Teil II

Grundlagen auf der Seite der Maschine

6 Technische Rahmenbedingungen

Nachdem wir im ersten Teil dieses Buchs vieles über die Wahrnehmung und Informationsverarbeitung des Menschen gelernt haben, beginnt der zweite Teil mit einigen grundlegenden Gedanken, wie die Schnittstelle seitens der Maschine eigentlich beschaffen ist oder sein sollte. Beginnen wollen wir mit den technischen Rahmenbedingungen bei der Ein- und Ausgabe.

6.1 Visuelle Darstellung

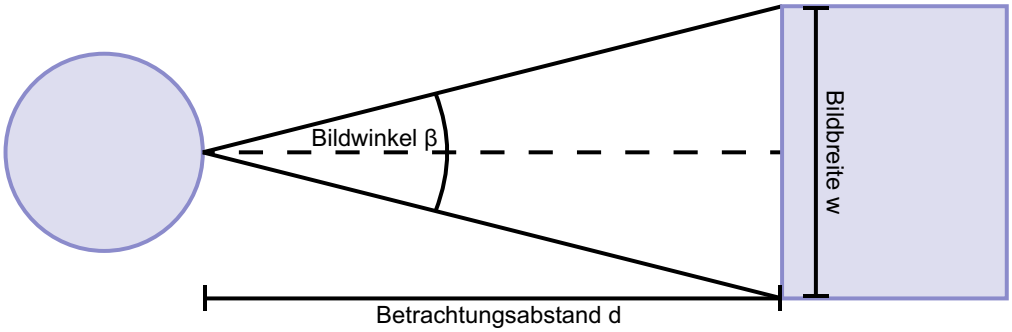
6.1.1 Räumliche Auflösung

In Abschnitt 2.1 wurde die prinzipielle Funktionsweise des menschlichen Auges erläutert und dessen räumliche Auflösung mit etwa einer Winkelminute beziffert. Wenn wir also zwei Punkte anschauen und der Winkel zwischen den beiden gedachten Blickstrahlen vom Auge zu diesen Punkten größer als $1/60°$ ist, dann können wir diese Punkte als getrennte Objekte wahrnehmen (volle Sehkraft und ausreichenden Kontrast vorausgesetzt). An einem gut ausgestatteten Bildschirmarbeitsplatz betrachten wir den Bildschirm aus einer Entfernung von mindestens einer **Bildschirmdiagonale**, meistens jedoch eher mehr. Dies erlaubt uns eine sehr einfache Abschätzung (sieh auch Abbildung 6.1): Entspricht der **Betrachtungsabstand** etwa der Bildschirmbreite (also etwas weniger als der Diagonalen), dann spannt der Bildschirm horizontal einen **Betrachtungswinkel** von $2 * \arctan \frac{1}{2}$ oder etwa $50°$ auf. Somit können wir maximal $60 * 50 = 3.000$ verschiedene Punkte unterscheiden. Eine horizontale **Bildschirmauflösung** von 3.000 Pixeln entspricht also etwa der Auflösungsgrenze des menschlichen Auges, wohlgemerkt an einem gut ausgestatteten Bildschirmarbeitsplatz mit großem Bildschirm und bei relativ naher Betrachtung.

Die gleiche Rechnung gilt für Druckerzeugnisse: Eine Din A4 Seite halten wir zum Lesen üblicherweise in einem Abstand, der mindestens ihrer längeren Kante entspricht, also etwa 30cm. Da die Winkelverhältnisse die selben sind, bedeutet dies, dass etwa 3.000 Pixel entlang dieser 30cm langen Kante der maximalen Auflösung des Auges entsprechen. Umgerechnet in die **Ortsauflösung** eines Druckers sind das $3000/30 = 100$ Pixel pro Zentimeter oder etwa 250Pixel pro Zoll (dpi)[1]. Die gängige Ortsauflösung von 300dpi für Druckerzeugnisse liegt also bei normalem Betrachtungsabstand etwas über der Auflösung des Auges.

