die Bearbeitung des Source Codes ist nur im Fokusbereich möglich und als Fokuspunkt wird nicht eine einzelne Zeile oder die Mausposition gewählt, sondern sämtliche Zeilen im Fokusbereich. Dies hat zur Folge, dass sich die Ansicht nur ändert, wenn der Benutzer scrollt,

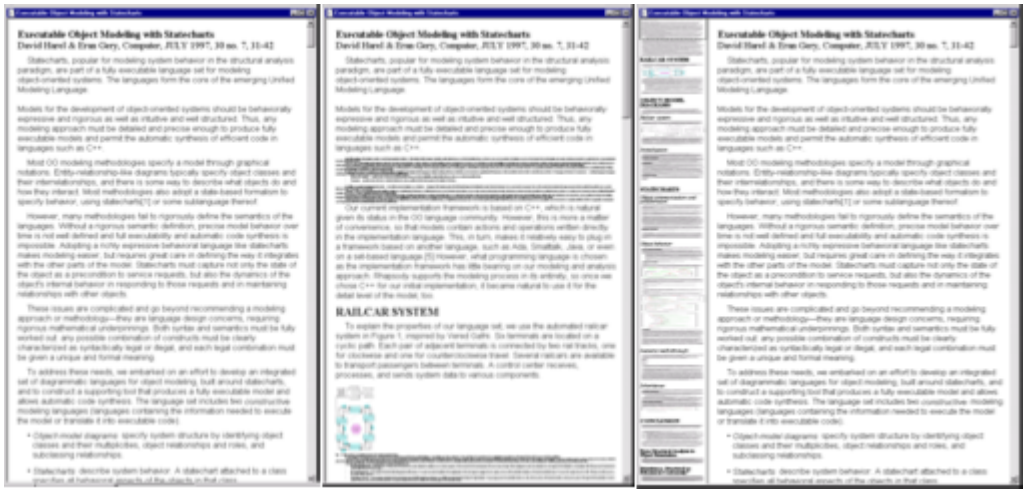

Abbildung 4.6: Unterschiedliche Interfaces zur Darstellung von Texten: Linear, Fisheye, Overview+Detail

und nicht wenn er z.B. die Cursorposition ändert. Dadurch wird erreicht, dass die Editoransicht relativ stabil bleibt und keinen zu häufigen Änderungen unterliegt, um dem Benutzer nicht die Wahrnehmung und das Verständnis der Ansicht zu erschweren. Die Darstellung von Text im Kontextbereich erfolgt in voller Größe um die Lesbarkeit zu wahren, wobei sich die Auswahl der Programmzeilen zur Darstellung im Kontextbereich in Abhängigkeit vom Fokusbereich ergibt. Im Rahmen eines kontrollierten Experiments mit 16 Teilnehmern wurde der Fisheye View Editor mit einem traditionellen Editor verglichen, in welchem der Sourcecode einfach linear dargestellt wurde. Das Experiment hat ergeben, dass es keinen signifikanten Unterschied in der Korrektheit der Taskerfüllung zwischen den Editoren gab. Die Task Completion Time war im Großen und Ganzen mit dem Fisheye Editor schneller, jedoch schwankten die Unterschiede zwischen den Task Completion Times stark in Abhängigkeit von der Art des zu erfüllenden Tasks. Die Testpersonen gaben an, den Fisheye Editor zu bevorzugen. Die Arbeit kommt zu dem Schluss, dass Fisheye Views zur Darstellung von Source Code vielversprechend sind.

Ein weiteres Beispiel für die Anwendbarkeit von Fisheye Views zur Textdarstellung findet sich in (Hornbæk & Frøkjær, 2001) und (Hornbæk &

Frøkjær, 2003). In diesen Arbeiten wurden drei unterschiedliche Interfaces zur Darstellung von Texten evaluiert: Ein traditionelles, lineares Interface, wie es für die Darstellung von Texten am Computer üblich und weit verbreitet ist, ein Fisheye Interface, bei dem weniger wichtige Textstellen verkleinert dargestellt werden bis der Benutzer darauf klickt, und ein Overview+Detail Interface, bei dem sich eine Übersicht über die gesamte Dokumentenstruktur in einer schmalen Leiste links vom Dokument befindet (siehe Abbildung 4.6). Im Rahmen eines Experiments mit 20 Teilnehmern wurden alle drei Interfaces evaluiert. Die Studie kam dabei zu dem Ergebnis, dass das Overview+Detail Interface am effektivsten ist. Darüber hinaus gaben 19 von 20 Testpersonen an, das Overview+Detail Interface den anderen Interfaces vorzuziehen. Es wurde festgestellt, dass die Testpersonen bei der Verwendung des Fisheye Interfaces die Dokumente am schnellsten zusammenfassen konnten, jedoch auch zu einem weniger umfassenden Verständnis des gelesenen Dokuments kamen. Weiters äußerten manche Testpersonen ihre Unzufriedenheit darüber, dass im Fisheye Interface der Computer vorherbestimmt, welche Teile des Dokuments wichtig sind und welche nicht. Die Studie kommt zu dem Schluss, dass von den drei evaluierten Interfaces das Overview+Detail Interface den Anwender am Besten beim Lesen von elektronischen Dokumenten am Bildschirm unterstützt. Weiters wird darauf hingewiesen, dass die Art der Visualisierung einen starken Einfluss auf Leseverhalten und Textverständnis hat.

Ein anderes Beispiel ist Fishnet (Baudisch et al., 2004), ein Webbrowser, der sich einer Fisheye Visualisierungsmethode bedient, um Webseiten komplett und ohne der Notwendigkeit zu scrollen am Bildschirm darstellt. Der Fokusbereich wird hierbei in Originalgröße und lesbar gerendert, während Text im Kontextbereich über und unter dem Fokusbereich räumlich komprimiert dargestellt wird. Der Kontextbereich wird hierbei dunkler dargestellt, um ihn visuell eindeutig vom Fokusbereich abzugrenzen. Um den Fokusbereich zu ändern bedient man sich eines Scrollbalkens: Bewegt man den Scrollbalken nach unten, so bewegt sich auch der Fokusbereich nach unten. Darüber hinaus unterstützt Fishnet Search Term Highlighting: Suchbegrif-

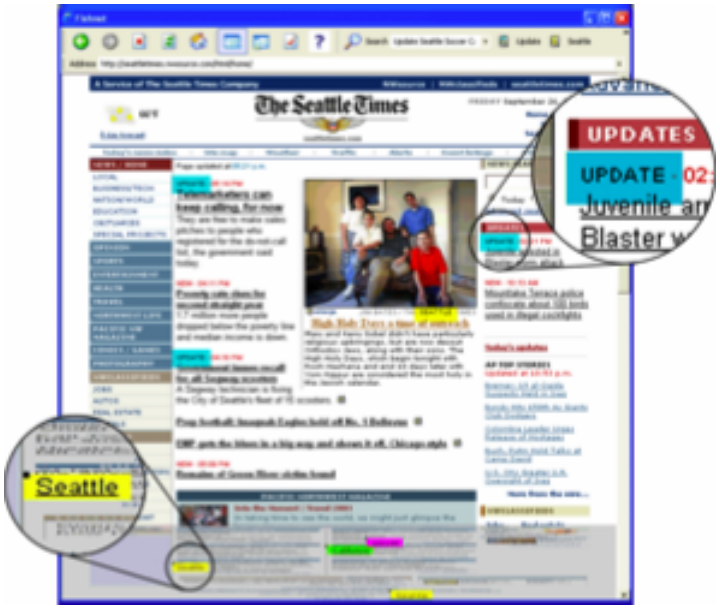


Abbildung 4.7: Fishnet Webbrowser: Der Kontextbereich im unteren Teil des Fensters wird dunkler dargestellt, Search Term Popouts sind farblich hinterlegt

fe werden überall in der Webseite farbig hinterlegt, um diese hervorzuheben. Suchbegriffe, die außerhalb des Fokusbereichs gefunden werden, werden ebenfalls farblich gut erkennbar hinterlegt und zusätzlich mittels Popouts hervorgehoben, um die Lesbarkeit selbst im Kontextbereich zu gewährleisten (siehe Abbildung 4.7). Ein Vorteil, der sich aus der Verwendung von Search Term Popouts in Verbindung mit einer Fisheye View des Texts ergibt, besteht darin, dass der Benutzer sofort einen kompletten Überblick über das gesamte Dokument erhält und ob der gesuchte Begriff darin enthalten ist ohne scrollen zu müssen. Im Rahmen einer Userstudie wurde Fishnet mit einem Overview+Detail Webbrowser und einem Browser mit einfacher linearer Darstellung der Seiten verglichen. Insgesamt 13 Testpersonen nahmen an der Studie teil. Die Studie ergab, dass die Vorzüge von jedem der drei Interfaces sehr stark von der Art des zu erfüllenden Tasks abhängig sind. Es zeigte sich, dass das Fisheye Interface insbesondere für Tasks geeignet ist, bei denen der Anwender nicht zwischen einzelnen Zeilen (wie z.B. Tabel-

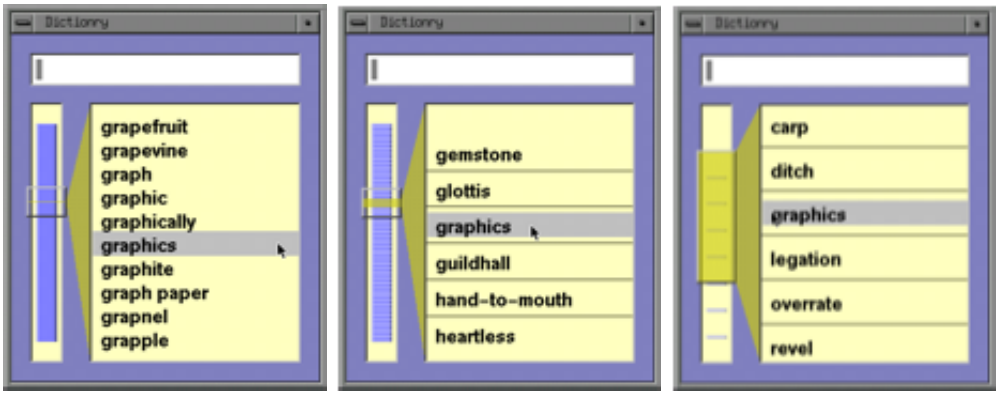


Abbildung 4.8: Lensbar Wörterbuchapplikation in drei unterschiedlichen Zoomstufen: Im ersten Bild wird jeder Eintrag angezeigt, im zweiten Bild jeder 256. Eintrag und im dritten Bild jeder 4096. Eintrag (v.l.n.r)

lenzeilen) eindeutig unterscheiden muss, weil bei solchen Aufgaben die Informationen im Kontextbereich aufgrund der visuellen Verzerrung stark an Aussagekraft verlieren. Insgesamt erwies sich der Fishnet Browser hinsichtlich Taskerfüllungszeit als die schnellste der drei getesten Varianten. Dieses Ergebnis wurde jedoch nicht von der subjektiven Präferenzbewertung der Testpersonen widergespiegelt: Während alle Testpersonen den Webbrowser mit Overview+Detail Darstellung mochten, polarisierte der Fisheye View Webbrowser: Drei Testpersonen gaben an, dieses Interface zu bevorzugen, während sechs Testpersonen eine starke Abneigung gegen dieses Interface äußerten. Ob sich diese Präferenzen nach längerfristiger Verwendung dieses ungewohnten Interfaces ändern könnten, müsste im Rahmen einer Langzeitstudie evaluiert werden.

### 4.3.2 Fisheye Views zur Listendarstellung

Lensbar (Masui, 1998) bedient sich Fisheye Views zur Listendarstellung. Hierbei dient die Fisheye View jedoch nicht als Grundlage für eine visuelle Verzerrung, sondern kommt als semantischer Filter zum Einsatz. Lensbar ist ein Tool zur Filterung und Visualisierung von langen Listen mit

dem Ziel, möglichst allgemein anwendbar und domänenunabhängig zu sein. Lensbar besteht aus einem Scrollbar und einem Scrollfenster, in dem Listeneinträge dargestellt werden. Eine Textbox dient dazu, Einträge mittels unscharfem Matching aus der Liste zu filtern. Zusätzlich bietet Lensbar eine Zoomfunktion: Sämtlichen Listeneinträgen wird ein DOI-Wert zugewiesen und sämtliche Einträge, deren DOI-Wert unterhalb des aktuellen Zoomlevels liegt werden ausgeblendet. Dem momentan ausgewählten Listeneintrag wird grundsätzlich ein besonders großer DOI-Wert zugewiesen um die Sichtbarkeit zu gewährleisten. Der Anwender kann den Zoomlevel ändern, indem er auf einen Listeneintrag klickt und entweder nach links dragt, um den Zoomlevel zu verkleinern bzw. nach rechts dragt, um den Zoomlevel zu erhöhen. Abbildung 4.8 zeigt die Funktionsweise dieser Zoomfunktion anhand einer Wörterbuchapplikation.

### 4.3.3 Fisheye Views zur Graphenvisualisierung

Sarkar & Brown (1992) diskutiert die Anwendbarkeit von Fisheye Views zur Graphenvisualisierung. Die Arbeit stellt eine graphische Interpretation von Fisheye Views vor und beschäftigt sich mit den dazugehörigen Layoutüberlegungen. Die Autoren stellen zuerst ein formelles Modell für visuell verzerrte Fisheye Views von Graphen vor und präsentieren danach zwei mögliche Implementierungsstrategien für das Modell, einerseits auf Basis einer kartesischen Transformation, andererseits auf Basis einer polaren Transformation. Im Rahmen der Arbeit wurde weiters ein Prototyp zur Darstellung von Fisheye Views für Graphen entwickelt.

Schaffer et al. (1996) haben die Anwendbarkeit von Fisheye Views zur Visualisierung von Clustered Networks untersucht und mit Zoommethoden verglichen. Die präsentierte Methode arbeitet mit zweidimensionalen Netzwerken bestehend aus Knoten und Verbindungen, deren Knoten hierarchisch geclustered sind. Die dem Clustering zugrundeliegende Hierarchie wird dann verwendet, um einzelne Cluster innerhalb des Netzwerks in unterschiedli-

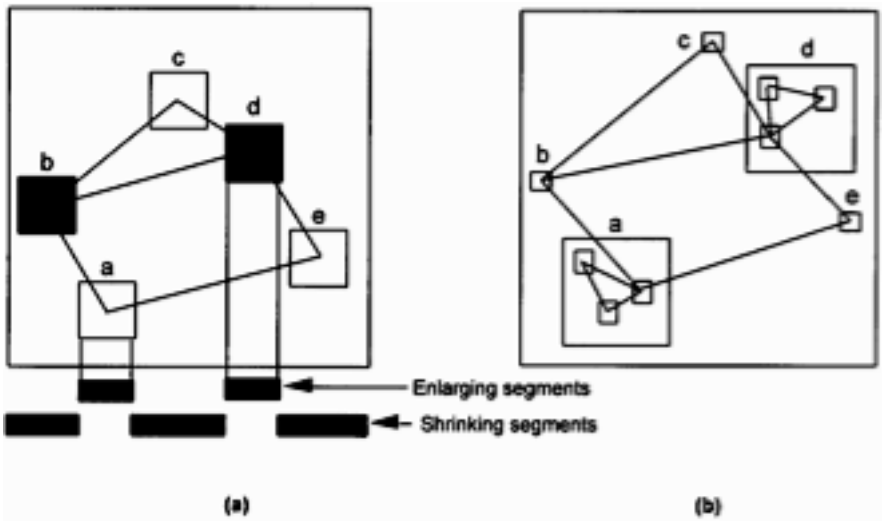


Abbildung 4.9: Fisheye View eines hierarchisch geclusterten Netzwerks. (a) zeigt das Netzwerk vor einer Zoom-Operation. (b) zeigt das Netzwerk nach der Expansion der Subnetzwerke a und d.

chen Ebenen von hierarchischen Details darzustellen. Auf jedem Level über dem Netzwerklevel werden Cluster als Icons dargestellt, die erweitert werden können, um den nächsten tiefergelegenen Level an Details anzuzeigen (siehe Abbildung 4.9). Im Rahmen eines Experiments mit 20 Testpersonen wurde diese Fisheye-Navigationsmethode mit einer Full Zoom-Methode verglichen. Bei der Full Zoom-Methode wurden ausgewählte Knoten so erweitert, dass sie den gesamten Bildschirm ausfüllten. Der einzige Unterschied aus Benutzersicht zwischen den beiden Methoden bestand in der Art der Visualisierung. Die Studie kam zu dem Ergebnis, dass die Taskerfüllungszeit bei der Fisheye-Methode signifikant schneller war als mit der Full-Zoom Methode. Es gab keine signifikanten Unterschiede zwischen den beiden Methoden hinsichtlich Erfolgsrate. Darüber hinaus stellte sich heraus, dass die Testpersonen bei der Verwendung der Fisheye Navigationsmethode signifikant weniger Zoomoperationen zur Erfüllung der Tasks durchführten. Die meisten Testpersonen gaben an, die Fisheye-Methode aufgrund des zusätzlichen Kontexts, den diese bietet, zu bevorzugen.

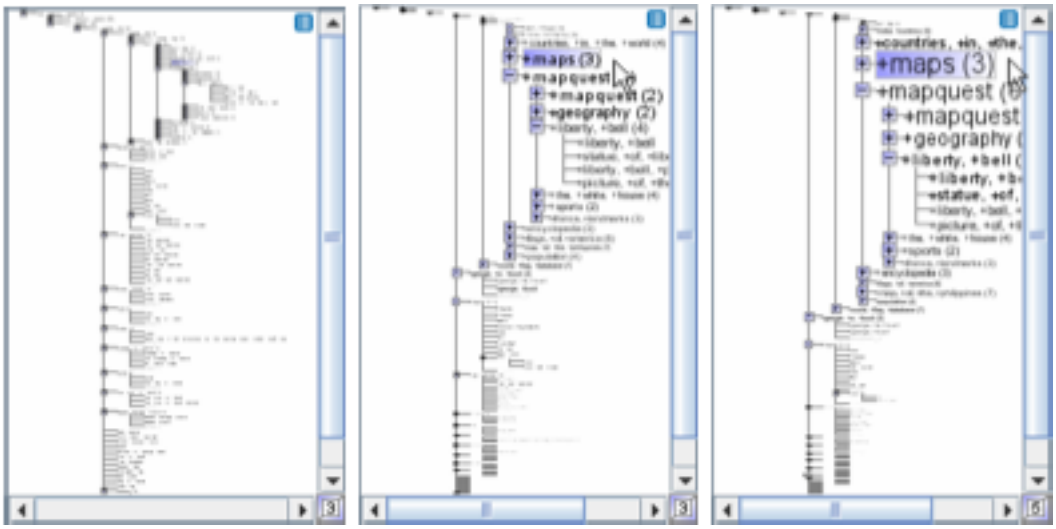


Abbildung 4.10: Textueller Fisheye Tree

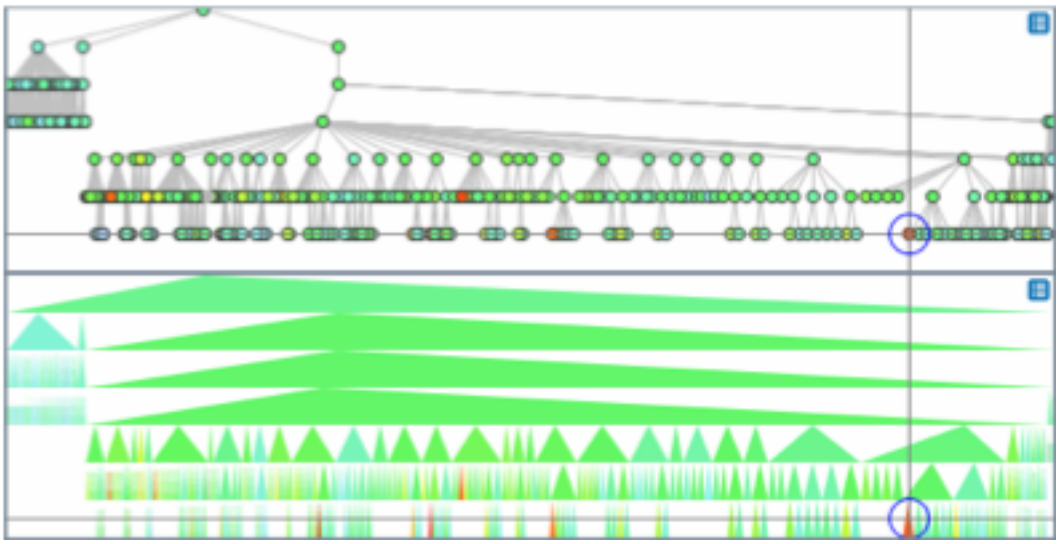

Abbildung 4.11: Nontextueller Fisheye Tree

Tominski et al. (2006) versucht unter anderem Overview+Detail und Focus+Kontext Visualisierungsmethoden für die Graphenvisualisierung zu kombinieren. Hierarchische Strukturen wie z.B. hierarchische Bäume sind für die Visualisierung von geclusterten Graphen von besonderer Bedeutung, deshalb wurden im Rahmen dieser Arbeit zwei Fisheye Tree Views für diesen Anwendungsbereich entwickelt. Bei den beiden Fisheye Tree Views handelt es sich einerseits um eine textuelle Fisheye Tree View (siehe Abbildung 4.10) und andererseits eine non-textuelle Fisheye Tree View (siehe Abbildung 4.11). Die Autoren weisen insbesondere auf die breite Anwendbarkeit der textuellen Fisheye Tree View hin. Die textuelle Fisheye Tree View bedient sich einer eindimensionalen Fisheye Verzerrung ähnlich jener der Fisheye Menus (Bederson, 2000). Der Fisheye Verzerrungseffekt wird in dieser Implementierung wie auch bei den Fisheye Menus vertikal angewandt. Die non-textuelle Fisheye Tree View bedient sich ebenfalls einer eindimensionalen Fisheye Verzerrung, jedoch erfolgt die Fisheye Transformation in diesem Fall in horizontaler Richtung.