Bei älteren Fernsehern (mit Seitenverhältnis 4:3) nimmt man einen Betrachtungsabstand an, der etwa der fünffachen Bilddiagonale entspricht, bei neueren **HDTV** Fernse-

[1] Die genauen Beziehungen zwischen dpi und ppi sind hier bewusst vereinfacht.



mmibuch.de/a/6.1

Abbildung 6.1: Berechnung der notwendigen Auflösung: Es gilt $\frac{w}{2} = d \tan \frac{\beta}{2}$ und damit $\beta = 2 \arctan \frac{w}{2d}$. Bei einer Auflösung von $\frac{1}{60}$ Grad werden also $60 * 2 * \arctan \frac{w}{2d}$ Pixel benötigt (alle Rechnungen in Grad).

hern wird als Betrachtungsabstand die doppelte Bilddiagonale zugrunde gelegt. Damit ergibt sich für HDTV eine benötigte Auflösung von $60 * 2 * \arctan \frac{1}{4} \approx 1.700$ Pixel (die Norm sieht 1.920 vor) und für Standard-TV $60 * 2 * \arctan \frac{1}{10} \approx 685$ Pixel (in der Praxis ca. 800). Wie man sieht, sind also die gängigen Auflösungen für Drucker und Bildschirme eng an die physiologischen Grenzen der visuellen Wahrnehmung angelehnt. Mehr Auflösung bringt natürlich Reserven für einen kürzeren Betrachtungsabstand, ist aber im normalen Betrieb nicht notwendig.

Betrachten wir den Bildschirm eines Smartphones, das eine Bildschirmkante von ca. 10cm aufweist in bequemer Armhaltung aus einem Abstand von ca. 50cm, so ergeben sich daraus ebenfalls $60 * 2 * \arctan \frac{5}{50} \approx 685$ Pixel, was etwa der Auflösung der ersten iPhone Generation entspricht. Betrachten wir ein Tablet mit 20cm Bildschirmkante aus der gleichen Entfernung, so ergeben sich $60 * 2 * \arctan \frac{10}{50} \approx 1350$ Pixel, was von der ersten iPad Generation nicht ganz erreicht wurde. Das **Head-Mounted Display** (**HMD**) *Oculus Rift*[2] weist einen horizontalen Bildwinkel von über 90 Grad bei einer horizontalen Auflösung von 1.280 Pixeln auf. Laut obiger Rechnung wären für einen solchen Bildwinkel jedoch $60 * 90 = 5.400$ Pixel notwendig, was erklärt, warum in diesem Display noch ganz deutlich Pixel zu sehen sind.

6.1.2 Zeitliche Auflösung

Die **zeitliche Auflösung** des menschlichen Auges liegt bei etwa 100ms. Dies bedeutet, dass wir aufeinanderfolgende Bilder bis zu einer Geschwindigkeit von etwa 10 Bildern pro Sekunde noch als getrennte Bilder wahrnehmen, darüber hinaus aber zunehmend als flüssiges **Bewegtbild**. Ab etwa 25 Bildern pro Sekunde sehen wir flüssige Bewegungen. Dies ist auch der Grund dafür, dass beim analogen Kinofilm eine Bildfolge von 24 bzw. 25 Bildern pro Sekunde aufgenommen wird. Bei der Darstellung am Fernsehgerät kam

[2] http://www.oculusvr.com

wegen der Bildröhre und ihrem sequenziellen Bildaufbau das zusätzliche Problem des Bildflimmerns hinzu, welches bei 25 Bildern pro Sekunde noch wahrnehmbar war. Aus diesem Grunde werden dort 50 sogenannte Halbbilder pro Sekunde angezeigt, was das Flimmern fast unmerklich werden lässt, die Informationsdichte jedoch nicht erhöht.