### 4.3.4 Fisheye Views zur Darstellung von tabellarischen Daten

TableLens (Rao & Card, 1994) ist eine Fisheye- Visualisierungsmethode zur Darstellung von Tabellen. TableLens unterstützt die Interaktion mit großen Tabellen und kombiniert die symbolische und graphische Repräsentation von Daten in einer einzigen, kohärenten Ansicht. Dies hilft dem Anwender dabei, Muster in den Tabellendaten ausfindig zu machen. TableLens ändert das Layout von Tabellen ohne Zeilen und Spalten zu "verbiegen", um die kohärente Darstellung von Zeilen und Spalten zu bewahren. Die Platzzuweisung der einzelnen Zellen erfolgt dabei unabhängig in zwei Dimensionen. Dennoch ergibt sich eine Interaktion beider Dimensionen in der Geometrie der Zellen. Es ergeben sich vier unterschiedliche Arten von Zellregionen durch die unterschiedliche Platzzuweisung entlang zweier Achsen: Fokusregionen, Zeilenfokusregionen, Spaltenfokusregionen und Nonfokusregionen. Fokuszellen

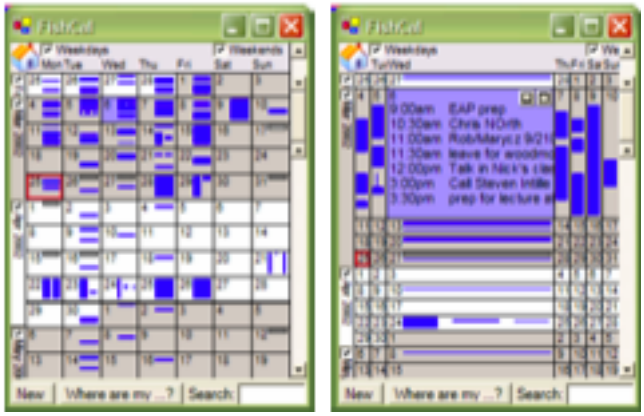

Abbildung 4.12: DateLens: Das Interface ist so konfiguriert, dass 12 Wochen in der Übersicht dargestellt werden. Durch die Auswahl eines Tages wird dieser größer dargestellt.

sind entlang beider Achsen vergrößert, Zeilenfokuszellen und Spaltenfokuszellen sind entlang jeweils einer Achse vergrößert und Nonfokuszellen sind überhaupt nicht vergrößert. Die Auswahl und Manipulation des Fokusbereichs erfolgt durch direkte Interaktion mit der Tabelle anhand von Keyboardbefehlen und Zeigergesten.

DateLens (ehemals FishCal benannt) (Bederson et al., 2003, 2004) ist eine Kalenderapplikation für PDAs, die sich Fisheye Views zur Visualisierung der Daten bedient. Zusätzlich unterstützt die Applikation Anwender durch die kompakte Darstellung von Daten, präzise Kontrolle über die angezeigte Zeitperiode und eine integrierte Suchfuntion. DateLens folgt vom grundsätzlichen Ansatz her dem Beispiel für ein Kalenderinterface, welches bereits von Furnas vorgestellt wurde (siehe auch Kapitel 4.1): Eine tabellarische Übersicht zeigt einzelne Tage und die Auswahl eines bestimmten Tages führt dazu, dass dieser Tag größer in der Tabelle dargestellt wird (siehe Abbildung 4.3 und Abbildung 4.12). Die Anzeige beginnt mit einer Übersicht über eine längere Zeitperiode. Durch die Auswahl eines Tages wird dieser zum Fokuselement und eine detailierte Liste der Termine an diesem Tag wird angezeigt. Das Interface zeigt im Gegensatz zu vielen anderen Kalenderapplikationen unterschiedliche Zeitspannen innerhalb eines einheitlichen Frameworks mit Hilfe

von animierten Übergängen, was dem Anwender dabei behilflich sein kann den Überblick zu behalten, wo er gerade ist. Für die visuelle Repräsentation von Terminen kommt ein semantischer Zoom zum Einsatz. Dadurch entfällt die Notwendigkeit von unterschiedlichen Ansichtsmodi, die Art der Darstellung von Terminen ist ausschließlich vom zur Verfügung stehenden Platz abhängig. Im Rahmen einer Benutzerstudie wurde die Usability von DateLens im Vergleich zu Microsoft Pocket PC 2002 Calendar evaluiert. Insgesamt nahmen 11 Testpersonen an der Studie teil und die Testpersonen wurden so ausgewählt, dass sie zum Zeitpunkt der Studie keinen PDA besaßen oder in laufender Verwendung hatten. Die Tests wurden in einem Pocket PC Emulator am PC durchgeführt, weil sich DateLens zum Zeitpunkt der Studie als zu ressourcenintensiv für die praktische Verwendung am Pocket PC erwies. Die Studie ergab, dass die Testpersonen die Tasks mit DateLens schneller erfüllen konnten, wobei dieses Ergebnis nur knapp signifikant war. Darüber hinaus war die Erfolgsrate für die Erfüllung der Tasks unter Verwendung von DateLens signifikant höher. Eine Umfrage zur Zufriedenheit der Testpersonen ergab keine signifikanten Unterschiede in der Zufriedenheit mit den beiden Programmen. Eine Folgestudie mit einer verbesserten Version von DateLens und erfahrenen Pocket PC Anwendern wurde direkt auf Pocket PCs anstatt in einem Emulator am PC durchgeführt. Die Studie lief über einen Zeitraum von drei Tagen und insgesamt 8 Testpersonen nahmen an ihr Teil. Nach Ablauf der drei Tage füllten die Testpersonen einen Fragebogen aus und nahmen an einem Experiment teil, in dessen Rahmen wiederum DateLens mit dem Pocket PC 2002 Calendar verglichen wurde. Die Studie fand keine signifikanten Unterschiede in der zeitlichen Performance zwischen beiden Applikationen. Die Umfrage zu den Präferenzen der Testpersonen ergab ebenfalls keine signifikanten Unterschiede zwischen den beiden Applikationen. In beiden Studien zeigte sich jedoch die Tendenz, dass DateLens eine bessere Performance für komplexe Tasks bietet, während der Pocket PC 2002 Calendar Performancevorteile bei einfachen Tasks bietet.

AppLens (Karlson et al., 2005) ist eine Applikation für PDAs, die dem Anwender ein Gitter bestehend aus 3x3 Zellen präsentiert. Jeder der neun Zellen

des AppLens-Gitters ist eine Applikation zugeordnet. In jeder der Zellen werden Informationen von einer der Applikationen dargestellt. Durch Auswahl einer Zelle wird diese zum Fokus der Darstellung und in Folge wird dieser Fokuszelle mehr Platz am Bildschirm zugewiesen, um umfangreichere Informationen darstellen zu können, während die anderen Zellen entsprechend verkleinert werden. Die Applikation ist laut Aussage der Autoren stark von DateLens beeinflusst und folgt einem ähnlichen Ansatz für die Darstellung einer tabellarischen Fisheyestruktur am PDA.

# Kapitel 5

# Design und Implementierung

Zur Evaluierung der Usability von eindimensional visuell verzerrten Fisheye Views in graphischen User Interfaces war es nötig, User Interface Elemente zu entwickeln, die sich einer solchen Verzerrung bedienen. Die Wahl fiel hierbei auf Fisheye View Listen, weil diese sich für die Anwendung einer eindimensionalen visuellen Verzerrung anbieten und diverse Interaktionsmöglichkeiten wie z.B. Selektion und Drag & Drop Operationen unterstützen. Darüber hinaus sind Listen ein etabliertes, verbreitetes User Interface Element, welches in vielen weitläufig verbreiteten Applikationen zum Einsatz kommt. Mit diesen Fisheye View Listen wurde im Anschluss ein Testprogramm zur Durchführung der Usability Tests entwickelt.

Das Design der im Rahmen dieser Arbeit implementierten Fisheye View Liste und der zugrundeliegende Algorithmus orientieren sich stark an Design und Implementierung der Fisheye Menus von Bederson (2000). Bei diesem Design werden sämtliche Listenelemente zu jedem Zeitpunkt vollständig am Bildschirm angezeigt, indem Listenelemente im Fokusbereich groß dargestellt werden, während Listenelemente außerhalb des Fokusbereichs zunehmend kleiner dargestellt werden. Der Fokusbereich ergibt sich hierbei in Abhängigkeit von der Position des Mauszeigers und die Darstellung der Liste ändert sich dynamisch in Abhängigkeit von Cursorbewegungen.

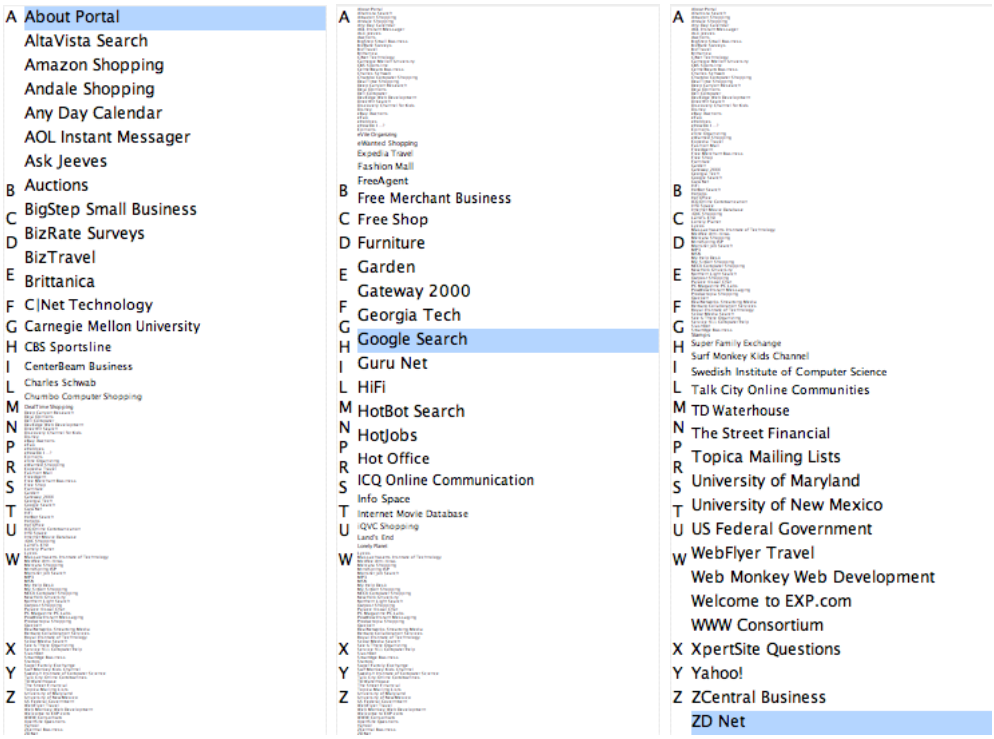


Abbildung 5.1: Fisheye View Liste mit dem Mauszeiger an drei unterschiedlichen Positionen: Einmal im oberen Bereich der Liste, einmal in der Mitte und einmal im unteren Bereich. Der Fokusbereich befindet sich jeweils in der Umgebung des Mauszeigers.

## 5.1 Designüberlegungen

Die im Rahmen dieser Arbeit entwickelte Fisheye View Liste hat den Zweck, lange Listen mit einer großen Anzahl an Listenelemten ohne Scrollbalken kompakt und vollständig am Bildschirm darzustellen. Sämtliche Listenelemente werden zu jedem Zeitpunkt am Bildschirm gleichzeitig dargestellt. Dies wird dadurch erreicht, dass Listenelemente in der Nähe des Fokuspunkts, in diesem Fall dem Mauszeiger, vergrößert dargestellt werden, während Listenelemente in größerer Entfernung vom Fokuspunkt kleiner dargestellt werden. Neben der Schriftgröße werden auch die Abstände zwischen den Listenelementen skaliert. Bei einer Mauszeigerbewegung bewegt sich der Fokusbereich

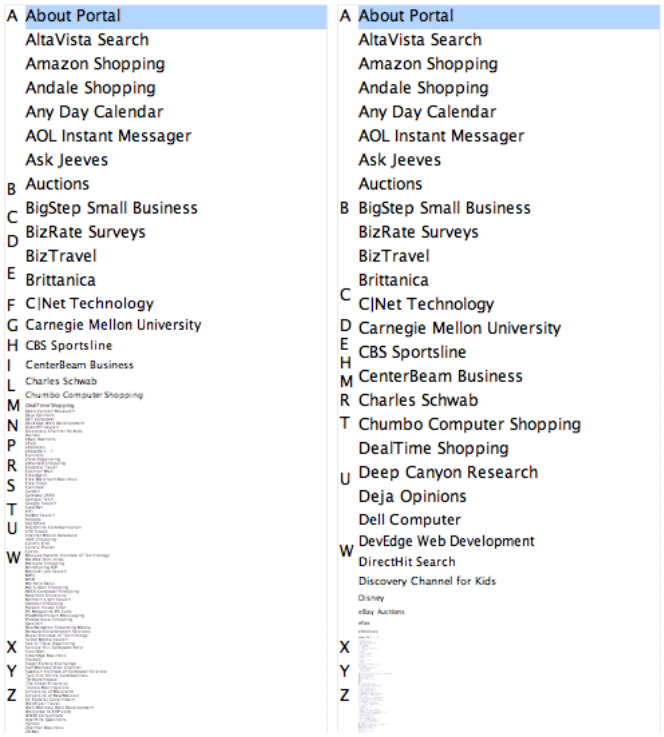


Abbildung 5.2: Fisheye View Listen mit unterschiedlich großen Fokusbereichen: Im linken Bild ist die Fokuslänge 7, im rechten Bild 20.

dynamisch mit der Position des Mauszeigers mit und die Darstellungsgröße der Listenelemente wird dynamisch skaliert. Der dabei auftretende Effekt ähnelt einer "Vergrößerungsblase", die dem Mauszeiger folgt. Für eine Illustration dieses Effekts siehe auch Abbildung 5.1. Auf diese Art und Weise ist es möglich, selbst lange Listen komplett sichtbar und ohne der Notwendigkeit zu scrollen am Bildschirm darzustellen, ohne die Lesbarkeit von und Interaktion mit einzelnen Listenelementen zu behindern.

Die hier präsentierte Implementierung einer Fisheye View Liste nutzt den gesamten zur Verfügung stehenden Screenspace, um die Liste in jedem Fall in ihrer Gesamtheit darzustellen. Es wurde bewußt davon abgesehen, einen schwächeren Skalierungseffekt unter gleichzeitiger Verwendung eines Scrollmechanismus für das Design zu verwenden, um dem Ideal einer Verschmelzung von Fokus+Kontext treu zu bleiben. Das Erscheinungsbild der Fisheye

View Liste kann im Wesentlichen durch zwei Parameter beeinflusst werden: Die maximale Schriftgröße für Listenelemente und die Größe des Fokusbereichs bzw. Fokuslänge. Die maximale Schriftgröße gibt an, wie groß Listenelemente im Fokusbereich dargestellt werden sollen und ist die größte Schriftgröße, die zur Darstellung der Liste zur Anwendung kommt. Die Fokuslänge gibt an, wieviele Listenelemente in der Umgebung des Fokuspunkts in voller Größe dargestellt werden sollen (siehe auch Abbildung 5.2). Die Wahl des Fokuslängenparameters beeinflusst direkt die Größe des Fokusbereichs und damit auch, wie viel Platz zur Darstellung von peripheren Listenelementen zur Verfügung steht. Die Fisheye DOI Funktion berechnet automatisch die Skalierung der einzelnen Listenelemente auf Basis des zur Verfügung stehenden Platzes am Bildschirm und je größer der Fokusbereich ausfällt, desto weniger Platz bleibt für die Darstellung von peripheren Listenelementen übrig. Bei Verwendung einer sehr großen Fokuslänge bleibt nur wenig Platz für die Darstellung der peripheren Listenelemente und der bei Fokusänderungen auftretende Verzerrungseffekt ist stärker. Bei Verwendung einer sehr kleinen Fokuslänge verhält sich die Darstellung der Liste "stabiler" und es steht mehr Platz für die Darstellung von peripheren Listenelementen zur Verfügung, jedoch werden auch weniger Listenelemente ausreichend groß dargestellt, um gut lesbar zu sein, was sich negativ auf die Übersichtlichkeit der Liste auswirken kann.

In Anlehnung an die Implementierung der Fisheye Menus von Bederson (2000) wurden alphabetische Indizes an der linken Seite der Fisheye View Liste implementiert. Aufgrund der starken Skalierung der Schriftgröße unterschiedlicher Listenelemente ergibt sich zwangsläufig, dass bei Listen ab einer gewissen Länge nicht mehr alle Listenelemente außerhalb des Fokusbereichs lesbar sind. Je nach Größe des Fokusbereichs und der Länge der Liste bedeutet dies, dass mitunter ein Großteil der Listenelemente im peripheren Darstellungsbereich zu klein dargestellt wird, um vom Anwender richtig wahrgenommen werden zu können. Deshalb ist es notwendig, der dargestellten Liste eine gewisse vorhersehbare Struktur zu geben, um zu verhindern, dass der Anwender dazu gezwungen wird, die Liste quasi im "Blindflug" abzusu-

chen. Eine einfache und praktische Möglichkeit einer Liste eine vorhersehbare Struktur zu verleihen, die in vielen Verwendungsszenarien anwendbar ist, besteht in der alphabetischen Sortierung der Listeneinträge. Selbst wenn nur ein kleiner Fokusbereich einer langen Liste für den Anwender lesbar ist, so ist bei alphabetischer Sortierung auf Basis der Kenntnis des Alphabets noch immer offensichtlich, ob sich ein bestimmter Listeneintrag nun ober- oder unterhalb des Fokusbereichs befindet. Der Anwender weiß dementsprechend zumindest ungefähr, in welche Richtung er navigieren muss und kann das gewünschte Listenelement genau ausfindig machen, nachdem er den ungefähren Bereich gefunden hat, in dem sich das gesuchte Listenelement befindet.