Zeitliche Verzögerungen, beispielsweise zwischen akustischem Sprachsignal und Mundbewegungen eines Fernsehsprechers, werden ebenfalls ab etwa 100ms erkennbar. Verzögerungen in einer Benutzerschnittstelle führen dazu, dass die Wahrnehmung von **Kausalität** leidet. Tätigt man eine Eingabe und die Rückmeldung in Form einer Ausgabe erfolgt innerhalb von 100ms, so wird die Ausgabe als direkt mit der Eingabe verknüpft empfunden. Erfolgt die Reaktion langsamer, aber immer noch innerhalb etwa einer Sekunde, so wird die Ausgabe immer noch als kausale Folge der Eingabe wahrgenommen. Bei Reaktionszeiten über einer Sekunde nimmt die Wahrnehmung der Kausalität immer weiter ab, und bei vielen Sekunden Reaktionszeit ohne zwischenzeitiges „Lebenszeichen“ stellen wir keinen kausalen Zusammenhang mehr zwischen Ein- und Ausgabe her. In manchen Fällen sind längere Zeiten zwischen Ein- und Ausgabe unumgänglich, manchmal sogar unvorhersehbar. Dies ist beispielsweise der Fall bei Webseiten, bei denen die Ladezeit einer neuen Seite von der Last des Servers und von vielen Netzwerkknoten dazwischen abhängt. In solchen Fällen muss der Benutzer durch irgendwie geartete Fortschrittsmeldungen oder andere „Lebenszeichen“ informiert werden (siehe auch Abschnitt 7.5 und Abbildung 7.6 auf Seite 74).

6.1.3 Darstellung von Farbe und Helligkeit

Bei der Darstellung der Funktionsweise des Auges in Abschnitt 2.1 wurde die Farbwahrnehmung mittels drei verschiedener Arten von **Zäpfchen** erklärt. Diese drei Arten von Rezeptoren werden durch Licht in den drei additiven **Grundfarben** Rot, Grün und Blau angeregt, und jede vom Menschen wahrnehmbare Farbe lässt sich somit als Linearkombination der Grundfarben im dreidimensionalen **RGB-Farbraum** darstellen. Diese drei Grundfarben werden technisch auch als **Farbkanäle** bezeichnet. Innerhalb eines Farbkanals kann der Mensch etwa 60 verschiedene Abstufungen wahrnehmen (vgl. Abschnitt 2.1). Damit würden etwa 6 Bit Auflösung (**Farbtiefe**) pro Kanal ausreichen, um flüssige Farbverläufe darzustellen. Da Computerspeicher jedoch in kleinsten Einheiten von 8 Bit organisiert sind, hat sich eine gängige Farbtiefe von $3 * 8 = 24$ Bit eingebürgert, die die Farbauflösung des menschlichen Auges etwas übersteigt.

Monitore für Bildschirmarbeitsplätze weisen mittlerweile einen **Kontrastumfang** von mindestens 1:1.000 auf. Das bedeutet, dass ein weißer Bildpunkt mindestens 1.000 mal so hell wie ein schwarzer Bildpunkt dargestellt wird. Dies deckt sich mit dem Kontrastumfang des menschlichen Auges ohne Akkommodation (vgl. Abschnitt 2.1) und kann mit der gängigen Farbtiefe von 24 Bit problemlos angesteuert werden. Spezialisierte Monitore für Darstellungen mit hohem Kontrastumfang, sogenannte **HDR** (High Dynamic Range) Monitore können bereits Kontrastumfänge von 200.000:1 bis angeblich 1.000.000:1 darstellen. Dies entspricht dem Dynamikumfang des menschlichen Auges mit Akkommodation und bedeutet, dass wir diese Kontraste immer nur lokal (also nicht über den gesamten Bildschirm hinweg) sehen können. HDR Darstellung mag zwar beeindruckend sein, hat deshalb aber nur wenige konkrete Anwendungen. Für die meisten Situationen ist ein Kontrastumfang von 1.000:1 bis 5:000:1 mehr als ausreichend.