Die Verwendung von alphabetischen Indizes, die natürlich nur bei alphabetisch sortierten Listen Sinn macht, soll den Anwender bei seiner Suche nach einem Listenelement noch weiter unterstützen. Durch die alphabetischen Indizes wird dem Anwender nicht nur vermittelt, ob sich ein Listenelement ober- oder unterhalb des momentanen Fokusbereichs befindet, es wird ihm auch angezeigt, in welchem ungefähren Bereich sich das gesuchte Listenelement befindet. Die alphabetischen Indizes werden hierbei in einem Bereich links von der Fisheye View Liste dargestellt. Es wird jeder Buchstabe angezeigt, der zumindest einmal als Anfangsbuchstabe eines Listenelements in der Liste enthalten ist und für den Platz ist. Neben ihrer Funktion als Orientierungshilfe erfüllen die alphabetischen Indizes auch eine gewisse Gruppierungsfunktion und vermitteln dem Anwender, wieviele Einträge mit jedem Anfangsbuchstaben in der Liste enthalten sind.

Das Design der alphabetischen Indizes entspricht exakt dem Design der alphabetischen Indizes der Fisheye Menus (Bederson, 2000): Jeder Indexbuchstabe wird so positioniert, dass bei einer Mausbewegung zur vertikalen Position des Indexbuchstabens das erste Element mit diesem Anfangsbuchstaben sich auf exakt derselben Höhe befindet. Auf diese Weise ist es dem Anwender möglich, schnell zu dem ungefähren Bereich zu navigieren, in dem sich ein gesuchtes Listenelement befindet. Aus diesem Design ergibt sich auch, dass die Anzeige der alphabetischen Indizes unabhängig vom Fokuspunkt und völlig statisch ist. Bei einem anderen Design der Indizes, welches Be-

derson (2000) präsentiert, befanden sich die alphabetischen Indizes immer auf derselben Höhe wie das erste Element mit diesem Anfangsbuchstaben. Dementsprechend veränderte sich die Position der Indizes jedes Mal, wenn sich die Position des entsprechenden Listeneintrags änderte. Wenn ein Anwender nun zu einem Indexbuchstaben navigieren wollte war es oft der Fall, dass sich dieser Index noch während des Navigationsvorgangs bereits wieder verschoben hatte.

Im Gegensatz zu Fisheye Menus (Bederson, 2000) implementieren die hier präsentierten Fisheye View Listen keinen Focus Lock Modus (für eine genaue Beschreibung dieser Funktion siehe auch Kapitel 4.2), weil sich diese Funktion in den von Bederson durchgeführten Tests als äußerst unintuitiv und schwer zugänglich erwiesen hat.

## 5.2 Algorithmus

Der den in dieser Arbeit entwickelten Fisheye View Listen zugrundeliegende Algorithmus basiert auf dem Algorithmus der Fisheye Menus von Bederson (2000). Grundlage einer Fisheye View ist üblicherweise eine Fisheye DOI Funktion, wie in Kapitel 2.4 beschrieben. Die Fisheye DOI Funktion erfüllt die Aufgabe, für jedes Element einer Datenstruktur einen Degree of Interest (DOI), also das Interesse des Benutzers an einem Element zu berechnen. Eine Degree of Interest Funktion besteht üblicherweise aus zwei Komponenten: Der a-priori Wichtigkeit des Elements und der Distanz zwischen Element und Fokuspunkt. Nach der Berechnung der DOI-Werte für alle Elemente einer Datenstruktur kann mit diesen Werten auf unterschiedliche Weise verfahren werden: Man kann die DOI-Werte beispielsweise dazu verwenden, Elemente mit niedrigem Degree of Interest vollständig zu filtern, man kann die Größe der Darstellung von Elementen am Bildschirm von ihrem Degree of Interest abhängig machen oder auch beide Methoden kombinieren.

Bei den hier präsentierten Fisheye View Listen werden die DOI-Werte

dazu verwendet, die Darstellungsgröße der Listenelemente am Bildschirm zu skalieren, um eine eindimensionale visuelle Fisheyeverzerrung zu erzielen. Die verwendete DOI-Funktion basiert auf der DOI-Funktion von Fisheye Menus (Bederson, 2000) und ist sehr einfach gehalten: Die verwendete DOI-Funktion bezieht keine a-priori Wichtigkeit der Elemente in ihre Berechnung mit ein, d.h. sämtliche Elemente verfügen grundsätzlich über die selbe a-priori Wichtigkeit. Dieser Ansatz ist insofern sinnvoll, als der Liste eine beliebige eindimensionale Datenstruktur von Elementen zugrundeliegt und keine einfache programmatische Möglichkeit zur Bestimmung der a-priori Wichtigkeit der Elemente bei völlig unbekannten Datenquellen möglich ist. Für die Berechnung der DOI-Werte ist bei der verwendeten DOI-Funktion dementsprechend nur die Distanz vom Fokuspunkt von Bedeutung. Als Fokuspunkt gilt jenes Element, über dem sich momentan der Mauszeiger befindet. Die Distanz vom Fokuspunkt entspricht der Anzahl der Elemente, die zwischen einem Element und dem Fokuselement liegen. Die Distanzfunktion lässt sich entsprechend folgendermaßen formulieren:

Sei . die Position des momentanen Fokuselements und x die Position des Elements, für welches die Distanz bestimmt werden soll, so ist die Distanzfunktion

$$D(., x) = abs(. - x) \tag{5.1}$$

Dadurch, dass keine a-priori Wichtigkeit der Elemente in die DOI-Funktion miteinfließt, entspricht die DOI-Funktion der Distanzfunktion. Auf Basis der DOI- bzw. Distanzfunktion wird nun die Darstellungsgröße von Listenelementen folgendermaßen berechnet:

- Falls D(.,x) $\leq$ Fokuslänge / 2, dann: Schriftgröße = maximale Schriftgröße
- Falls D(.,x) > Fokuslänge / 2, dann:
  Schriftgröße = maximale Schriftgröße - D(..x)
- Falls maximale Schriftgröße - D(.,x) < minimale Schriftgröße, dann:
  Schriftgröße = minimale Schriftgröße

Auf diese Weise werden Listenelemente in der Nähe des Fokuspunkts in voller Größe dargestellt. Die Anzahl der in voller Größe dargestellten Listenelemente ist abhängig von der definierten Fokuslänge. Elemente außerhalb des Fokusbereichs werden stufenweise in Abhängigkeit von ihrer Distanz zum Fokuspunkt kleiner skaliert, bis die berechnete Schriftgröße unter einen bestimmten Schwellenwert, die minimale Schriftgröße, fällt. Ist dieser Schwellenwert erreicht, so werden die Listenelemente nicht mehr kleiner skaliert, sondern in der minimalen Schriftgröße angezeigt.

Die minimale Schriftgröße wird hierbei so berechnet, dass die größte Schriftgröße zur Darstellung von peripheren Listenelementen verwendet wird, bei der der Platz am Bildschirm bei der Darstellung aller Listenelemente völlig ausgefüllt wird. Falls die Anzahl der Listenelemente so groß ist, dass nicht alle Listenelemente gleichzeitig am Bildschirm dargestellt werden können, so wird automatisch versucht, die Fokuslänge oder die maximale Schriftgröße solange zu reduzieren, bis die Liste vollständig am Bildschirm Platz findet.

Die Berechnung der Schriftgröße von Elementen erfolgt in ganzen Zahlen, weil Text nur in ganzzahligen Größen am Bildschirm dargestellt werden kann. Deshalb kann es bei Verwendung des beschriebenen Algorithmus aufgrund von Rundungsfehlern dazu kommen, dass nicht der gesamte zur Verfügung stehende Platz am Bildschirm zur Listendarstellung genutzt wird. Um diesem Problem entgegenzuwirken wird nach der Berechnung sämtlicher Schriftgrößen überprüft, ob freier Platz übrig bleibt. Ist dies der Fall, so wird dieser ungenutzte Platz dazu verwendet, den Fokusbereich zu vergrößern.

## 5.3 Implementierung

Die Implementierung der Fisheye View Liste für die geplanten Usability Tests erfolgte in Java auf Basis des Swing GUI Toolkits. Java ist eine etablierte und weitverbreitete Programmiersprache und verspricht gute Portabilität

und Kompatibilität mit einer Vielzahl von Betriebssystemen. Das Swing GUI Toolkit ist (gemeinsam mit AWT und Java 2D) Teil der Java Foundation Classes und erlaubt die Entwicklung von konsistenten User Interfaces für Java Programme unter Windows, Mac OS X und Linux. Swing bietet umfangreichere und bequemere GUI-Komponenten als AWT. Im Gegensatz zu AWT sind Swing Widgets vollständig in Java implementiert und damit plattformunabhängig, während AWT an das Windowing System des zugrundeliegenden Betriebssystems gebunden ist. Dadurch ist ein einheitliches Verhalten von Swing Widgets auf allen Plattformen gewährleistet. Für die Implementierung der Fisheye View Liste wurden zwei unterschiedliche Implementierungsstrategien erprobt und es wurden zwei Fisheye View Listen unter Verwendung beider Implementierungsstrategien entwickelt.

Bei der ersten Implementierungsstrategie wurde eine Fisheye View Liste direkt von der Klasse JComponent abgeleitet. Dementsprechend ist die entwickelte Komponente vollständig selbst dafür verantwortlich sich am Bildschirm zu zeichnen und sämtliche Utilityfunktionen (z.B. Funktionen zum Hinzufügen und Verändern von Listenelementen oder zur Manipulation des Verhaltens und der Erscheinung der Liste) müssen ebenfalls von der Komponente selbst implementiert werden. Im Rahmen der Erfordernisse dieser Arbeit wurden deshalb nur die für die Umsetzung der Testumgebung minimal erforderlichen Funktionen der Fisheye View Liste implementiert. Daraus ergibt sich, dass die auf diese Weise implementierte Liste nicht als einfacher "drop-in" Ersatz für eine JList geeignet ist.

Bei der zweiten Implementierungsstrategie wurde eine Fisheye View Liste direkt von der Klasse JList abgeleitet. Dementsprechend bietet diese Implementierung der Fisheye View Liste sämtliche Standardfunktionen einer JList wie z.B. Funktionen zur Kontrolle des Auswahlverhaltens. Zur Verwaltung der Listendaten wird wie für JLists üblich ein ListModel verwendet. Diese Fisheye View Liste verwendet den üblichen Zeichenmechanismus für JLists, nämlich einen anwendungsspezifischen CellRenderer, bzw. in diesem konkreten Fall mehrere anwendungsspezifische CellRenderer. Um die dynamische Darstellung einer visuell verzerrten Fisheye View mittels CellRenderer um-

zusetzen wird bei dieser Implementierung ein eigener CellRenderer für jede mögliche Fokusposition auf Basis der zuvor berechneten Lookup-Tables erzeugt und bei einem Fokuswechsel wird der JList der entsprechende CellRenderer zugewiesen. Der Umstand, dass für jede mögliche Fokusposition ein eigener CellRenderer verwendet wird bedeutet, dass für jedes Listenelement ein eigener CellRenderer benötigt wird, wodurch bei langen Listen sehr viele CellRenderer Objekte benötigt werden. In der Praxis hat sich gezeigt, dass die Neuinstanzierung eines CellRenderer-Objekts bei jedem Fokuswechsel sehr ressourcenintensiv ist und die Performance und Reaktionsfähigkeit der Liste stark beeinträchtigt. Deshalb werden sämtliche CellRenderer für alle Fokuspositionen gecached, wobei folgende, sehr simple Caching Strategie verwendet wird: Nach der Instanzierung der Liste ist der CellRenderer Cache noch leer. Bei jedem Fokuswechsel wird nun überprüft, ob ein entsprechender CellRenderer für diese Fokusposition bereits erzeugt wurde. Ist dies nicht der Fall, so wird ein neuer CellRenderer für diese Fokusposition instanziert und im RendererCache abgelegt, ansonsten wird der bereits gecachete CellRenderer verwendet. Diese Caching-Strategie wurde gewählt, um zu verhindern, dass nach der Instanzierung der Liste sämtliche CellRenderer initialisiert werden müssen, was negative Auswirkungen auf die Initialisierungsdauer der Liste hätte. In der Praxis hat sich gezeigt, dass zumindest bei Listen mit nur einigen hundert Elementen die höheren Speicheranforderungen bei der Verwendung eines CellRenderer Cache weniger ins Gewicht fallen als die Nachteile der on-demand Instanzierung von CellRenderern.

Beide Implementierungsstrategien haben ihre Vor- und Nachteile, wie sich im Laufe der Weiterentwicklung und praktischen Verwendung zeigte. Der größte Vorteil der Fisheye View Liste auf Basis der JComponent ist die Geschwindigkeit. Diese Implementierung erwies sich im Vergleich zur JList-basierten Fisheye View Liste um ein Vielfaches schneller und auch erheblich weniger ressourcenhungrig. Dies ist darauf zurückzuführen, dass diese Komponente sich völlig selbstständig am Bildschirm zeichnet und man deshalb bei der Implementierung wesentlich besser auf die äußerst dynamische Natur der Darstellung eingehen konnte. Bei der Implementierung der Fisheye View

Liste auf Basis der JList wurde sehr schnell offensichtlich, dass diese Komponente und die zur Verfügung stehenden Mechanismen zur Anpassung ihrer Erscheinung in Form von anwendungsspezifischen CellRenderer-Objekten nicht für sehr dynamische Änderungen der Listendarstellung in Echtzeit in Abhängigkeit vom User-Input gedacht sind. In der Praxis zeigte sich, dass die JComponent-basierte Fisheye View Liste problemlos auf einem fünf Jahre alten Computer lief, während die JList-basierte Fisheye View Liste auf demselben System so langsam lief, dass sie praktisch nicht mehr benutzbar war. Es sei jedoch erwähnt, dass bei beiden Implementierungen zweifelsfrei noch erhebliche Verbesserungen der Performance möglich sind.

Ein weiterer Vorteil der JComponent-basierten Fisheye View Liste, der ebenfalls aus dem Umstand erwächst, dass diese Komponente sich selbst vollständig zeichnet, ist die Flexibilität und Anpassbarkeit der Darstellung. Die Implementierung der seitlichen alphabetischen Indizes wäre in der JList-basierten Fisheye View Liste unter Verwendung von anwendungsspezifischen CellRenderern nur sehr schwierig (wenn überhaupt) realisierbar gewesen. Ähnlich würde es sich beispielsweise mit der Implementierung eines Focus Lock Modus im Stil der Fisheye Menus (Bederson, 2000) verhalten.

Ein Nachteil unserer JComponent-basierten Implementierung ist wie bereits erwähnt, dass diese keinen einfachen “drop-in” Ersatz für JLists darstellt. Die entwickelte Fisheye View Liste auf Basis der JList hingegen lässt sich sehr einfach als Ersatz für eine JList in jedem Java Swing Interface verwenden und eine solche Ersetzung würde nur minimale Änderungen am bestehenden User Interface Code erforderlich machen.

Aufgrund der Erfordernisse dieser Arbeit wurde beschlossen, sich bei der Weiterentwicklung der Komponenten auf die JComponent-basierte Fisheye View Liste zu beschränken, weil diese von Beginn an eine wesentlich bessere Reaktionsfähigkeit und Benutzbarkeit versprach und weniger Probleme im Rahmen der weiteren Entwicklung absehbar waren. Dementsprechend wurden die alphabetischen Indizes auch nur in dieser Implementierung realisiert.

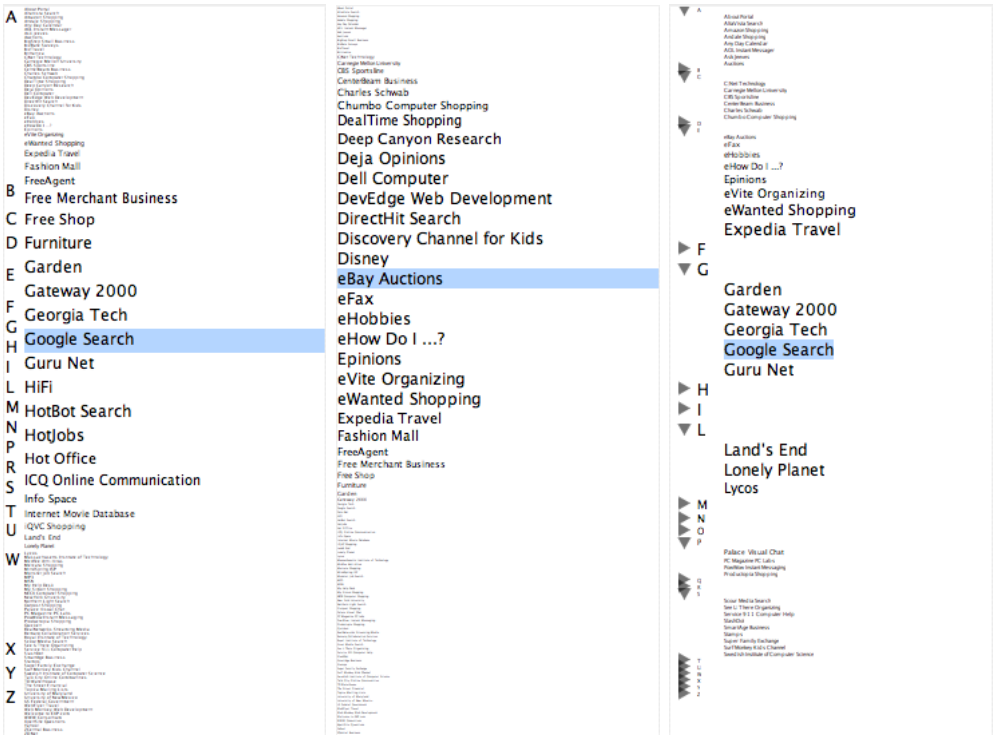


Abbildung 5.3: Die drei im Rahmen dieser Arbeit implementierten User Interface Widgets: Fisheye View Liste auf Basis einer JComponent, Fisheye View Liste auf Basis einer JList und Prototyp eines Fisheye View Tree

Neben den beiden Implementierungen einer Fisheye View Liste wurde auch noch ein Prototyp von einem Baum mit eindimensionaler visueller Fisheye Verzerrung entwickelt. Die Implementierung des Fisheye View Baums orientierte sich hierbei an der Implementierung der JList-basierten Fisheye View Liste. Der Fisheye View Baum wurde direkt von der Klasse JTree abgeleitet. Zur Darstellung der eindimensionalen visuellen Fisheye Verzerrung kam eine Reihe von anwendungsspezifischen TreeCellRenderer-Objekten zum Einsatz. Aufgrund der bereits erwähnten Einschränkungen und Nachteile dieser Implementierungsstrategie wurde die Weiterentwicklung dieses Prototyps im Rahmen dieser Arbeit jedoch nicht weiter verfolgt.

# Kapitel 6

# Methode

Im Rahmen der Aufgabenstellung wurde die Usability von Fisheye View Listen zum Auswählen, Ablegen und Sortieren von Datenelementen anhand von Usability Tests evaluiert. Zur Evaluierung wurden Vergleichstests zwischen der im Rahmen dieser Arbeit implementierten Fisheye View Liste und bestehenden, etablierten User Interface Elementen durchgeführt. Ziel der Usability Tests ist es, die Usability von Fisheye View Listen hinsichtlich Effektivität, Effizienz und Zufriedenheit der Anwender zu evaluieren und mit etablierten User Interface-Elementen wie Listen mit Scrollbalken und Baumdarstellungen zu vergleichen. Insgesamt 10 Testpersonen, ausnahmlos langjährige Computeranwender, nahmen an den Usability Tests teil.

Die zu testenden Interfaces orientieren sich in Layout und Aussehen an bestehenden, verbreiteten Desktop Applikationen wie z.B. Mail Clients oder Media Playern, um eine gewisse Vertrautheit in den Testpersonen zu wecken und die Anwendbarkeit der Ergebnisse auf bereits existierende Applikationen zu gewährleisten. Für die Durchführung der Tests wurde ein eigenständiges Testprogramm entwickelt, welches die zu testenden Interfaces implementiert und umfassende, automatische Logging-Mechanismen bietet. Neben der Erhebung von quantitativen Daten im Rahmen des automatisierten Loggings sollen auch qualitative Daten bezüglich Akzeptanz und Zufriedenheit der

Testpersonen erhoben werden. Zur Erhebung dieser Daten kam ein Post-Test-Fragebogen zum Einsatz.

## 6.1 Testziele

Ziel des Usability Tests ist es, die Usability von Fisheye View Listen hinsichtlich Effektivität, Effizienz und Zufriedenheit der Anwender in Tests mit realen Anwendern zu evaluieren und die Akzeptanz der Testpersonen für diesen unkonventionellen Interfacetyp zu ermitteln. Es soll festgestellt werden, inwiefern sich die Benutzbarkeit von Fisheye View Listen von etablierten User Interface Elementen wie Listen mit Scrollbalken und Baumdarstellungen unterscheidet. Weiters sollte festgestellt werden, ob sich Unterschiede in der Eignung der drei getesten Interfaces für Tasks vom Typ "Selection" oder "Drag & Drop" feststellen lassen und welche Interfaces für welche Art von Task besonders gut oder schlecht geeignet sind.

Neben der Evaluierung von performancebezogenen Daten soll auch die Zufriedenheit und Akzeptanz der Testpersonen gegenüber der Fisheye View Liste analysiert werden. Zu diesem Zweck wurden sowohl qualitative anhand eines Fragebogens als auch quantitative Daten durch automatisches Logging im Rahmen der Tests erhoben.

Darüber hinaus soll auch festgestellt werden, inwiefern der für die Tests entwickelte Prototyp einer Fisheye View Liste für die Verwendung von realen Anwendern geeignet ist. Es soll festgestellt werden, welche Schwächen und Probleme der entwickelte Prototyp noch aufweist, um diese in Zukunft beheben zu können, und welche Vorteile er gegenüber traditionellen User Interface Elementen aufweist, um diese in der weiteren Entwicklung und zur Verbesserung des Prototyps berücksichtigen zu können.

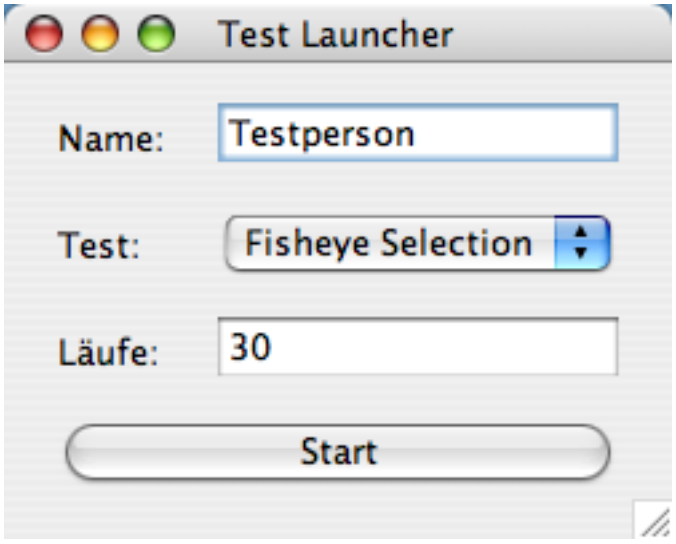


Abbildung 6.1: Testprogramm Launcher Screen

## 6.2 Testprogramm

Im Rahmen dieser Arbeit wurde auf Basis der in Kapitel 5 beschriebenen Fisheye View Liste ein Testprogramm zur Durchführung der Usability Tests entwickelt. Das Testprogramm wurde als Java Applikation entwickelt, um möglichst hohe Portabilität zu gewährleisten, und bedient sich des Swing GUI Toolkits. Das Testprogramm besteht aus einem einfachen Launcher-Fenster, in dem die grundlegende Konfiguration der Tests (Name der Testperson, Art des Tests, Anzahl der Testdurchläufe) vorgenommen werden kann (siehe Abbildung 6.1). Der Launcher bietet dabei folgende sechs Testarten zur Auswahl:

- Fisheye Selection
- List Selection
- Tree Selection
- Fisheye Drag & Drop

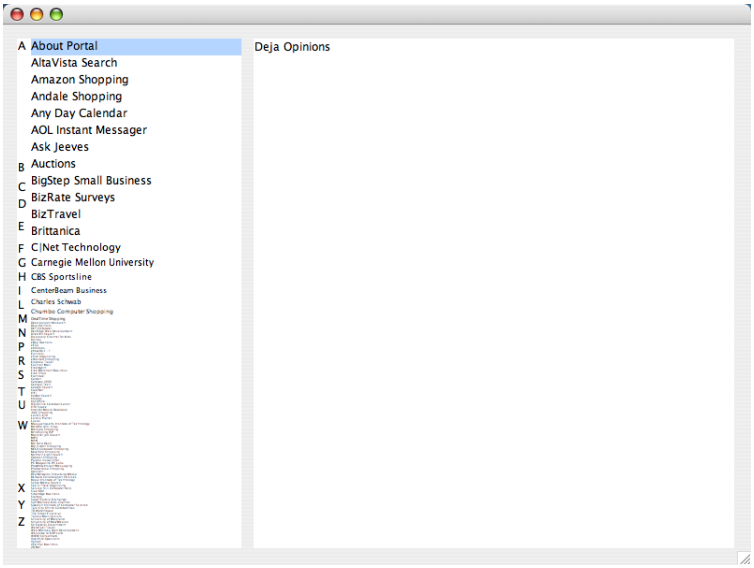


Abbildung 6.2: Testprogramm mit Fisheye View Liste

- List Drag & Drop
- Tree Drag & Drop,

Dies entspricht jeder möglichen Kombination aus Tasktyp und Interfacetyp.

Nach der Auswahl der Art des Tests kann der Test durch Mausklick auf den Start-Button gestartet werden. Dadurch wird ein neues Fenster geöffnet, welches je nach Testart ein GUI-Element vom Typ Fisheye View Liste, Liste oder Baum enthält. In diesem Fenster werden die eigentlichen Tests durchgeführt. Das Fenster ist 800 x 600 Pixel groß und der Aufbau des Fensters ist folgendermaßen (siehe Abbildung 6.2, Abbildung 6.3 und Abbildung 6.4): Links befindet sich wie erwähnt ein GUI-Element vom Typ Fisheye View Liste, Liste oder Baum, in dem eine eindimensionale Datenstruktur dargestellt wird. Fisheye View Liste und Liste mit Scrollbalken sind hierbei 240 x 545 Pixel groß, die Baumdarstellung ist 280 x 545 Pixel groß. Die Schriftgrößen in der Fisheye View Liste reichen von 12 Pixel für die Elemente im Fokusbereich bis hin zu 3 Pixel für Elemente außerhalb des Fokusbereichs. Bei der Darstellung als Baum werden die Elemente dieser eindimensionalen

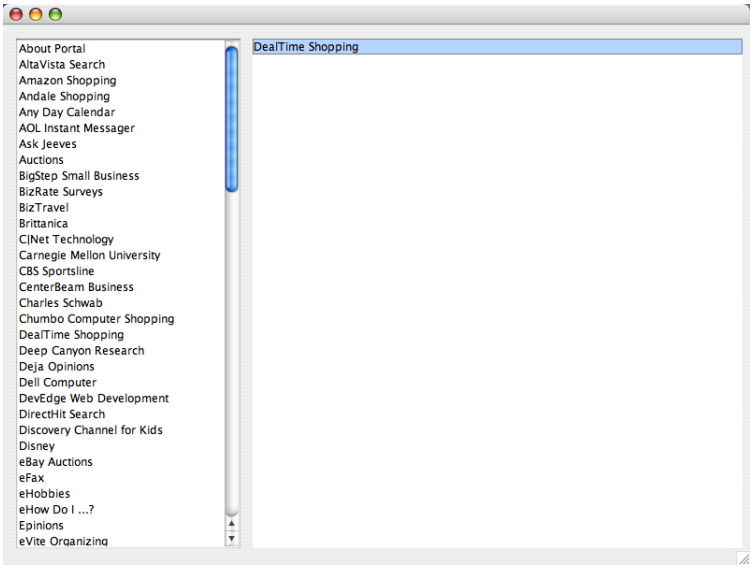


Abbildung 6.3: Testprogramm mit traditioneller Liste

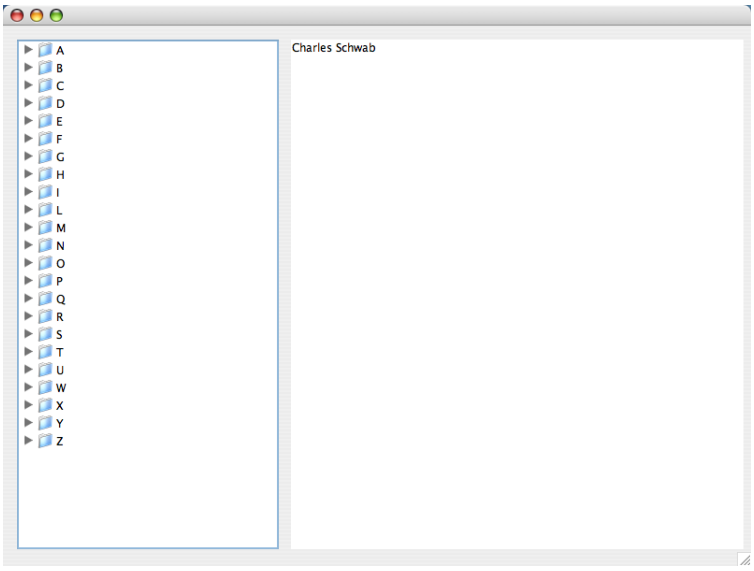


Abbildung 6.4: Testprogramm mit Baumdarstellung zur hierarchischen Gruppierung

Datenstruktur alphabetisch in Ordnern mit dem entsprechenden Buchstaben als Label gruppiert. Bei der Liste mit Scrollbalken handelt es sich um eine Swing JList und beim Tree handelt es sich um einen Swing JTree. Die Liste bietet Autoscroll-Funktionalität und der Tree bietet sowohl Autoscroll- als auch Autoexpand-Funktionalität. Mit Autoscroll ist gemeint, dass der Viewport des GUI-Elements automatisch nach oben bzw. unten scrollt, wenn sich der Benutzer mit gedrückter Maustaste (beispielsweise während eines Drag-Vorgangs) mit dem Mauszeiger dem oberen oder unteren Rand des GUI-Elements nähert. Autoexpand bezeichnet eine Funktion, bei der ein Ordner in der Baumdarstellung automatisch geöffnet wird, wenn der Benutzer mit gedrückter Maustaste für eine gewisse Zeit über diesem Ordner verharrt. Rechts davon befindet sich eine Liste, in der jeweils ein Element angezeigt wird, welches auch in der linkerhand dargestellten Datenstruktur enthalten ist. Diese Liste ist 514 x 545 Pixel (bzw. 484 x 545 Pixel im Fenster mit Baumdarstellung) groß. Der grundsätzliche Aufbau des Testprogramms ist typischen Desktop-Applikationen wie z.B. Mozilla Firefox oder Microsoft Outlook nachempfunden, um eine gewisse Vertrautheit mit dem Programmaufbau in den Testpersonen zu wecken und die Anwendbarkeit der Ergebnisse auf praktische Anwendungsszenarien zu gewährleisten.

Die Aufgabe der Testperson ist abhängig vom gewählten Tasktyp “Selection” oder “Drag & Drop”. Beim Tasktyp “Selection” ist es die Aufgabe der Testperson, das rechts dargestellte Element in der Darstellung der Datenstruktur auf der linken Seite mittels Mausklick auszuwählen. Beim Tasktyp “Drag & Drop” ist es die Aufgabe der Testperson, das auf der rechten Seite dargestellte Element auf das korrespondierende Element in der Darstellung der Datenstruktur auf der linken Seite mittels Drag & Drop zu ziehen und dort abzulegen.

Das Testprogramm verfügt über einen automatischen Logging-Mechanismus, welcher performancerelevante und nutzungsbedingte Charakteristika automatisch mitprotokolliert. Eine genaue Übersicht über sämtliche protokollierte Daten findet sich in Kapitel 6.6. Sämtliche Logdaten werden automatisch nach Testbedingung und Testperson in einer Textdatei als Comma Separated

Values gespeichert.

## 6.3 Testpersonen

Insgesamt zehn Testpersonen wurden für die Durchführung der Usability Tests aus dem Bekanntenkreis des Autors rekrutiert. Die Testpersonen erhielten für ihre Teilnahme an den Tests keinerlei monetäre oder anderweitige Aufwandsentschädigungen. Die Testpersonen waren zum Zeitpunkt des Tests zwischen 20 und 25 Jahren alt (Mittelwert 23,6, Standardabweichung 2,065). Sieben Testpersonen waren männlich und drei Testpersonen waren weiblich. 8 Testpersonen waren Rechtshänder und 2 Testpersonen waren Linkshänder. Sämtliche Testpersonen hatten als höchste abgeschlossene Ausbildung wenigstens die Matura und vier Testpersonen hatten einen Universitätsabschluss.

| Person | Alter | Geschlecht | Händigkeit | Ausbildung | Beruf |
|---|---|---|---|---|---|
| TP 1 | 24 | männlich | Rechtshänder | Matura | Student |
| TP 2 | 25 | männlich | Rechtshänder | Universität | Student |
| TP 3 | 21 | männlich | Linkshänder | Matura | Student |
| TP 4 | 21 | weiblich | Rechtshänder | Matura | IT Admin |
| TP 5 | 20 | weiblich | Rechtshänder | Matura | Student |
| TP 6 | 25 | männlich | Rechtshänder | Universität | Marketing |
| TP 7 | 25 | männlich | Rechtshänder | Matura | Programmierer |
| TP 8 | 25 | männlich | Rechtshänder | Matura | Student |
| TP 9 | 25 | weiblich | Linkshänder | Universität | Therapeut |
| TP 10 | 25 | männlich | Rechtshänder | Universität | Ökonom |

Tabelle 6.1: Daten des Backgroundfragebogens

Bei der Auswahl der Testpersonen wurde darauf geachtet, dass sämtliche Testpersonen über mehrere Jahre Erfahrung im Umgang mit Computern verfügten. Dies war unter anderem deshalb von zentraler Bedeutung, um gewährleisten zu können, dass sämtliche Testpersonen sehr vertraut im Umgang mit einer Maus und den damit üblichen Interaktionsmöglichkeiten wie z.B. Drag & Drop sind. Dies soll gewährleisten, dass die Ergebnisse nicht

durch mangelnde Vertrautheit im Umgang mit dem Eingabegerät bzw. kurzfristige Übungseffekte während der Durchführung der Tests verfälscht werden. Weiters soll dadurch möglichst ausgeschlossen werden, dass eine deutliche Verbesserung der gemessenen Performance durch umfangreichere Übung und Erfahrung im Umgang mit Computermäusen möglich wäre. Sämtliche Testpersonen hatten umfangreiche Erfahrung in der Verwendung von traditionellen Listen und Baumdarstellungen als Teil von Desktop Applikationen. Keine der Testpersonen hatte irgendwelche Erfahrungen in der Verwendung von Fisheye Views als Teil von graphischen User Interfaces.

Die Testpersonen wurden darum gebeten, einen Background- und Pre-Test-Fragebogen vor der Durchführung der Tests auszufüllen. Die durch diese Fragebögen erhobenen Daten sind in Tabelle 6.1, Tabelle 6.2 und Tabelle 6.3 im Detail ersichtlich.

| Frage | TP 1 | TP 2 | TP 3 | TP 4 | TP 5 |
|---|---|---|---|---|---|
| Erfahrung im Umgang mit PCs: Anfänger (1) - Experte (5) | 2 | 5 | 4 | 4 | 5 |
| Seit wann verwenden Sie PCs? | | | | | |
| weniger als 1 Jahr | | | | | |
| mehr als 1 Jahr | | | | | |
| mehr als 3 Jahre | X | | X | | X |
| mehr als 10 Jahre | | X | | X | |
| Wofür verwenden Sie PCs überwiegend? | | | | | |
| beruflich | | | | X | |
| privat | X | X | X | | X |
| anderweitig | | | | | |
| Wie häufig verwenden Sie PCs? | | | | | |
| 8 Stunden oder mehr am Tag | | X | | X | X |
| 4-8 Stunden am Tag | | | X | | |
| 2-4 Stunden am Tag | X | | | | |
| 1-2 Stunden am Tag | | | | | |
| Mehrmals pro Woche | | | | | |
| Mehrmals pro Monat | | | | | |
| Seltener | | | | | |
| Welches Betriebssystem verwenden Sie überwiegend? | | | | | |
| Windows | X | X | | X | |
| Mac OS | | | X | | X |
| Linux | | | | | |
| Sonstiges | | | | | |
| Welche Art von Zeigegerät verwenden Sie überwiegend? | | | | | |
| Mouse | X | X | X | X | X |
| Touchpad | | | | | |
| Sonstiges | | | | | |

Tabelle 6.2: Daten des Pre-Test-Fragebogens, Teil 1

| Frage | TP 6 | TP 7 | TP 8 | TP 9 | TP 10 |
|---|---|---|---|---|---|
| Erfahrung im Umgang mit PCs: Anfänger (1) - Experte (5) | 4 | 5 | 3 | 3 | 3 |
| Seit wann verwenden Sie PCs? | | | | | |
| weniger als 1 Jahr | | | | | |
| mehr als 1 Jahr | | | | | |
| mehr als 3 Jahre | | | | X | X |
| mehr als 10 Jahre | X | X | X | | |
| Wofür verwenden Sie PCs überwiegend? | | | | | |
| beruflich | X | | | | X |
| privat | | X | X | X | |
| anderweitig | | | | | |
| Wie häufig verwenden Sie PCs? | | | | | |
| 8 Stunden oder mehr am Tag | | X | | | |
| 4-8 Stunden am Tag | X | | X | | X |
| 2-4 Stunden am Tag | | | | | |
| 1-2 Stunden am Tag | | | | | |
| Mehrmals pro Woche | | | | X | |
| Mehrmals pro Monat | | | | | |
| Seltener | | | | | |
| Welches Betriebssystem verwenden Sie überwiegend? | | | | | |
| Windows | X | | X | X | X |
| Mac OS | | | | | |
| Linux | | X | | | |
| Sonstiges | | | | | |
| Welche Art von Zeigegerät verwenden Sie überwiegend? | | | | | |
| Mouse | X | X | X | X | X |
| Touchpad | | | | | |
| Sonstiges | | | | | |

Tabelle 6.3: Daten des Pre-Test-Fragebogens, Teil 2

## 6.4 Testszenarien

Bei der Gestaltung der Testumgebung und des Testaufbaus wurde versucht, sich möglichst an klassischen, etablierten Desktop-Applikationen zu orientieren. Der Aufbau des Testprogramms ist typischen Desktop-Applikationen wie z.B. Mozilla Firefox oder Microsoft Outlook nachempfunden. Typisch für solche Programme ist beispielsweise, dass die Darstellung einer Datenstruktur wie z.B. eine Liste von Ordnern zur Kategorisierung von Mails oder Bookmarks üblicherweise in einem kleinen Fensterbereich links erfolgt, während die Darstellung von Daten, mit welchen der Benutzer interagiert und welche sortiert werden können in einem größeren Fensterbereich auf der rechten Seite erfolgt. Das Layout des Testprogramms orientiert sich auch an diesem Aufbau. Dies hat den Zweck, eine gewisse Vertrautheit mit dem Testprogramm in den Testpersonen zu wecken und soll außerdem dazu beitragen, dass die Ergebnisse der Usability Tests und die gewonnen Erkenntnisse auch auf tatsächliche, praktische Anwendungsszenarien anwendbar sind. Mit Hilfe des Testprogramms wurden neben Fisheye View Listen auch traditionelle Listen mit Scrollbalken und hierarchische Baumdarstellungen evaluiert. Diese beiden Arten von GUI-Elementen wurden für den Vergleich mit Fisheye View Listen herangezogen, weil es sich hierbei um weit verbreitete, etablierte GUI-Elemente handelt, die in zahllosen Desktop-Applikationen in genau dieser Funktion zum Einsatz kommen.