Alle diese Betrachtungen berücksichtigen jedoch noch kein Umgebungslicht, gehen also vom Display in einem dunklen Raum aus. Fällt hingegen Umgebungslicht auf den Bildschirm, dann wird der Kontrastumfang drastisch reduziert. Dies ist beispielsweise der Fall, wenn Smartphones oder Digitalkameras bei Sonnenlicht betrieben werden. Nehmen wir zuerst ein harmloses Beispiel: Wenn ein Laptop-Display eine Helligkeit von $200\frac{cd}{m^2}$ besitzt, sich in einem Raum mit einer Raumhelligkeit von 200 Lux (beispielsweise einem mäßig beleuchteten Zimmer) befindet, und an seiner Oberfläche etwa 1% des Umgebungslichtes reflektiert, dann wird seine dunkelste Stelle (schwarzes Pixel) durch das Umgebungslicht mit etwa $2\frac{cd}{m^2}$ leuchten, und das hellste Pixel mit etwa $202\frac{cd}{m^2}$. Es verbleibt also ein Kontrastumfang von etwa 100:1, womit das Display immer noch gut lesbar ist. Bringt man das gleiche Display in helles Sonnenlicht mit 100.000 Lux, dann hellt das Umgebungslicht die dunkelste Stelle auf $1.000\frac{cd}{m^2}$ und die hellste Stelle auf $1.200\frac{cd}{m^2}$ auf, so dass der sichtbare Kontrastumfang auf 1,2:1 reduziert wird, was das Display praktisch unlesbar macht. Aus diesem Grund funktionieren selbstleuchtende Displays sehr schlecht bei hellem Sonnenlicht und neben einer hohen maximalen Helligkeit ist es wichtig, dass möglichst wenig Umgebungslicht vom Display reflektiert wird. Das ist der Grund, warum glänzende Monitore ein höheres verbleibendes Kontrastverhältnis erreichen als die eigentlich angenehmeren matten Monitor-Oberflächen.

6.2 Akustische Darstellung

Der menschliche Hörsinn (vgl. Abschnitt 2.2.1) hat einen Dynamikumfang von etwa 120 dB. Dabei entsprechen jeweils 6dB einer Verdopplung der Signalstärke. Das bedeutet, dass zwischen dem leisesten wahrnehmbaren Geräusch, der sogenannten **Hörschwelle**, und dem lautesten noch als Geräusch wahrnehmbaren Reiz, der sogenannten **Schmerzgrenze** etwa 20 Verdopplungen liegen und somit ein mit 20 Bit Auflösung digitalisiertes Signal diesen gesamten Bereich abdecken kann. Praktisch genutzt wird davon im Alltag jedoch nur ein viel kleinerer Bereich von etwa 20-90dB, was etwa 12 Bit entspricht. Aus dieser Überlegung ergibt sich, dass die in der CD-Norm festgelegten 16 Bit Auflösung für die Signalstärke mehr als ausreichend sind, um den gesamten sinnvoll nutzbaren Lautstärkebereich abzudecken. Für Sprache allein (30-70dB) genügen sogar 8 Bit, wie es auch im ISDN Standard für digitale Sprachübertragung verwendet wird.

Neben der Signalauflösung (**Quantisierung**) interessiert natürlich auch die benötigte zeitliche Auflösung (**Diskretisierung**). Diese lässt sich einfach mittels des **Nyquist-Theorems** abschätzen: Die höchste Frequenz, die das (junge und gesunde) menschliche Ohr hört, sind etwa 20.000Hz. Um ein Signal mit dieser Frequenz fehlerfrei zu digitalisieren, ist eine Abtastung mit mehr als der doppelten Abtastrate nötig. Der CD-Standard sieht daher eine Abtastrate von 44.100Hz vor. Zusammen mit der Signalauflösung von 16 Bit deckt die Audio-CD somit physiologisch gesehen die zeitliche und dynamische Auflösung des menschlichen Gehörs ab.