Auch bei der Auswahl der von den Testpersonen zu erfüllenden Tasks wurde versucht, sich an typischen Information-Management-Tasks zu orientieren, welche aus der Benutzung von verbreiteten Desktop-Applikationen erwachsen. Es wurde beschlossen, sowohl Selection- als auch Drag & Drop Tasks mit Fisheye View Listen, traditionellen Listen mit Scrollbalken und hierarchischen Baumdarstellungen zu evaluieren weil es sich hierbei um typische Mausinteraktionen mit Daten in graphischen User Interfaces handelt. Sowohl Selection Tasks als auch Drag & Drop Tasks sind hierbei folgende Charakteristika gemein: Das zu testende GUI-Element (also Fisheye View Liste, Liste oder Baum) zeigt eine eindimensionale Datenstruktur, wobei bei

der Darstellung als Baumstruktur die Einträge der eindimensionalen Datenstruktur alphabetisch nach Anfangsbuchstaben in Ordnern gruppiert werden. Der Testperson wird ein Datenelement angezeigt, welches in jener Datenstruktur enthalten ist, die in jenem GUI-Element dargestellt wird, welches getestet werden soll. Bei einem Task vom Typ Selection Task ist es nun die Aufgabe der Testperson, das gezeigte Datenelement in der Datenstruktur zu finden und mittels Linksklick zu selektieren. Ein Selection Task gilt als erfolgreich abgeschlossen, wenn die Testperson das gezeigte Datenelement korrekt selektiert. Ein Selection Task gilt als fehlgeschlagen, wenn die Testperson ein falsches Datenelement aus der Datenstruktur mittels Mausklick selektiert. Bei einem Task vom Typ Drag & Drop Task ist es die Aufgabe der Testperson, das gezeigte Datenelement korrekt mittels Drag & Drop auf dem korrespondierenden Datenelement in der dargestellten Datenstruktur abzulegen. Ein Drag & Drop Task gilt als erfolgreich abgeschlossen, wenn die Testperson das gezeigte Datenelement korrekt ablegt. Ein Drag & Drop Task gilt als fehlgeschlagen, wenn die Testperson das gezeigte Datenelement auf einem falschen Datenelement in der dargestellten Datenstruktur ablegt. Diese Tasks ähneln dabei der typischen Interaktion mit verschiedenen Desktop-Programmen: Der Selection Task ist beispielsweise der Auswahl eines Bookmarks in einem Browser ähnlich, während der Drag & Drop Task beispielsweise der Vorgehensweise zur Kategorisierung von E-Mails in Ordnern in einem Mailclient oder dem Hinzufügen einer Website zu den eigenen Bookmarks ähnelt.

Die Tests bestehen nur aus diesen zwei Tasks, die in mehreren Testdurchläufen wiederholt werden. Nach jedem Testdurchlauf wird die Testumgebung in ihren Ausgangszustand zurückgesetzt. Das heißt der Viewport von Liste und Baumdarstellung wird zurückgesetzt und bei der Baumdarstellung werden sämtliche geöffneten Ordner wieder geschlossen. In Fisheye View Listen wird beim Verlassen des Mauszeigers der Fokus auf das erste Element zurückgesetzt. Dies geschieht, weil jeder Task für sich alleine und nicht als Teil eines größeren Arbeitspakets stehen soll und um sicherzustellen, dass jeder Task unter denselben Grundbedingungen durchgeführt wird. Diese Vor-

gehensweise birgt jedoch das Risiko, die Testpersonen anfänglich zu verwirren und um diesem Risiko entgegenzuwirken wurden die Testpersonen vor Beginn der Usability Tests im Rahmen der Einführung ausdrücklich auf dieses Verhalten aufmerksam gemacht. Darüber hinaus tragen die Probedurchläufe vor der Durchführung der eigentlichen Tests dazu bei, dieses Risiko der Verwirrung weiter zu minimieren, weil die Testpersonen zu Beginn der Tests bereits mit diesem Verhalten vertraut sind.

Weiters wird immer nur ein einziges Datenelement in der "Aufgabenliste" (also jener Liste, in der jene Datenelemente angezeigt werden, welche selektiert bzw. abgelegt werden sollen) angezeigt. Nach Beendigung eines Tasks, egal ob erfolgreich oder nicht, wird die Aufgabenliste mit einem neuen Datenelement zur Selektion bzw. Sortierung neu aufgebaut. Auch dies hat seine Ursache darin, dass jeder Task für sich alleine stehend und unter denselben Grundbedingungen durchgeführt werden soll, um die Nachvollziehbarkeit der Usability Tests zu gewährleisten. Würden mehrere Datenelemente gleichzeitig in der Aufgabenliste angezeigt werden, so würden die Tasks eher Teilaufgaben in einem großen Aufgabenpaket entsprechen. Dies wiederum birgt die Gefahr, dass die Ergebnisse z.B. durch die Anwendung von konkreten Strategien zur effizienten Erfüllung der Aufgaben verfälscht werden könnten. Weiters könnten die Testergebnisse dadurch beeinflusst werden, dass die Testpersonen sich erst einen Überblick über die dargestellten Datenelemente verschaffen müssten und durch die Auswahlmöglichkeiten gebremst werden könnten.

Die Datenelemente zur Selektion bzw. Sortierung werden vollkommen zufällig und mit möglicher Wiederholung aus der im Rahmen der Tests verwendeten eindimensionalen Datenstruktur ausgewählt. Als Datenstruktur wurde für die Durchführung der Tests eine Liste von 100 populären Websites gewählt, welche auch bereits im Rahmen der Tests von Bederson (2000) zum Einsatz kam. Daraus ergibt sich, dass in der Fisheye View Liste und herkömmlichen Liste 100 Datenelemente direkt angezeigt werden, während in der hierarchischen Baumdarstellung die 100 Datenelemente alphabetisch nach Anfangsbuchstaben in Ordnern gruppiert sind und sich somit eine Ord-

Abbildung 6.5: Testumgebung

nerstruktur auf 2 Ebenen mit maximal 26 Items auf oberster Hierarchieebene und insgesamt 100 Items auf zweiter Hierarchieebene ergibt.

Die Testpersonen wurden darauf hingewiesen, dass es bei den Tests um Schnelligkeit und Korrektheit der Taskerfüllung unter den unterschiedlichen Testbedingungen geht und wurden dementsprechend darum gebeten, die Tasks möglichst schnell und korrekt durchzuführen.

## 6.5 Testmaterial und Einrichtung

Für die Durchführung der Tests wurde auf eine ruhige Umgebung mit entspannter, freundlicher Atmosphäre geachtet, um äußere Einflüsse auf die Testpersonen bei der Durchführung der Tests zu minimieren (siehe Abbildung 6.5).

Die Tests wurden an einem Schreibtisch durchgeführt, auf dem das Testsystem aufgebaut war. Als Testsystem kam ein Apple MacBook mit 13”-TFT-Display zum Einsatz. Auf dem Testsystem waren Mac OS X 10.4, das Java 2 Runtime Environment 1.5 und das in Kapitel 6.2 beschriebene Testprogramm installiert. Es war kein externer Monitor an dem System angeschlossen, sämtliche Tests wurden auf dem integrierten 13” Bildschirm des Apple MacBooks mit einer Auflösung von 1280 x 800 Pixel durchgeführt. Als einziges Peripheriegerät war eine Maus als Zeigegerät an das Testsystem angeschlossen. Bei der Maus handelte es sich um eine Microsoft Basic Optical Mouse mit optischem Tracking System und Mousewheel. Die Maus hat eine symmetrische Form und ist laut Herstellerangaben sowohl für Links- als auch Rechtshänder geeignet. Die Maus war auf einem runden, schwarzen Mousepad mit textiler Oberfläche platziert. Zum Zeitpunkt des Tests waren keine anderen Fenster außer dem Testprogramm auf dem Monitor des Testsystems angezeigt.

## 6.6 Testdesign und Messungen

Die Usability Tests wurden als Within-Subject Tests konzipiert. Das bedeutet, dass jede Testperson den Test in jeder Testbedingung durchführt. Dieses Testdesign hat den möglichen Nachteil, dass einerseits Lern- und Gewöhnungseffekte, andererseits aber auch Ermüdungseffekte von Test zu Test auftreten und das Ergebnis beeinflussen können. Um das Risiko ungewollter Lern- und Ermüdungseffekte zu minimieren wurde die Reihenfolge der unterschiedlichen Testbedingungen von Testperson zu Testperson wie in Tabelle 6.4 aufgelistet gegenbalanciert. Darüber hinaus wurden vor dem Beginn der eigentlichen Usability Tests kurze Probeläufe in allen Testbedinungen mit allen Testpersonen durchgeführt, um die Testpersonen mit der Testumgebung, dem Testprogramm und der grundsätzlichen Aufgabenstellung vertraut zu machen. Auch die Reihenfolge der Probeläufe wurde Tabelle 6.4 entsprechend variiert.

| Person | 1. Testbedingung | 2. Testbedingung | 3. Testbedingung |
|---|---|---|---|
| TP 1 | List | Tree | Fisheye |
| TP 2 | List | Tree | Fisheye |
| TP 3 | List | Tree | Fisheye |
| TP 4 | Tree | Fisheye | List |
| TP 5 | Fisheye | List | Tree |
| TP 6 | Tree | Fisheye | List |
| TP 7 | Tree | Fisheye | List |
| TP 8 | Fisheye | List | Tree |
| TP 9 | Fisheye | List | Tree |
| TP 10 | Fisheye | List | Tree |

Tabelle 6.4: Counterbalancing der Reihenfolge der Testbedingungen

Die unabhängigen Variablen bei den Usability Tests sind Tasktyp und Interfacetyp. Die Tests waren in sechs Blöcken mit je einem Block pro Testbedingung (Interfacetyp Fisheye View Liste, Liste mit Scrollbalken und Baumansicht jeweils mit Tasktyp Selection und Drag & Drop) organisiert. Jeder Block bestand lediglich aus einer einzigen Art von Task (Selection oder Drag & Drop), welcher von den Testpersonen wiederholt durchgeführt wurde. Die Anzahl der Testdurchläufe pro Block wurde in Anlehnung an (Kaptelinin, 1995) auf 10 Probedurchläufe zur Einführung und Gewöhnung der Testpersonen und 30 eigentliche Testdurchläufe festgelegt. Die Probedurchläufe wurden jeweils als eigenständige, abgeschlossene Testeinheit vor der Durchführung der eigentlichen Usability Tests durchgeführt und hatten den Zweck, die Testteilnehmer mit dem Testprogramm und den zu erfüllenden Tasks vertraut zu machen.

In ersten Versuchen hat sich gezeigt, dass ein vollständiger Test von einer Testbedingung mit 10 Probeläufen und 30 echten Testdurchläufen von geübten Anwendern in unter 5 Minuten zu bewältigen ist. Daraus ergibt sich, dass ein Test mit allen sechs Testbedingungen (Fisheye View Liste, Liste und Tree jeweils mit Selection und Drag & Drop Task) in 30 Minuten durchgeführt werden kann. Bedenkt man die zusätzlichen Zeiterfordernisse von Begrüßung, Einführung, Einarbeitung in die Testumgebung, Ausfüllen der Background-, Pre- und Post-Test-Fragebögen, Debriefing und Verabschie-

dung, so muss für die Durchführung eines kompletten Tests insgesamt ca. 90 Minuten pro Testperson veranschlagt werden. Obwohl eine Erhöhung der Anzahl der Testdurchläufe aufgrund der relativ kurzen Testdauer von 5 Minuten pro Testbedingung denkbar wäre, ergaben Selbstversuche und auch der Pilottest, dass 10 Probeläufe und 30 echte Testdurchläufe an die Grenzen der Belastbarkeit der Testpersonen gehen.

Die Testpersonen erhielten während der Durchführung der Tests keinerlei Feedback über Korrektheit und Performance ihrer erbrachten Leistung. Als Testmethode kam die Performance Measurement Methode zur Erhebung von leistungsbezogenen, quantitativen Daten zum Einsatz. Sowohl Probedurchläufe als auch Testdurchläufe wurden automatisch mitprotokolliert, wobei die Daten der Probedurchläufe nicht zur weiteren Auswertung gedacht sind. Wie bereits in Kapitel 6.2 verfügt das für die Durchführung der Tests entwickelte Testprogramm über einen automatischen Loggingmechanismus. Folgende Daten werden als abhängige Variablen automatisch protokolliert:

- Korrektheit: Die Korrektheit des Selection- bzw. Drag & Drop-Vorgangs
- Task-Zeit: Die Zeit von Beginn des Tasks bis zur Erfüllung des Tasks
- Source-Item: Jenes Element, welches der Testperson in der Aufgabenliste rechts angezeigt wird
- Target-Item: Jenes Element, welches die Testperon (in Abhängigkeit vom Tasktyp) selektiert, bzw. auf welchem der Drag & Drop Vorgang beendet wird
- Last Scroll Time: Zeitpunkt, zu dem die Testperson zuletzt gescrollt hat (nur bei Liste und Tree)
- Last Expansion Time: Zeitpunkt, zu dem die Testperson zuletzt einen Ordner geöffnet hat (nur bei Tree)

Darüber hinaus werden folgende Daten für Tasks vom Typ “Drag & Drop” protokolliert:

- Drag-Zeit: Die Zeit, die der eigentliche Drag & Drop Vorgang (vom "Aufheben" des Ursprungselements bis zum "Fallenlassen" am Zielelement) beansprucht
- Autoscroll: Gibt an, ob die Testperson die Autoscroll-Funktion zur Erfüllung des Tasks verwendet hat (nur bei Liste und Tree)
- Autoexpand: Gibt an, ob die Testperson die Autoexpand-Funktion zur Erfüllung des Tasks verwendet hat (nur bei Trees)

Wie bereits in Kapitel 6.4 beschrieben gelten für die Tasks folgende Taskerfüllungskriterien: Ein Selection Task gilt als erfolgreich durchgeführt, wenn die Testperson das in der Aufgabenliste angezeigte Datenelement in der Repräsentation der Datenstruktur ausfindig macht und mittels linkem Mausklick selektiert. Ein Selection Task gilt als fehlgeschlagen, wenn die Testperson ein falsches Element in der Datenstruktur mittels Mausklick selektiert. Ein Drag & Drop Task gilt als erfolgreich durchgeführt, wenn die Testperson das in der Aufgabenliste angezeigte Datenelement mittels Drag & Drop aufhebt, auf das korrespondierende Datenelement in der Repräsentation der Datenstruktur zieht und den Drag & Drop Vorgang dort beendet. Ein Drag & Drop Task gilt als fehlgeschlagen, wenn die Testperson das Datenelement aus der Aufgabenliste mittels Drag & Drop auf einem falschen Datenelement in der Datenstruktur ablegt.

Beide Tasks können nur entweder vollständig erfolgreich sein oder vollständig fehlschlagen, es gibt keine teilweise Erfüllung oder Korrektheit. Dementsprechend wird der Taskerfüllungsgrad vom Testprogramm einfach als bool'scher Wert "true" oder "false" protokolliert. Die Messung sämtlicher Zeiten im Testprotokoll erfolgt in Millisekunden. Die Zeitmessung der Task-Zeit beginnt mit der Anzeige eines neuen Elements in der Aufgabenliste und endet, sobald der Task korrekt oder inkorrekt beendet wurde. Im Rahmen der Auswertung der Tests hat sich gezeigt, dass aufgrund des verwendeten Logging-Mechanismus Messfehler in der Größenordnung von einer Millisekunde auftreten konnten. Diese Fehler sind jedoch ausreichend gering, um

auf die Gültigkeit und Aussagekraft der gemessenen Zeiten keinen nennenswerten Einfluss zu haben.

Zur Erhebung qualitativer Daten hinsichtlich Zufriedenheit und Akzeptanz der Testpersonen wurden die Testteilnehmer nach der Durchführung der Tests gebeten, einen Post-Test-Fragebogen auszufüllen.

Das Design des Experiments sah Tasktyp und Interfacetyp als unabhängige Variablen und Taskerfüllungszeit und Taskkorrektheit als abhängige Variablen vor. Der Effekt von Interfacetyp auf die gemessenen Performancewerte wurde mittels Repeated Measures ANOVA getestet. Weitere paarweise Vergleiche wurden mittels Paired Samples T-Test durchgeführt. Die Präferenzdaten der Testpersonen wurden mittels Wilcoxon's Signed Ranks Test analysiert.

### 6.6.1 Ablauf

An den Usability Tests nahmen jeweils eine Testperson und ein Testleiter teil. Aufgabe des Testleiters war es, die Testumgebung vor der Ankunft der Testpersonen vorzubereiten, die Testpersonen einzuweisen, die Fragebögen zu verteilen und den Ablauf sowie die Aufgaben zu erklären. Die Tests wurden vom Testleiter überwacht und der Testleiter machte Notizen über besondere Vorkommnisse und Kommentare der Testpersonen während der Durchführung der Tests.

Der genaue Ablauf der Usability Tests gestaltete sich folgendermaßen (siehe auch Tabelle 6.5): Nach der Ankunft der Teilnehmer am Testort und einer Begrüßung wurden die Testpersonen zur Testumgebung geführt. Vor Beginn der Tests wurde den Testpersonen der Zweck des Tests erklärt. Danach wurden die Testpersonen gebeten einen Background-Fragebogen zur Erhebung personenbezogener Daten und einen Pre-Test-Fragebogen zur Erhebung testrelevanter Charakteristika (wie z.B. Erfahrung im Umgang mit Computern) auszufüllen. In weiterer Folge wurde den Testpersonen das Testprogramm und die zu erfüllenden Aufgaben erklärt.

| Zeit | Aktivität |
|---|---|
| 5 min | Begrüßung |
| 5 min | Einleitung und Orientierung |
| 10 min | Ausfüllen der Background- und Pre-Test-Fragebögen |
| 5 min | Erklärung des Testprogramms und der Aufgaben |
| 10 - 15 min | Durchführung der Probeläufe |
| 20 - 30 min | Durchführung der Usability Tests |
| 10 - 15 min | Ausfüllen des Post-Test-Fragebogens |
| 5 min | Bedanken und Verabschiedung |

Tabelle 6.5: Ablauf der Usability Tests

Danach wurde mit der Durchführung der Tests begonnen. Zuerst wurden die Testpersonen gebeten, Probeläufe bestehend aus 10 Task-Wiederholungen je Testbedingung zu absolvieren. Die Reihenfolge der Probeläufe richtete sich hierbei bereits nach der vom Counterbalancing Plan vorgegebenen Reihenfolge. Die Testpersonen wurden darauf aufmerksam gemacht, dass es hierbei noch nicht auf Leistung ankommt und dass sie sich zur Eingewöhnung in die Testumgebung ruhig Zeit lassen und Fragen an den Testleiter stellen können.

Nachdem sämtliche Probeläufe absolviert waren, wurde mit den eigentlichen Usability Tests begonnen. Die Testpersonen wurden zu diesem Zeitpunkt noch einmal ausdrücklich darauf hingewiesen, dass für die folgenden Tests Taskerfüllungszeit und Korrektheit von Bedeutung sind und dass sie sich bitte bemühen sollten, möglichst schnell und fehlerfrei die Tasks zu absolvieren. Die Reihenfolge der Testblöcke ergab sich aus dem vorbereiteten Counterbalancing Plan und entsprach dementsprechend der Reihenfolge der Probeläufe.

Nach Durchführung der Usability Tests wurden die Testpersonen gebeten, den Post-Test-Fragebogen zur Erhebung ihrer persönlichen Präferenzen und subjektiven Eindrücke auszufüllen. Danach wurde den Testpersonen für ihre Teilnahme an den Tests gedankt und man verabschiedete sich.

### 6.6.2 Anweisungen an die Testpersonen

Die Testpersonen erhielten vor der Durchführung der Usability Tests folgende Informationen und Anweisungen: Nach Ankunft und Begrüßung wurde den Testpersonen zuerst der Zweck der Tests und deren ungefährer Ablauf erklärt. Weiters wurde den Teilnehmern die grundsätzliche Idee hinter Fisheye Views kurz vorgestellt. Danach wurden die Testpersonen gebeten, Background- und Pre-Test-Fragebogen auszufüllen. Die Testpersonen wurden ausdrücklich darauf hingewiesen, dass nicht sie getestet werden, sondern die Qualität der unterschiedlichen Interfaces.

Danach wurde den Testpersonen die Testumgebung und das Testprogramm kurz vom Testleiter präsentiert und der grundsätzliche Aufbau der Testinterfaces erklärt. Anschließend wurden den Testpersonen die Tasks erklärt und kurz vom Testleiter praktisch demonstriert. Die Teilnehmer wurden informiert, dass vor der Durchführung der Tests Probeläufe absolviert werden sollen und dass es bei diesen Probeläufen noch nicht auf Leistung ankommt. Die Testpersonen wurden angehalten, sich bei offenen Fragen oder Unklarheiten während der Probeläufe an den Testleiter zu wenden. Nach Beendigung der Probeläufe wurden die Testpersonen darauf hingewiesen, dass nun die eigentlichen Tests folgen würden und dass bei diesen Tests Taskerfüllungszeit und Korrektheit von Bedeutung seien. Die Teilnehmer wurden dementsprechend gebeten, möglichst schnell und fehlerfrei zu arbeiten. Die Testpersonen wurden weiters darauf hingewiesen, dass die Testläufe automatisch mitprotokolliert würden.

Nach Durchführung der Tests wurden die Teilnehmer gebeten einen Post-Test-Fragebogen zur Erhebung qualitativer Daten auszufüllen. Es stand den Teilnehmern frei, sich bei Fragen zum Fragebogen an den Testleiter zu wenden.

## 6.7 Pilottest

Vor der Durchführung der Usability Tests mit den Testpersonen wurde ein Pilottest mit einem freiwilligen Teilnehmer durchgeführt. Ziel dieses Pilottests war es, mögliche Probleme im Testdesign, Testaufbau bzw. mit dem Testprogramm zu identifizieren und auf mögliche Lücken in der Vorbereitung der Tests aufmerksam zu werden.

Der Ablauf des Pilottests entsprach grundsätzlich dem Ablauf der geplanten Usability Tests wie in Kapitel 6.6.1 beschrieben. Im Rahmen des Pilottests wurden jedoch nicht nur drei unterschiedliche Interfaces evaluiert, sondern auch ein Viertes: Eine Fisheye View Liste auf Basis einer JComponent, eine Fisheye View Liste auf Basis einer JList, eine traditionelle Liste mit Scrollbalken und eine Baumdarstellung. Für eine Beschreibung der Unterschiede zwischen der Implementierung einer Fisheye View Liste auf Basis einer JComponent und auf Basis einer JList siehe Kapitel 5.3. Die Evaluierung der beiden Fisheye View Listen im Pilottest hatte den Zweck herauszufinden, welche der beiden Implementierungen besser geeignet für die Durchführung der Usability Tests ist.

Der Pilottest ergab, dass die Fisheye View Liste auf Basis der JComponent besser gelungen und angenehmer zu bedienen ist. Die Testperson gab an, dass diese Implementierung spürbar performanter und reaktionsschneller war und eine flüssigere Darstellung und Interaktion ermöglicht. Weiters gab die Testperson an, dass die Anzeige der alphabetischen Indizes links von der Liste eine wichtige Orientierungshilfe bei der Benutzung der Fisheye View Liste darstellt. Weiters zeigte sich im Rahmen des Pilottests sehr klar, dass mehr als 30 Wiederholungen der Tasks pro Testbedingung insbesondere in Anbetracht der recht hohen Anzahl von Testbedingungen unzumutbar für die Testpersonen sind.

Auf Basis der Erkenntnisse aus dem Pilottest wurde beschlossen, die Usability Tests mit der JComponent-basierten Implementierung einer Fisheye View Liste durchzuführen. Das Testprogramm und die darin implementierte

Logging-Funktionalität konnte sich im Rahmen des Pilottests bewähren und bedurfte keiner weiteren Änderungen. Die Anzahl der Taskwiederholungen wurde nicht weiter erhöht und bei 30 belassen. Es zeigten sich keine Probleme im Aufbau und der Formulierung der Fragebögen, diese wurden deshalb nicht weiter verändert.

Neben dem Pilottest wurde im Vorfeld der Usability Tests ein Experten-Selbsttest vom Autor über einen längeren Zeitraum durchgeführt, um Lerneffekte in der Bedienung der Testprogramme über mehrere Sessions hinweg ausschließen zu können. Über einen Zeitraum von zwei Wochen wurden die Testaufgaben vom Autor im Abstand von zwei Tagen, somit insgesamt sieben mal, wiederholt durchgeführt. Die Analyse der Daten lässt auf keine Lerneffekte im Rahmen der laufenden Verwendung der Testprogramme schließen. Deshalb wurden Lerneffekte im weiteren Verlauf des Experiments nicht berücksichtigt.

# Kapitel 7

# Ergebnisse

Jede der zehn Testpersonen führte 30 Taskwiederholungen von zwei unterschiedlichen Tasks mit den drei getesteten Interfacevarianten Fisheye View Liste, Liste mit Scrollbalken und Baumdarstellung durch. Daraus ergaben sich 1800 Messdatensätze aus den Usability Tests zur weiteren Auswertung. Nachdem Unterschiede zwischen Selection Tasks und Drag & Drop Tasks zu erwarten waren wurde beschlossen, die Ergebnisse für die beiden Tasktypen getrennt zu betrachten um taskspezifische Unterschiede der drei evaluierten Interfacevarianten zu finden.

## 7.1 Taskerfüllungszeit

Der Mittelwert der Taskerfüllungszeiten wurde für jede Testperson je Tasktyp und Interfacetyp berechnet. Das Ergebnis für Selection Tasks je Testperson ist in Tabelle 7.1 und das Ergebnis für Drag & Drop Tasks je Testperson ist in Tabelle 7.2 ersichtlich. Bei der Berechnung des Mittelwerts der Taskerfüllungszeiten wurden nur jene Datensätze miteinbezogen, bei denen der Task korrekt erfüllt wurde, weil die Taskerfüllungszeit für inkorrekt gelöste Tasks nicht aussagekräftig ist. Aus diesem Grund wurden 19 von den 1800

Messdatensätzen in der Auswertung der Taskerfüllungszeit ausgefiltert. Für eine Beschreibung der Korrektheitskriterien hinsichtlich Taskerfüllung siehe Kapitel 6.6.

Bei den mittleren Taskerfüllungszeiten für Selection Tasks zeigte sich die Baumdarstellung überlegen: Neun von zehn Testpersonen konnten die Selection Tasks unter Verwendung der Baumdarstellung am schnellsten durchführen, nur eine Testperson war unter Verwendung der Fisheye View Liste am schnellsten. Drei Testpersonen waren unter Verwendung der Fisheye View Liste am langsamsten, sechs Testpersonen waren unter Verwendung der Liste am langsamsten und eine Testperson war unter Verwendung der Baumdarstellung am langsamsten.

| | Fisheye View Liste | Liste | Baumdarstellung |
|---|---|---|---|
| TP 1 | 4,95 (1,54) | 4,46 (1,83) | **3,89** (1,02) |
| TP 2 | **3,56** (1,58) | 3,67 (1,48) | 3,77 (0,86) |
| TP 3 | 4,32 (1,82) | 4,70 (1,56) | **3,68** (0,63) |
| TP 4 | 4,34 (1,30) | 4,37 (1,95) | **4,20** (0,92) |
| TP 5 | 3,04 (1,02) | 2,80 (1,00) | **2,74** (0,53) |
| TP 6 | 3,32 (1,16) | 3,42 (1,04) | **3,20** (0,43) |
| TP 7 | 3,44 (1,05) | 4,34 (1,39) | **3,26** (0,60) |
| TP 8 | 3,91 (1,13) | 4,16 (1,73) | **3,35** (0,89) |
| TP 9 | 5,31 (1,84) | 4,60 (1,67) | **3,65** (0,93) |
| TP 10 | 4,28 (1,46) | 4,97 (2,14) | **3,93** (0,92) |

Tabelle 7.1: Mittelwert der Taskerfüllungszeiten der Usability Tests für Selection Tasks pro Testperson in Sekunden. Standardabweichung in Klammern. Die schnellsten Zeiten sind jeweils fett hervorgehoben.

Bei den mittleren Taskerfüllungszeiten für Drag & Drop Tasks konnten sechs von zehn Testpersonen die Tasks unter Verwendung der Baumdarstellung am schnellsten durchführen und jeweils zwei Testpersonen waren unter Verwendung der Fisheye View Liste bzw. unter Verwendung der Liste am schnellsten. Zwei Testpersonen waren unter Verwendung der Fisheye View Liste am langsamsten und acht Testpersonen waren unter Verwendung der Liste am langsamsten. Keine Testperson war unter Verwendung der Baumdarstellung am langsamsten.

| | Fisheye View Liste | Liste | Baumdarstellung |
|---|---|---|---|
| TP 1 | **5,30** (2,19) | 5,94 (1,88) | 5,40 (1,04) |
| TP 2 | 4,35 (1,76) | **4,18** (1,08) | 4,25 (0,56) |
| TP 3 | 4,79 (1,44) | 4,85 (1,56) | **4,44** (0,76) |
| TP 4 | **5,49** (1,84) | 6,48 (1,98) | 5,66 (1,15) |
| TP 5 | 3,63 (0,95) | **3,04** (1,03) | 3,28 (0,59) |
| TP 6 | 6,23 (2,22) | 6,44 (1,87) | **5,12** (1,04) |
| TP 7 | 4,99 (1,95) | 5,66 (2,30) | **4,87** (0,86) |
| TP 8 | 4,39 (1,70) | 5,87 (1,96) | **4,34** (0,60) |
| TP 9 | 5,45 (1,78) | 6,39 (2,19) | **4,75** (0,91) |
| TP 10 | 5,56 (1,81) | 7,33 (4,16) | **4,47** (0,75) |

Tabelle 7.2: Mittelwert der Taskerfüllungszeiten der Usability Tests für Drag & Drop Tasks pro Testperson in Sekunden. Standardabweichung in Klammern. Die schnellsten Zeiten sind jeweils fett hervorgehoben.

Auf Basis der mittleren Taskerfüllungszeiten je Testperson wurden die Mittelwerte der Taskerfüllungszeiten je Interfacevariante und Tasktyp berechnet. Diese sind in Tabelle 7.3 und in den Abbildungen 7.1 und 7.2 dargestellt. Sowohl bei Selection Tasks, als auch bei Drag & Drop Tasks erwies sich die Baumdarstellung als am schnellsten in der Bedienung, gefolgt von der Fisheye View Liste. Die traditionelle Liste erwies sich in beiden Fällen als am langsamsten.

Die Performanceunterschiede zwischen den Interfaces wurden mittels Repeated Measures ANOVA über die mittleren Taskerfüllungszeiten der einzelnen Testpersonen analysiert. Für Selection Tasks war der Effekt von Interfacetyp auf die Taskzeiten signifikant ($F(2,18) = 7{,}84$, $p < 0{,}01$). Die paarweise Analyse der mittleren Taskerfüllungszeiten bei Selection Tasks mittels Paired Samples T Test ergab einen signifikanten Unterschied zwischen Fisheye View Liste ($M = 4{,}05$, $SD = 0{,}73$) und Baumdarstellung ($M = 3{,}57$, $SD = 0{,}43$) ($t(9) = 2{,}84$, $p < 0{,}05$) und zwischen Liste ($M = 4{,}15$, $SD = 0{,}66$) und Baumdarstellung ($t(9) = 4{,}04$, $p < 0{,}01$). Zwischen Fisheye View Liste und herkömmlicher Liste konnte kein signifikanter Unterschied bei den Taskerfüllungszeiten festgestellt werden. Daraus lässt sich schlussfolgern, dass die Baumansicht signifikant schneller als Fisheye View Liste und Liste war.

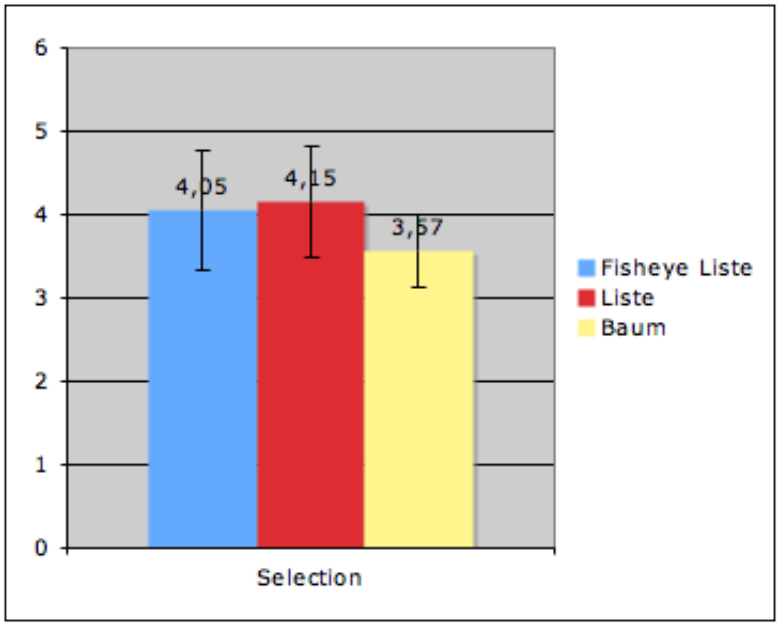


Abbildung 7.1: Mittelwert der Taskerfüllungszeiten der Usability Tests für Selection Tasks in Sekunden.

Für Drag & Drop Tasks war der Effekt von Interfacetyp auf die Taskzeiten ebenfalls signifikant ($F(2,18) = 8{,}89$, $p < 0{,}01$). Die paarweise Analyse der mittleren Taskerfüllungszeiten bei Drag & Drop Tasks ergab einen signifikanten Unterschied zwischen Liste ($M = 5{,}62$, $SD = 1{,}27$) und Fisheye View Liste ($M = 5{,}02$, $SD = 0{,}75$) ($t(9) = 2{,}58$, $p < 0{,}05$), zwischen Fisheye View Liste und Baumdarstellung ($M = 4{,}66$, $SD = 0{,}64$) ($t(9) = 2{,}46$, $p < 0{,}05$) und zwischen Liste und Baumdarstellung ($t(9) = 3{,}32$, $p < 0{,}01$). Daraus lässt sich schlussfolgern, dass die Baumdarstellung signifikant schneller als Fisheye View Liste und Liste und dass die Fisheye View Liste signifikant schneller als die Liste war.

| | Fisheye View Liste | Liste | Baumdarstellung |
|---|---|---|---|
| Selection | 4,05 (0,73) | 4,15 (0,66) | 3,57 (0,43) |
| Drag & Drop | 5,02 (0,75) | 5,62 (1,27) | 4,66 (0,64) |

Tabelle 7.3: Mittelwert der Taskerfüllungszeiten der Usability Tests in Sekunden. Standardabweichung in Klammern.

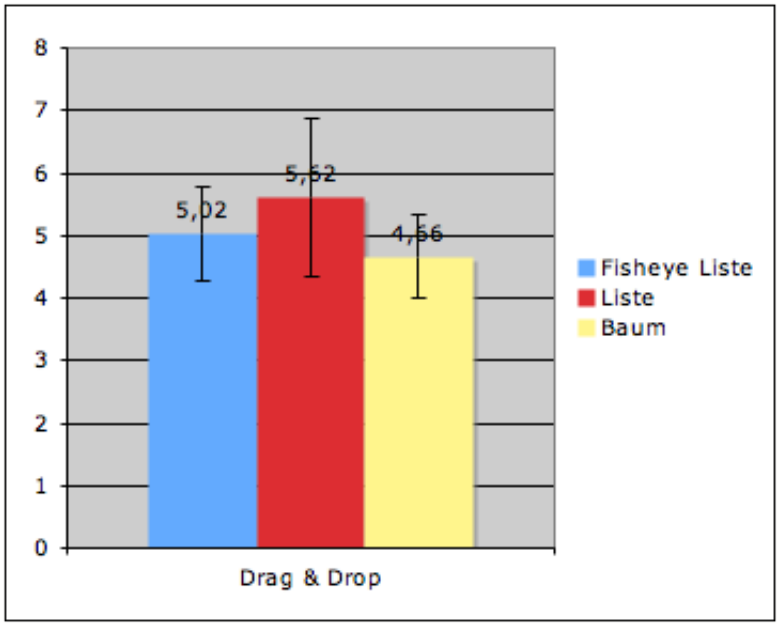


Abbildung 7.2: Mittelwert der Taskerfüllungszeiten der Usability Tests für Drag & Drop Tasks in Sekunden.

## 7.2 Korrektheit

Für jede Testperson wurde die Taskkorrektheitsrate je Tasktyp und Interfacetyp berechnet. Die Korrektheitsrate für Selection Tasks je Testperson ist in Tabelle 7.4 ersichtlich und die Korrektheitsrate für Drag & Drop Tasks ist in Tabelle 7.5 ersichtlich. Die Taskkorrektheitsrate gibt den Anteil der korrekt erfüllten Tasks in Prozent an. Für eine genaue Beschreibung der Erfülllungs- bzw. Nichterfüllungskriterien der Tasks siehe Kapitel 6.6.