Zeitunterschiede zwischen den beiden Ohren werden einerseits zur räumlichen Ortung akustischer Ereignisse verwendet. Zeitlich verzögerte und abgeschwächte Signale bilden aber beispielsweise auch den Klang von Räumen ab, da dort durch Schallreflexionen genau solche verzögerten und abgeschwächten Signale entstehen. Sehr kurze Verzöge-

rungen bis etwa 35ms integriert das Ohr dabei zu einem Signal. Zwischen 50ms und 80ms löst sich dieser Zusammenhang immer weiter auf und über etwa 80ms wird das verzögerte Signal mehr und mehr als diskretes Echo wahrgenommen [32]. Interessanterweise liegt diese Grenze in der gleichen Größenordnung wie die maximal zulässige Verzögerung zwischen Ton und Bild, die Grenze des Übergangs vom Einzelbild zum Bewegtbild, und die Grenze bei der Wahrnehmung eines direkten Zusammenhangs zwischen Aktion und Reaktion. Eine Verzögerung von 100ms kann also in allen diesen Bereichen als kritische Grenze angesehen werden, was somit als allgemeine, wenn auch ganz grobe Richtlinie für den Umgang mit Timing-Fragen angesehen werden kann.

6.3 Moore's Law

In den voran gegangenen Abschnitten wurde deutlich, dass sich verbreitete Werte für die zeitliche und räumliche Auflösung von Ausgabemedien an der Auflösung der entsprechenden menschlichen Sinne orientieren, sofern dies technisch möglich ist. Die menschlichen Sinne und damit besagte Kennwerte verändern sich lediglich in evolutionären Zeiträumen, bleiben jedoch in den von uns überschaubaren Zeiträumen gleich. Gleiches gilt für die kognitiven Fähigkeiten des Menschen. Anders verhält es sich jedoch mit der Rechenleistung und Speicherkapazität von Computersystemen. Der Amerikaner **Gordon E. Moore** formulierte das später nach ihm benannte **Mooresche Gesetz**, welches besagt, dass die Rechenleistung und Speicherkapazität von Silizium-Chips sich ungefähr alle 18 Monate verdoppeln. Je nach Quelle und Anwendungsgebiet finden sich in der Literatur auch Werte zwischen 12 und 24 Monaten. Allen Quellen gemeinsam ist jedoch die Tatsache, dass Rechenleistung und Speicherkapazität exponentiell wachsen, und zwar innerhalb durchaus überschaubarer Zeiträume.

Obwohl diesem Wachstum irgendwann physikalische Grenzen gesetzt sind, können wir doch noch für viele Jahre mit einem solchen Wachstum rechnen. Dies bedeutet, dass Rechengeschwindigkeit und Speicherplatz (mit Ausnahme algorithmisch komplexer Probleme) nicht mehr wirklich die bestimmenden oder begrenzenden Faktoren beim Bau interaktiver Systeme sind. Selbst wenn während des Entwurfs eines interaktiven Systems noch fraglich ist, ob die benötigte Geschwindigkeit oder Kapazität zur Verfügung stehen, können wir davon ausgehen, dass die technische Entwicklung dies in den allermeisten Fällen für uns lösen wird.

Umgekehrt bedeutet dies aber auch eine ständig wachsende Herausforderung beim Entwurf der Benutzerschnittstelle, da diese dem nicht mitwachsenden menschlichen Benutzer ja immer mehr Rechenleistung und Informationsmenge vermitteln muss. Dies gilt umso mehr, als wir bei der visuellen Darstellung - wie oben beschrieben - bereits an den physiologischen Grenzen der menschlichen Wahrnehmung angelangt sind. Rein *technisch* ist daher in der Schnittstelle keine allzu große Steigerung mehr möglich. Stattdessen wächst die Herausforderung, *konzeptuell* noch etwas an den bestehenden Schnittstellen zu verbessern. Darauf werden die nächsten drei Kapitel eingehen.