Auf Basis der Taskkorrektheitsrate pro Person wurde die mittlere Taskkorrektheitsrate für jeden Interfacetyp berechnet (siehe Tabelle 7.6). Insgesamt erwies sich die Baumdarstellung für beide Tasktypen als am wenigsten fehleranfällig in der Benutzung. Weder die Analyse mittels Repeated Measures ANOVA noch Paired Samples T Test ergab für Selection Tasks oder für Drag & Drop Tasks signifikante Unterschiede der Taskkorrektheitsrate zwischen irgendwelchen der drei Interfacetypen.

| | Fisheye View Liste | Liste | Baumdarstellung |
|---|---|---|---|
| TP 1 | 93,33 (25,37) | 100 (0) | 100 (0) |
| TP 2 | 100 (0) | 100 (0) | 100 (0) |
| TP 3 | 100 (0) | 100 (0) | 100 (0) |
| TP 4 | 100 (0) | 96,67 (18,26) | 93,33 (25,37) |
| TP 5 | 100 (0) | 100 (0) | 100 (0) |
| TP 6 | 100 (0) | 96,67 (18,26) | 100 (0) |
| TP 7 | 93,33 (25,37) | 100 (0) | 100 (0) |
| TP 8 | 96,67 (18,26) | 100 (0) | 100 (0) |
| TP 9 | 93,33 (25,37) | 100 (0) | 100 (0) |
| TP 10 | 96,67 (18,26) | 90 (30,51) | 100 (0) |

Tabelle 7.4: Taskkorrektheitsrate der Usability Tests für Selection Tasks pro Testperson in Prozent. Standardabweichung in Klammern.

| | Fisheye View Liste | Liste | Baumdarstellung |
|---|---|---|---|
| TP 1 | 100 (0) | 100 (0) | 100 (0) |
| TP 2 | 100 (0) | 100 (0) | 100 (0) |
| TP 3 | 96,67 (18,26) | 100 (0) | 100 (0) |
| TP 4 | 100 (0) | 100 (0) | 100 (0) |
| TP 5 | 100 (0) | 100 (0) | 100 (0) |
| TP 6 | 100 (0) | 100 (0) | 100 (0) |
| TP 7 | 100 (0) | 100 (0) | 100 (0) |
| TP 8 | 100 (0) | 100 (0) | 100 (0) |
| TP 9 | 100 (0) | 93,33 (25,37) | 100 (0) |
| TP 10 | 96,67 (18,27) | 100 (0) | 100 (0) |

Tabelle 7.5: Taskkorrektheitsrate der Usability Tests für Drag & Drop Tasks pro Testperson in Prozent. Standardabweichung in Klammern.

| | Fisheye View Liste | Liste | Baumdarstellung |
|---|---|---|---|
| Selection | 97,33 (3,06) | 98,33 (3,24) | 99,33 (2,11) |
| Drag & Drop | 99,33 (1,41) | 99,33 (2,11) | 100 (0) |

Tabelle 7.6: Mittlere Taskkorrektheitsrate der Usability Tests in Prozent. Standardabweichung in Klammern.

Bei den insgesamt 1800 Messdatensätze wurden insgesamt 19 Fehler gemessen. Bei einer näheren Betrachtung dieser Fehler fällt auf, dass bei 15 dieser Fehler das vorgegebene Ziel um nur ein Listenelement verfehlt wurde und dass bei zwei weiteren Fehlern das vorgegebene Ziel um weniger als 5 Listenelemente verfehlt wurde. Lediglich bei vier von den 19 Fehlern wurde das vorgegebene Ziel bei weitem verfehlt. Dies deutet darauf hin, dass es sich bei den aufgetretenen Fehlern um knappe Ausrutscher im Zielvorgang handelte.

## 7.3 Präferenzen und Zufriedenheit der Testpersonen

Neben der Erhebung von quantitativen Daten wurden die Testteilnehmer gebeten, einen Post-Test-Fragebogen zur Erhebung qualitativer Daten zu beantworten. Im Rahmen dieses Fragebogens war es Aufgabe der Testpersonen, sämtliche drei Interfacetypen auf einer neunstufigen semantisch-differentiellen Skala hinsichtlich sieben unterschiedlicher, bipolarer Charakteristika zu bewerten. Die Mittelwerte der Präferenzangaben je Charakteristik sind in Tabelle 7.7 ersichtlich.

Die Baumdarstellung konnte sich hinsichtlich sechs der sieben untersuchten Charakteristika vor Fisheye View Liste und Liste durchsetzen und wurde am besten bewertet. Lediglich bezüglich der Charakteristik "Langweilig - Unterhaltsam" wurde die Fisheye View Liste am besten bewertet. Ein möglicher Grund hierfür ist die Neuartigkeit und dynamische Natur der Fisheye View Liste. Bezüglich der sechs Charakteristika, in denen die Baumdarstellung am besten abgeschnitten hat, schnitt die Fisheye View Liste in vier Fällen besser als die Liste ab, nämlich bei "Furchtbar - Großartig", "Frustrierend - Zufriedenstellend", "Langsam - Schnell" und "Lästig - Angenehm". Die Fisheye View Liste schnitt bezüglich der Charakteristika "Schwierig - Einfach" und "Schwierig zu erlernen - Einfach zu erlernen" am schlechtesten ab, was dar-

an liegen kann, dass es sich bei der Fisheye View Liste um einen neuen und unvertrauten Interfacetyp handelt. Insgesamt geht die Baumdarstellung aus dieser Bewertung als klarer Favorit der Testpersonen hervor, während die Liste auf die stärkste Abneigung stieß.

| Skala von 1 - 9 | Fisheye V. Liste | Liste | Baumdarstellung |
|---|---|---|---|
| Furchtbar-Großartig | 6,2 (1,4) | 3,3 (1,6) | **7,6** (1,3) |
| Frustrierend-Zufriedenstellend | 6,7 (1,6) | 3,9 (1,4) | **7,3** (1,6) |
| Schwierig-Einfach | 7,0 (1,8) | 7,4 (1,3) | **8,5** (0,8) |
| Langsam-Schnell | 6,7 (1,3) | 3,6 (1,2) | **7,4** (1,5) |
| Schwierig-Einfach zu erlernen | 7,2 (2,1) | 8,0 (1,2) | **8,5** (0,7) |
| Langweilig-Unterhaltsam | **7,0** (1,7) | 3,6 (1,3) | 5,1 (1,2) |
| Lästig-Angenehm | 6,0 (2) | 3,4 (1,6) | **7,3** (1,6) |

Tabelle 7.7: Mittelwerte der Präferenzangaben aus dem Post-Test-Fragebogen je Charakteristik. Standardabweichung in Klammern. Die Skala verläuft von 1 für das linke Extremum bis 9 für das rechte Extremum. Die am stärksten ausgeprägten Präferenzen sind fett hervorgehoben.

Eine Wilcoxon-Analyse der erhobenen Präferenzdaten erbrachte folgende Ergebnisse:

- Hinsichtlich der Charakteristik “Furchtbar - Großartig” ließen sich signifikante Unterschiede zwischen Liste (3,3) und Fisheye View Liste (6,2) ($z = 2{,}32$, $p < 0{,}05$) sowie zwischen Liste und Baumdarstellung (7,6) ($z = 2{,}81$, $p < 0{,}01$) feststellen.

- Hinsichtlich der Charakteristik “Frustrierend - Zufriedenstellend” ließen sich signifikante Unterschiede zwischen Liste (3,9) und Fisheye View Liste (6,7) ($z = 2{,}36$, $p < 0{,}05$) sowie zwischen Liste und Baumdarstellung (7,3) ($z = 2{,}83$, $p < 0{,}01$) feststellen.

- Hinsichtlich der Charakteristik “Schwierig - Einfach” ließen sich signifikante Unterschiede zwischen Fisheye View Liste (7) und Baumdarstellung (8,5) ($z = 1{,}98$, $p < 0{,}05$) sowie zwischen Liste (7,4) und Baumdarstellung ($z = 2{,}03$, $p < 0{,}05$) feststellen.

- Hinsichtlich der Charakteristik “Langsam - Schnell” ließen sich signifikante Unterschiede zwischen Liste (3,6) und Fisheye View Liste (6,7) ($z = 2{,}84$, $p < 0{,}01$) sowie zwischen Liste und Baumdarstellung (7,4) ($z = 2{,}72$, $p < 0{,}01$) feststellen.

- Hinsichtlich der Charakteristik “Schwierig zu erlernen - Einfach zu erlernen” ließen sich keine signifikanten Unterschiede zwischen irgendwelchen der drei Interfacetypen feststellen.

- Hinsichtlich der Charakteristik “Langweilig - Unterhaltsam” ließen sich signifikante Unterschiede zwischen Liste (3,6) und Fisheye View Liste (7) ($z = 2{,}38$, $p < 0{,}05$), zwischen Fisheye View Liste und Baumdarstellung (5,1) ($z = 2{,}14$, $p < 0{,}05$) sowie zwischen Liste und Baumdarstellung ($z = 2{,}21$, $p < 0{,}05$) feststellen.

- Hinsichtlich der Charakteristik “Lästig - Angenehm” ließen sich signifikante Unterschiede zwischen Liste (3,4) und Fisheye View Liste (6) ($z = 2{,}03$, $p < 0{,}05$) sowie zwischen Liste und Baumdarstellung (7,3) ($z = 2{,}83$, $p < 0{,}01$) feststellen.

Tabelle 7.8 zeigt die durchschnittlichen Präferenzwerte aus dem Post-Test-Fragebogen je Testperson. Hierbei wurde der Mittelwert der Wertungen der einzelnen Charakteristika von jeder Testperson berechnet. Betrachtet man diese Ergebnisse, so gaben sechs der zehn Testpersonen im Schnitt der Baumdarstellung den Vorzug, während vier Testpersonen der Fisheye View den Vorzug gaben. Acht Testpersonen bewerteten im Schnitt die Liste am schlechtesten und zwei Testpersonen bewerteten die Fisheye View am schlechtesten.

Neben der Bewertung der drei unterschiedlichen Interfacetypen auf der beschriebenen semantisch-differentiellen Skala war es Aufgabe der Testpersonen, den Interfacetyp der ihnen am besten gefiel, den Interfacetyp der ihnen am schlechtesten gefiel und den Interfacetyp, mit dem sie den Eindruck hatten, die Aufgaben am schnellsten erfüllen zu können im Post-Test-Fragebogen festzuhalten. Die Ergebnisse dieser Befragung sind in Tabelle 7.9 ersichtlich:

| | Fisheye V. Liste | Liste | Baumdarstellung |
|---|---|---|---|
| TP 1 | 7,0 (1,9) | 4,1 (3,0) | **8,1** (1,9) |
| TP 2 | **8,3** (0,8) | 4,9 (2,9) | 6,7 (1,7) |
| TP 3 | 5,9 (0,9) | 4,3 (2,6) | **8,3** (0,5) |
| TP 4 | 5,4 (2,0) | 5,9 (2,2) | **8,1** (1,9) |
| TP 5 | 6,0 (2,1) | 5,3 (1,6) | **6,7** (1,3) |
| TP 6 | **8,0** (0,8) | 4,4 (3,1) | 6,4 (2,1) |
| TP 7 | **7,1** (1,1) | 4,4 (2,2) | 7,0 (1,0) |
| TP 8 | **6,7** (1,6) | 3,6 (2,5) | 6,6 (1,7) |
| TP 9 | 5,4 (2,0) | 6,1 (1,2) | **7,9** (1,5) |
| TP 10 | 7,0 (1,0) | 4,4 (0,8) | **8,0** (1,4) |

Tabelle 7.8: Mittelwerte der Präferenzangaben aus dem Post-Test-Fragebogen je Testperson. Es wurde der Mittelwert der Beurteilung sämtlicher Fragen je Testperson ermittelt. Standardabweichung in Klammern. Die stärksten Präferenzen sind fett hervorgehoben.

Acht Testpersonen gefiel die Baumdarstellung am besten und zwei Testpersonen gefiel die Fisheye View Liste am besten. Acht Testpersonen befanden die Liste als am schlechtesten und zwei Testpersonen fanden die Fisheye View Liste am schlechtesten. Diese Ergebnisse spiegeln im wesentlichen auch in etwa die Beobachtungen aus Tabelle 7.8 wieder. Neun Testpersonen hatten das Gefühl, die Tasks mit dem Bauminterface am schnellsten erfüllen zu können und eine Testperson hatte das Gefühl, die Tasks mit der Fisheye View Liste am schnellsten erfüllen zu können.

## 7.4 Interaktionsverhalten und Beobachtungen

Neben der Erhebung von performancerelevanten Daten zu Taskerfüllungszeit und Taskkorrektheit wurden vom Testprogramm auch weitere Daten über das Interaktionsverhalten der Testpersonen während der Durchführung der Tasks erhoben und automatisch protokolliert. Für eine genaue Übersicht über sämtliche Daten, die vom Testprogramm erhoben wurden, siehe Kapitel 6.6.

Unter anderem wurde protokolliert, welche Elemente den Testpersonen

| | Besten | Schlechtesten | Schnellsten |
|---|---|---|---|
| TP 1 | Tree | Liste | Tree |
| TP 2 | Fisheye View Liste | Liste | Tree |
| TP 3 | Tree | Liste | Tree |
| TP 4 | Tree | Fisheye View Liste | Tree |
| TP 5 | Tree | Liste | Tree |
| TP 6 | Fisheye View Liste | Liste | Fisheye View Liste |
| TP 7 | Tree | Liste | Tree |
| TP 8 | Tree | Liste | Tree |
| TP 9 | Tree | Fisheye View Liste | Tree |
| TP 10 | Tree | Liste | Tree |

Tabelle 7.9: Ergebnisse der Befragung der Testpersonen, welchen Interfacetyp sie am Besten, am Schlechtesten, bzw. am Schnellsten empfanden.

zur Selektion bzw. Sortierung in der Aufgabenliste vorgegeben wurden. Auf Basis dieser Daten ließ sich eine interessante Beobachtung bezüglich der Taskerfüllungszeiten bei Verwendung der Fisheye View Listen anstellen: Sowohl bei Selection Tasks als auch bei Drag & Drop Tasks war die Taskerfüllungszeit merklich kürzer, wenn das Element aus der Aufgabenliste sich unter den ersten zehn oder letzten zehn Elementen der Fisheye View Liste befand. Bei Selection Tasks war die durchschnittliche Taskerfüllungszeit bei Elementen unter den ersten oder letzen zehn Listeneinträgen der Fisheye View Liste 2,85 Sekunden (SD = 0,91) und bei Elementen aus dem mittleren Bereich der Fisheye View Liste 4,30 Sekunden (SD = 1,55), ein signifikanter Unterschied ($t(290) = 6{,}50$, $p < 0{,}01$). Bei Drag & Drop Tasks war die durchschnittliche Taskerfüllungszeit bei Elementen unter den ersten oder letzen zehn Listeneinträgen der Fisheye View Liste 3,69 Sekunden (SD = 1,21) und bei Elementen aus dem mittleren Bereich der Fisheye View Liste 5,43 Sekunden (SD = 1,90), ein signifikanter Unterschied ($t(296) = 7{,}22$, $p < 0{,}01$). Eine mögliche Erklärung für diese Beobachtung besteht darin, dass die ungefähre Position der Elemente am Anfang und am Ende der Fisheye View Liste besonders einfach zu identifizieren sind. In einer alphabetisch sortierten Liste ist offensichtlich, dass ein Element mit dem Anfangsbuchstaben A weit am oberen Ende der Liste zu finden sein muss, während ein Element mit dem Anfangsbuchstaben Z am Ende der Liste zu finden sein muss. Obwohl

die Implementierung der alphabetischen Indizes dem Anwender grundsätzlich bei der ungefähren Lokalisierung von Elementen anhand des Anfangsbuchstabens behilflich sein soll, ist bei Randelementen eine solche ungefähre Lokalisierung auch ohne Zuhilfenahme der alphabetischen Indizes problemlos möglich und dementsprechend schneller.

Bei einer näheren Betrachtung der Taskerfüllungszeiten beim Interfacetyp Liste mit Scrollbalken, bei der ebenfalls die Position der Aufgabenelemente in der Listendarstellung einbezogen wurde, bestätigte sich eine bereits im Vorfeld bestehende, naheliegende Vermutung: Die Taskerfüllungszeit erwies sich als deutlich kürzer, wenn das Aufgabenelement im Intial-Viewport der Liste gelegen war, also in jenem Bereich, der im Ausgangszustand der Liste direkt und ohne zu scrollen sichtbar ist. Diese Beobachtung ließ sich sowohl Selection als auch Drag & Drop Tasks anstellen. Bei Selection Tasks war die durchschnittliche Taskerfüllungszeit bei Aufgabenelementen im Initial-Viewport der Liste 2,97 Sekunden (SD = 1,18) und bei Aufgabenelementen außerhalb des Inital-Viewports der Liste 4,70 Sekunden (SD = 1,64), ein signifikanter Unterschied ($t(293) = 9{,}21$, $p < 0{,}01$). Bei Drag & Drop Tasks war die durchschnittliche Taskerfüllungszeit bei Aufgabenelementen im Initial-Viewport der Liste 3,65 Sekunden (SD = 1,49) und bei Aufgabenelementen außerhalb des Inital-Viewports der Liste 6,49 Sekunden (SD = 2,28), ein signifikanter Unterschied ($t(296) = 10{,}97$, $p < 0{,}01$).

Weiters wurde das Autoscroll- und Autoexpandverhalten der Testpersonen für Drag & Drop Tasks bei Listen mit Scrollbalken und Baumdarstellung automatisch mitprotokolliert. Für die Untersuchung des Autoscrollverhaltens wurden nur jene Drag & Drop Operationen betrachtet, bei denen sich das Drag & Drop Ziel außerhalb des Initial-Viewports der Liste befand, d.h. bei denen das Drag & Drop Ziel auch tatsächlich so gelegen war, dass die Testperson scrollen musste um zu ihm zu gelangen. Von diesen Drag & Drop Operationen wurde in nur 32 von 206 Fällen (15,53 %) die Autoscrollfunktion genutzt. Betrachtet man das Autoexpandverhalten der Testpersonen, so fällt auf, dass bei 269 der 300 Drag & Drop Operationen (89,66 %) die Autoexpand-Funktion des Baums verwendet wurde. Sowohl Autoscroll- als

auch Autoexpandfunktion ermöglichen dem Anwender bei Drag & Drop Operationen, den Drag & Drop Vorgang in einer einzigen fließenden Bewegung durchzuführen, ohne zuvor scrollen oder Ordner öffnen zu müssen. Das Nutzungsverhalten dieser beiden Funktionen im Rahmen der Tests weist darauf hin, dass die Autoscrollfunktion von den Testpersonen als nicht ausreichend bequem und effizient angesehen wurde, um diese konsequent zu nutzen, während die Autoexpandfunktion überwiegend verwendet wurde.

# Kapitel 8

# Diskussion der Ergebnisse

Die Ergebnisse der Usability Tests weisen darauf hin, dass das Interface mit Baumdarstellung sowohl für Selection Tasks als auch Drag & Drop Tasks hinsichtlich Taskerfüllungszeit der Fisheye View Liste und der Liste mit Scrollbalken überlegen ist. Die Testteilnehmer waren in der Lage, die Aufgaben bei dem Interface mit Baumdarstellung signifikant schneller zu lösen als mit den beiden anderen Interfacevarianten. Für Selection Tasks ließ sich kein signifikanter Unterschied bei den Taskerfüllungszeiten zwischen Fisheye View Liste und Liste mit Scrollbalken feststellen. Bei Drag & Drop Tasks hingegen war ein signifikanter Unterschied bei den Taskerfüllungszeiten zwischen Fisheye View Liste und Liste mit Scrollbalken bemerkbar: Die Fisheye View Liste erwies sich für diese Art von Tasks als signifikant schneller.