Übungsaufgaben:

1. Eine völlig theoretische Überlegung: Sie sollen als neuer Verantwortlicher des Visualisierungszentrums ein CAVE (CAVE Automatic Virtual Environment) aufbauen und müssen entscheiden, welche Displays oder Projektoren Sie kaufen. Mit Ihrem Basiswissen aus MMI wollen Sie zunächst berechnen, wie viele Pixel das menschliche Auge auflösen würde. Dazu nehmen Sie an, dass das CAVE die Form eines Würfels mit 4m Kantenlänge hat und der Benutzer in der Mitte steht. Wie viele Pixel Breite müsste Ihr Display demnach für eine Wand haben? Kommt eine der gängigen Anzeigenormen in diese Größenordnung? Wie viele Displays dieser Norm benötigen Sie ggf. für ein 5-seitiges CAVE, und wie groß sind die Pixel, wenn man sie auf dem Display selbst nachmisst.

2. Sie möchten in einer Benutzerstudie herausfinden, wie gut Text auf einem Smartphone in verschiedenen Situationen lesbar ist. Dafür lassen Sie Leute einen Text auf dem gleichen Bildschirm lesen, einmal am Schreibtisch ihres Büros, und einmal beim Gehen über den Uni-Campus. Welcher Faktor außer der Bewegung könnte die Lesbarkeit noch beeinflussen, und wie könnten Sie Ihre Studie abändern, um diesen Faktor auszuschalten?

7 Grundregeln für die UI Gestaltung

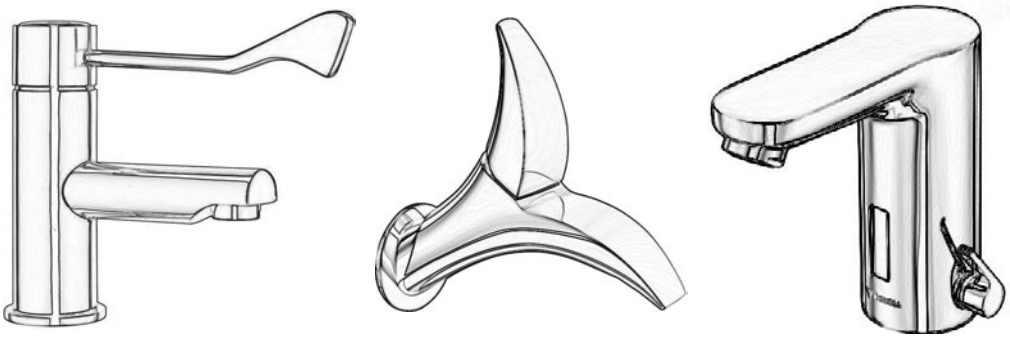
Bei der Gestaltung von Benutzerschnittstellen gibt es eine Reihe von grundlegenden Prinzipien, deren Beachtung zu effizienteren, benutzbareren und verständlicheren Ergebnissen führt. Dieses Kapitel führt die wichtigsten dieser Prinzipien auf und erläutert sie jeweils an Beispielen aus verschiedenen Interaktions-Kontexten.

7.1 Affordances

Das Konzept der **Affordance** stammt ursprünglich aus der Kognitionspsychologie und bezeichnet dort die Eigenschaft, eine bestimmte Handlung mit einem Objekt oder an einem Ort zu ermöglichen. **Don Norman** [85] übertrug es auf den Bereich der Mensch-Maschine-Interaktion, wo es im engeren Sinne bedeutet, dass ein Objekt des Alltags uns bestimmte Aktionen mit ihm ermöglicht oder uns sogar dazu einlädt. Im Deutschen wird der Begriff gelegentlich mit **Angebotscharakter** übersetzt, was sich jedoch unseres Erachtens nicht in der Breite durchgesetzt hat. Wir verwenden daher in diesem Abschnitt den englischen Begriff Affordance.

Die von Norman genannten Beispiele für Affordances sind physikalischer Natur: ein runder Knopf hat die Affordance, gedreht oder gedrückt zu werden. Je nach seiner physikalischen Ausprägung ermöglich er evtl. noch, gezogen zu werden. Er lädt uns jedoch nicht dazu ein, ihn seitlich zu kippen oder zu verschieben. Ein typischer Wasserhahn lädt uns ganz klar zum Drehen ein. Ein Hebel hingegen bietet uns an, ihn in jede beliebige Richtung zu bewegen. Abbildung 7.1 zeigt verschiedene Ausführungen von Wasserhähnen, die uns mittels ihrer verschiedenen Affordances besser oder schlechter mitteilen, wie sie zu bedienen sein könnten. Abbildung 7.4 auf Seite 71 Rechts zeigt ein Beispiel für eine widersprüchliche Affordance.

Übertragen auf den Computer bedeutet dies, dass unsere grafischen Benutzerschnittstellen immer visuell klar erkennen lassen sollten, welche Operationen möglich sind. Ein einfaches Beispiel hierfür sind die als dreidimensional angedeuteten Schaltflächen (**Buttons**) in den uns vertrauten grafischen Benutzerschnittstellen. Die Tatsache, dass sie plastisch aus der Bildschirmebene herauszukommen scheinen macht sie nicht einfach nur hübscher. Sie sendet dem Benutzer ein wichtiges Signal, indem sie ihn an die Druckknöpfe der physikalischen Welt erinnert. Auch diese stehen plastisch aus ihrem Umfeld hervor und laden dazu ein, gedrückt zu werden. Die angedeutete plastische Form unterscheidet den beschrifteten, aber interaktiven Button vom nicht interaktiven **Label** oder **Textfeld**, das womöglich den gleichen Text enthält, aber nicht zum Drücken einlädt.



mmibuch.de/a/7.1

Abbildung 7.1: Verschiedene Wasserhähne mit verschiedenen Affordances: Bei manchen ist die Bedienung leichter zu erkennen als bei anderen.

Neben diesen physikalischen Affordances gibt es im erweiterten Verständnis des Begriffes auch andere Arten von Affordances: Als **social Affordance** bezeichnet man beispielsweise die Tatsache, dass ein bestimmter Ort oder Kontext (z.B. ein Café oder ein Fitnesstudio) die Möglichkeit zu bestimmten sozialen Aktivitäten, wie z.B. Kommunikation oder Selbstdarstellung bietet. Der derzeitige Boom im Bereich sozialer Medien zeigt, dass auch digitale Orte solche sozialen Affordances bieten können.

7.2 Constraints

Die Affordances der in Abbildung 7.1 gezeigten Wasserhähne alleine reichen bei genauerem Nachdenken nicht aus, um deren Bedienung vollständig zu erschließen. Der Hebel im linken Bild kann prinzipiell in alle Richtungen bewegt werden. Wenn wir jedoch aus dem normalen (verschlossenen) Zustand heraus versuchen, ihn herunter zu drücken, spüren wir einen physikalischen Widerstand. Der Hebel lässt sich nicht herunterdrücken. Diese Einschränkung der Interaktionsmöglichkeiten wird als **physikalisches Constraint** bezeichnet. In vertikaler Richtung lässt sich der Hebel nur anheben, wodurch Wasser fließt. Die Affordances reichen jedoch auch zusammen mit diesem physikalischen Constraint immer noch nicht aus, die Bedienung völlig unmissverständlich zu vermitteln. Was bleibt, ist die Unklarheit, was ein Schieben des Hebels nach rechts oder links bewirkt. Von der Funktionalität her betrachtet bleibt nur noch die Wassertemperatur übrig. Hier haben wir in unserem Kulturkreis vom Kindesalter an gelernt, dass links warm und rechts kalt bedeutet. Diese Zuordnung bezeichnet man daher als **kulturelles Constraint**. Andere Beispiele für kulturelle Constraints sind beispielsweise fest mit Bedeutung belegte Farben (rot = stop, grün = OK) oder Symbole. Schließlich gibt es auch noch **logische Constraints**, wie z.B. die Tatsache, dass es keinen Sinn ergibt, ein Musikstück gleichzeitig vorwärts und rückwärts abzuspielen, oder dass ein Licht nicht gleichzeitig an- und ausgeschaltet sein kann.