Eine Erklärung für diesen Unterschied besteht darin, dass die Fisheye View Liste es Anwendern erlaubt, den Drag & Drop Vorgang in einer einzigen, ununterbrochenen, flüssigen Bewegung durchzuführen. Dies wird durch die visuelle Fisheyeverzerrung ermöglicht, welche die gleichzeitige Darstellung sämtlicher Listenelemente in einem komplett sichtbaren Fenster ohne Scrollbalken erlaubt. Wenn hingegen ein Listeneintrag bei der Liste mit Scrollbalken nicht im Ausgangs-Viewport der Liste sichtbar ist, so sind die Anwender entweder dazu gezwungen, zuerst in jenen Bereich zu scrollen, in

dem sich das gewünschte Zielelement befindet bevor der Drag & Drop Vorgang begonnen werden kann, oder aber auf die Autoscroll-Funktionalität der Liste zurückzugreifen. Wie im Kapitel 7.4 beschrieben wurde die Autoscroll-Funktionalität von den Testpersonen im Rahmen der Usability Tests kaum genutzt. Auf die Frage, wieso sie diese Funktionalität so wenig verwendeten, gaben viele Testpersonen an, dass Sie dieses Feature als unbequem und mühsam empfanden. Darüber hinaus wurde die Autoscroll-Funktionalität als zu langsam empfunden, um damit eine größere Distanz in der Liste zu scrollen. Dementsprechend waren die Testpersonen bei Verwendung der Fisheye View Liste in der Lage, die Drag & Drop Operation in einer einzigen Bewegung durchzuführen, während die Operation bei Verwendung der Liste mit Scrollbalken meist in zwei unterschiedliche Bewegungskomponenten zerlegt war, was zu schnelleren Taskerfüllungszeiten mit dem Fisheye View Listeninterface führte.

Im Gegensatz zu den Taskerfüllungszeiten ließ sich im Rahmen der Analyse der Taskkorrektheitsraten kein signifikanter Effekt von Interfacetyp feststellen. Die Taskkorrektheitsrate der Fisheye View Liste war zwar sowohl für Selection als auch Drag & Drop Tasks tendennziell niedriger als bei den beiden anderen Interfacevarianten, jedoch ließ sich kein signifkanter Unterschied zu Baumdarstellung und Liste mit Scrollbalken feststellen.

Die Ergebnisse des Background-Fragebogens ergaben, dass die Testpersonen im Allgemeinen dem Interface mit Baumdarstellung den Vorzug gaben. Dieses Interface wurde in sechs von den sieben Bewertungscharakteristika am besten bewertet, lediglich bezüglich der Charakteristik “Langweilig - Unterhaltsam” schnitt die Fisheye View Liste besser als die Baumdarstellung ab. Die Fisheye View Liste schnitt darüber hinaus bei den Charakteristika “Schwierig - Einfach” und “Schwierig zu erlernen - Einfach zu erlernen” am schlechtesten ab. Diese Bewertungen kann man dadurch erklären, dass die Fisheye View Liste für die Testpersonen völlig neu und unbekannt war und deshalb einerseits kurzweiliger und unterhaltsamer in der Benützung war, andererseits aber auch als schwieriger zu bedienen empfunden wurde. Die Liste mit Scrollbalken erzielte im Schnitt die schlechteste Bewertung und erzielte

bei fünf der sieben Bewertungscharakteristika das niedrigste Ergebnis.

Im Allgemeinen gaben die Testpersonen der Fisheye View Liste den Vorzug über die Liste mit Scrollbalken. Darüber hinaus gaben sämtliche Testpersonen an, dass sie die Bedienung der Fisheye View Liste als intuitiv und leicht verständlich empfanden. Betrachtet man die individuellen Präferenzen der einzelnen Testpersonen, so fällt jedoch auf, dass Vorlieben und Zufriedenheit mit der Fisheye View Liste stark schwankten: Während zwei Testpersonen angaben, dass ihnen die Fisheye View Liste am besten gefiel, gaben auch zwei andere Testpersonen an, dass ihnen die Fisheye View Liste am wenigsten gefiel. Diese starken individuellen Unterschiede in den Präferenzen spiegeln auch die Ergebnisse der früheren Studie zum Thema Fisheye Menus von Bederson (2000) wider.

Ein Usabilityproblem, welches im Rahmen der Tests offensichtlich wurde und von allen Testteilnehmern beklagt wurde, war die Schwierigkeit der präzisen Zielauswahl am Ende des Zielvorgangs. Diese Schwierigkeit ergibt sich aus dem radikal verringerten Motorspace zur Auswahl von Elementen bei visuell verzerrten Fisheye Views. Wie bereits in Kapitel 3.3 diskutiert ergibt sich dieser Effekt aus der Entkoppelung von visueller Größe und Motorgröße für die Auswahl von Elementen. Diese Entkoppelung von Darstellungsgröße und Motorgröße ist unumgänglich, weil sämtliche Listenelemente zu jedem Zeitpunkt mittels Zeigerbewegung in der fixen vertikalen Größe der Liste selektierbar sein müssen. Fisheye Menus (Bederson, 2000) bedienen sich des in Kapitel 4.2 beschriebenen Focus Lock Modus, um eine Lösung für dieses Problem bereitzustellen. Die Evaluierung der Fisheye Menus ergab jedoch, dass dieser Focus Lock Modus für Anwender nicht intuitiv verständlich war, weswegen im Rahmen dieser Arbeit von der Implementierung eines solchen Modus abgesehen wurde. Mögliche Lösungsansätze für das Problem des verringerten Motorspace wurden auch bereits kurz in Kapitel 3.3 diskutiert.

Obwohl die Testteilnehmer im Allgemeinen der Baumdarstellung den Vorzug gaben und damit auch signifikant schnellere Taskerfüllungszeiten für Selection und Drag & Drop Tasks erzielten, weisen die schnelleren Tas-

kerfüllungszeiten bei Drag & Drop Tasks unter Verwendung der Fisheye View Liste im Vergleich zur Liste mit Scrollbalken sowie die generelle Präferenz der Fisheye View Liste gegenüber der Liste mit Scrollbalken auf gewisse Vorteile bei der Verwendung von visuellen Fisheye Verzerrungseffekten in graphischen User Interfaces hin. Die Ergebnisse der Usability Tests lassen die Anwendung einer eindimensionalen visuellen Fisheyeverzerrung in graphischen User Interfaces grundsätzlich vielversprechend erscheinen. Visuelle Fisheye Verzerrungseffekte können die Anzahl der unterschiedlichen Bewegungskomponenten, welche zur Durchführung einer Operation notwendig sind, in gewissen Fällen verringern, was positive Auswirkungen auf die Taskerfüllungszeit haben kann, wie sich auch im Rahmen dieser Arbeit gezeigt hat. Eine visuelle Fisheye Verzerrung ließe sich auch ohne weiteres auf diverse andere graphische User Interface Elemente anwenden, wie beispielsweise auf Baumdarstellungen, wie der in Kapitel 5.3 präsentierte Prototyp eines Fisheye View Baums zeigt. Grundsätzlich lässt sich feststellen, dass die Verwendung eines visuellen Fisheye Verzerrungseffekts bei der Visualisierung von listenähnlichen Strukturen als Alternative zu Scrollmechanismen in Betracht gezogen werden kann.

Die präsentierte Implementierung einer Fisheye View Liste ist grundsätzlich auch für andere Eingabegeräte als Maus geeignet. In der Praxis hat sich beispielsweise gezeigt, dass die hier präsentierte Fisheye View Liste grundsätzlich für die Verwendung mit einem Touchpad geeignet ist. Einzige Grundvoraussetzung für die Verwendbarkeit der Fisheye View Liste ist die Interaktion mit dem System über einen Zeiger, der kontinuierlich bewegt wird. In einem informellen Test wurde beispielsweise festgestellt, dass die Verwendung dieser Implementierung einer Fisheye View Liste auf einem Tablet PC nur eingeschränkt möglich ist, weil die Interaktion mit dem System mittels Stiftberührungen zu einem abrupten Hüpfen des Zeigers führt und deshalb die Bestimmung des Fokuspunkts auf Basis der Zeigerposition nicht möglich ist. Für die Anwendung auf Touchscreens wären deshalb Änderungen am Interaktionsverhalten der Fisheye View Liste notwendig.

Vergleicht man die Ergebnisse der Usability Tests mit den Ergebnissen der

Studie über Fisheye Menus von Bederson (2000) so lassen sich interessante Parallelen feststellen. Obwohl offensichtlich Unterschiede zwischen hierarchischen Dropdown-Menüs und hierarchischen Baumdarstellungen bestehen, so lässt sich doch feststellen, dass in beiden Tests die alphabetische Gruppierung in einer hierarchischen Struktur den größten Zuspruch der Testpersonen fand. Verglichen mit einer linearen Darstellung mit Scrollbalken erwies sich die Fisheye View in beiden Studien als beliebter bei den Testpersonen. Auch der Vergleich der gemessenen Zeiten bei der Erfüllung von Selection Tasks zwischen Fisheye Menus und Fisheye View Lists weist Ähnlichkeiten auf. Im Rahmen des Expert Timing Tests der Fisheye Menus erwies sich die hierarchische Gruppierung am schnellsten, gefolgt von Fisheye View, Scrollbalken und Arrowbar. In den im Rahmen dieser Arbeit durchgeführten Usability Tests erwies sich ebenfalls die hierarchische Gruppierung in Ordnern am schnellsten, gefolgt von Fisheye View Listen und Liste mit Scrollbalken.

# Kapitel 9

# Conclusio

Im Rahmen dieser Arbeit wurde die Usability und Userakzeptanz von Fisheye View Listen im Vergleich zu traditionellen Listen mit Scrollbalken und Baumdarstellungen evaluiert. Zu diesem Zweck wurden vergleichende Usability Tests der unterschiedlichen Interfacevarianten durchgeführt. Die Ergebnisse der Tests zeigen, dass die Taskerfüllungszeit bei Baumdarstellungen sowohl für Selection- als auch für Drag & Drop Tasks am schnellsten war, während sich die traditionelle Liste mit Scrollbalken bei beiden Tasktypen als am langsamsten erwies. Die Fisheye View Liste fiel bezüglich Taskerfüllungsgeschwindigkeit zwischen Baumdarstellung und Liste mit Scrollbalken und erwies sich für Drag & Drop Tasks als signifikant schneller als die Liste mit Scrollbalken. Ein möglicher Grund hierfür besteht darin, dass der Drag & Drop Vorgang bei Fisheye View Listen aus einer einzigen, flüssigen Bewegung besteht, während bei einer Liste mit Scrollbalken zuerst in den richtigen Zielbereich gescrollt werden muss, bevor der eigentliche Drag & Drop Vorgang durchgeführt werden kann. Bezüglich Taskkorrektheitsrate ließen sich keine signifikanten Unterschiede zwischen den drei Interfacevarianten feststellen. Die erhobenen Daten über Anwenderpräferenzen spiegelten die Ergebnisse der Auswertung der Taskerfüllungszeiten wieder: Die Testteilnehmer bevorzugten überwiegend das Baumdarstellungs-Interface, gefolgt von Fisheye View Liste und Liste mit Scrollbalken auf dem letzten Platz.

Obwohl sich die Fisheye View Liste weder hinsichtlich Taskerfüllungszeit noch Anwenderpräferenzen gegen Baumdarstellungen durchsetzen konnte, sind die Ergebnisse für die Anwendung einer eindimensionalen visuellen Fisheyeverzerrung in User Interfaces grundsätzlich vielversprechend. Eine solche visuelle Fisheyeverzerrung ließe sich beispielsweise auch auf Baumdarstellungen anwenden, und so wie sich die Fisheye View Liste der traditionellen Liste mit Scrollbalken als überlegen herausgestellt hat, so könnte auch ein Fisheye View Baum Vorteile gegenüber einem herkömmlichen Baum aufweisen. Visuelle Fisheye Verzerrungseffekte könnten als alternative zu Scrollbalken Anwendung finden und könnten in manchen Fällen die Anzahl der unterschiedlichen Bewegungskomponenten reduzieren, welche zur Durchführung eines Tasks notwendig sind.

Weiters besteht bei der in dieser Arbeit präsentierten Implementierung einer Fisheye View Liste noch Verbesserungspotential. Der radikal verringerte Motorspace zur Auswahl von Listenelementen erwies sich im Rahmen der Usability Tests als eines der größten Probleme. Dennoch erwies sich die Fisheye View Liste der traditionellen Liste mit Scrollbalken als überlegen. Eine Lösung dieses Problems erscheint auf jeden Fall notwendig, bevor Fisheye Views in der Praxis breite Anwendung finden können.

Für zukünftige Arbeiten auf diesem Gebiet bietet sich deshalb die Auseinandersetzung mit möglichen Lösungsansätzen zur Beseitigung des Problems des verringerten Motorspace bei der Auswahl von Elementen und die Entwicklung und Evaluierung eines voll funktionsfähigen Fisheye View Baums an.

# Literaturverzeichnis


Apperley, M. D., Tzavaras, I., & Spence, R. (1982). A bifocal display technique for data presentation. In *Proceedings of Eurographics'82, Conference of the European Association for Computer Graphics*, (pp. 27–43).

Backvall, P., Martensson, P., & Qvarfordt, P. (2000). Using fisheye for navigation on small displays. In *Proceedings of NordiCHI*.

Baudisch, P., Lee, B., & Hanna, L. (2004). Fishnet, a fisheye web browser with search term popouts: a comparative evaluation with overview and linear view. In *AVI '04: Proceedings of the working conference on Advanced visual interfaces*, (pp. 133–140). New York, NY, USA: ACM Press.

Bederson, B. B. (2000). Fisheye menus. In *UIST '00: Proceedings of the 13th annual ACM symposium on User interface software and technology*, (pp. 217–225). New York, NY, USA: ACM Press.

Bederson, B. B., Clamage, A., Czerwinski, M. P., & Robertson, G. G. (2003). A fisheye calendar interface for pdas: providing overviews for small displays. In *CHI '03: CHI '03 extended abstracts on Human factors in computing systems*, (pp. 618–619). New York, NY, USA: ACM Press.

Bederson, B. B., Clamage, A., Czerwinski, M. P., & Robertson, G. G. (2004). Datelens: A fisheye calendar interface for pdas. *ACM Trans. Comput.-Hum. Interact.*, *11*(1), 90–119.

Bederson, B. B., & Hollan, J. D. (1994). Pad++: a zooming graphical interface for exploring alternate interface physics. In *UIST '94: Proceedings of*

*the 7th annual ACM symposium on User interface software and technology*, (pp. 17–26). New York, NY, USA: ACM Press.

Blanch, R., Guiard, Y., & Beaudouin-Lafon, M. (2004). Semantic pointing: improving target acquisition with control-display ratio adaptation. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, (pp. 519–526). New York, NY, USA: ACM Press.

Card, S. K., Mackinlay, J. D., & Shneiderman, B. (Eds.) (1999). *Readings in information visualization: using vision to think*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Demian, P., & Fruchter, R. (2006). Finding and understanding reusable designs from hierarchical repositories. *Information Visualization Journal*, *5*(1), 28–46.

Donelson, W. C. (1978). Spatial management of information. In *SIGGRAPH '78: Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, (pp. 203–209). New York, NY, USA: ACM Press.

Fitts, P. M. (1954). The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, *47*(6), 381–391.

Furnas, G. W. (1982). The fisheye view: a new look at structured files. *Bell Laboratories Technical Memorandum, #82-11221-22*.

Furnas, G. W. (1986). Generalized fisheye views. *SIGCHI Bull.*, *17*(4), 16–23.

Furnas, G. W. (1997). Effective view navigation. In *CHI '97: Proceedings of the SIGCHI conference on Human factors in computing systems*, (pp. 367–374). New York, NY, USA: ACM Press.

Furnas, G. W. (2006). A fisheye follow-up: further reflections on focus + context. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, (pp. 999–1008). New York, NY, USA: ACM Press.

Gutwin, C. (2002). Improving focus targeting in interactive fisheye views. In *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*, (pp. 267–274). New York, NY, USA: ACM Press.

Hornbæk, K., Bederson, B. B., & Plaisant, C. (2002). Navigation patterns and usability of zoomable user interfaces with and without an overview. *ACM Trans. Comput.-Hum. Interact.*, *9*(4), 362–389.

Hornbæk, K., & Frøkjær, E. (2001). Reading of electronic documents: the usability of linear, fisheye, and overview+detail interfaces. In *CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems*, (pp. 293–300). New York, NY, USA: ACM Press.

Hornbæk, K., & Frøkjær, E. (2003). Reading patterns and usability in visualizations of electronic documents. *ACM Trans. Comput.-Hum. Interact.*, *10*(2), 119–149.

Huang, M. L., & Quan, W. (2004). 21df-browser: A multiple fisheye distortion technique for visualizing and navigating hierarchies with large number of leaves. *iv*, (pp. 277–285).

Jakobsen, M. R., & Hornbæk, K. (2006). Evaluating a fisheye view of source code. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, (pp. 377–386). New York, NY, USA: ACM Press.

Kadmon, N., & Shlomi, E. (1978). A polyfocal projection for statistical surfaces. *Cartograph J.*, *15*(1), 36–41.

Kaptelinin, V. (1995). A comparison of four navigation techniques in a 2d browsing task. In *CHI '95: Conference companion on Human factors in computing systems*, (pp. 282–283). New York, NY, USA: ACM Press.

Karlson, A. K., Bederson, B. B., & SanGiovanni, J. (2005). Applens and launchtile: two designs for one-handed thumb use on small devices. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, (pp. 201–210). New York, NY, USA: ACM Press.

Lamping, J., & Rao, R. (1999). The hyperbolic browser: a focus + context technique for visualizing large hierarchies. *Readings in information visualization: using vision to think*, (pp. 382–408).

Lamping, J., Rao, R., & Pirolli, P. (1995). A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, (pp. 401–408). New York, NY, USA: ACM Press/Addison-Wesley Publishing Co.

Leung, Y. K., & Apperley, M. D. (1994). A review and taxonomy of distortion-oriented presentation techniques. *ACM Transactions on Computer-Human Interaction*, *1*(2), 126–160.

Mackinlay, J. D., Robertson, G. G., & Card, S. K. (1991). The perspective wall: Detail and context smoothly integrated. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, (pp. 173–176). New York, NY, USA: ACM Press.

Masui, T. (1998). Lensbar - visualization for browsing and filtering large lists of data. *infovis*, *00*, 113.

Norman, K. L. (1991). *The Psychology of Menu Selection: Designing Cognitive Control at the Human/Computer Interface*. Westport, CT, USA: Greenwood Publishing Group Inc.

Rao, R., & Card, S. K. (1994). The table lens: merging graphical and symbolic representations in an interactive focus + context visualization for tabular information. In *CHI '94: Proceedings of the SIGCHI conference on Human factors in computing systems*, (pp. 318–322). New York, NY, USA: ACM Press.

Robertson, G. G., & Mackinlay, J. D. (1993). The document lens. In *UIST '93: Proceedings of the 6th annual ACM symposium on User interface software and technology*, (pp. 101–108). New York, NY, USA: ACM Press.

Sarkar, M., & Brown, M. H. (1992). Graphical fisheye views of graphs. In *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*, (pp. 83–91). New York, NY, USA: ACM Press.

Schaffer, D., Zuo, Z., Greenberg, S., Bartram, L., Dill, J., Dubs, S., & Roseman, M. (1996). Navigating hierarchically clustered networks through fisheye and full-zoom methods. *ACM Trans. Comput.-Hum. Interact.*, *3*(2), 162–188.

Skopik, A., & Gutwin, C. (2005). Improving revisitation in fisheye views with visit wear. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, (pp. 771–780). New York, NY, USA: ACM Press.

Sutherland, I. E. (1964). Sketch pad a man-machine graphical communication system. In *DAC '64: Proceedings of the SHARE design automation workshop*, (pp. 6.329–6.346). New York, NY, USA: ACM Press.

Tominski, C., Abello, J., van Ham, F., & Schumann, H. (2006). Fisheye tree views and lenses for graph visualization. In *IV '06: Proceedings of the conference on Information Visualization*, (pp. 17–24). Washington, DC, USA: IEEE Computer Society.

# Abbildungsverzeichnis

# Tabellenverzeichnis