Abbildung 7.2: Verschiedene Mappings von Kochplatten zu Reglern: Das räumliche Mapping von links nach rechts passt, aber im linken Beispiel bleibt unklar, welche Regler zu den vorderen bzw. hinteren Platten gehören.

mmibuch.de/a/7.2

7.3 Mappings

Eine wichtige Möglichkeit für die Strukturierung von Benutzerschnittstellen ist die Zuordnung von Bedienelementen zu Objekten in der physikalischen Welt. Eine solche Zuordnung heißt im Englischen **Mapping**. Die einfachste Form sind direkte räumliche Mappings. Hierzu folgendes einfache Beispiel: Abbildung 7.2 zeigt zwei autarke Gaskochfelder des gleichen Herstellers. Die Regler für die einzelnen Flammen des Kochfeldes wurden jedoch in beiden Fällen unterschiedlich angeordnet: In der linken Variante befinden sich die Regler in einer Reihe, die Kochflammen jedoch in einem Rechteck. Insbesondere befinden sich die beiden linken und die beiden rechten Flammen jeweils genau hintereinander, die zugehörigen Köpfe jedoch nebeneinander. Welcher Knopf bedient also welche Flamme? Um das herauszufinden, muss der Benutzer die Beschriftung der Knöpfe lesen, die die Zuordnung jeweils über ein Symbol andeuten. In der rechten Variante ist diese Mehrdeutigkeit aufgehoben: die hinteren seitlichen Flammen sind den hinteren Knöpfen zugeordnet, die vorderen den vorderen. Eine weitere Beschriftung ist unnötig und fehlt folgerichtig auch. Diese Variante besitzt ein klares räumliches **Mapping** zwischen Bedienelement und gesteuerter Funktion.

Ein weiteres Beispiel für ein räumliches Mapping ist die Anordnung der Kontrolltasten für die Medienwiedergabe. Diese geht zurück auf die Anordnung der Steuertasten beim Tonbandgerät (Abbildung 7.3 links). Bei solchen Geräten wurde das durchlaufende Band von der linken Spule ab- und auf die rechte Spule aufgewickelt. Da es keinen klaren technischen Grund dafür gibt, warum dies nicht auch anders herum sein könnte, hat man sich damals vermutlich an der bei uns vorherrschenden Leserichtung von links nach rechts orientiert, also an einem **kulturellen Constraint**. Die rechte Taste (Vorspultaste) bewirkte damit eine Bewegung des Bandes nach rechts, die linke Taste nach links. Dazwischen befanden sich Pause und Start. Es gab also ein räumliches Mapping zwischen den Kontrolltasten und den damit ausgelösten Bewegungsrichtungen des Bandes. Die so entstandene Anordnung hat sich über verschiedene Gerätegenerationen und Technologien (Musikkassette, DAT, MD, CD, DVD, BlueRay, MP3) hinweg gehal-

mmibuch.de/a/7.3

Abbildung 7.3: Räumliches Mapping: Das Tonband lief bei der Wiedergabe von links nach rechts am Tonkopf vorbei. Die Anordnung der Knöpfe für Rückspulen, Vorspulen und Wiedergabe orientiert sich bis heute an dieser Bewegungsrichtung.

ten, obwohl die ursprüngliche physikalische Entsprechung schon früh verschwunden war. Ein Grund dafür könnte auch das immer noch eingehaltene **kulturelle Constraint** der Leserichtung sein.

Als drittes Beispiel wollen wir uns die Steuertasten eines Aufzuges anschauen. Diese sind üblicherweise übereinander angeordnet, wobei die oberste Taste dem obersten Stockwerk entspricht, die unterste dem untersten Stockwerk, und eine speziell markierte Taste oft das Erdgeschoss oder die Eingangsebene des Gebäudes bezeichnet. Dieses räumliche **Mapping** ist an Verständlichkeit nicht zu übertreffen: Oben drücken bedeutet nach oben fahren, unten drücken bedeutet nach unten fahren. Ein solcher Aufzug ist auch von Kindern zu bedienen, die nicht lesen können (vorausgesetzt sie erreichen die Tasten und können diese zumindest abzählen). Man findet jedoch auch horizontal angeordnete Steuertasten, bei denen dann zumeist die Stockwerke von links nach rechts hoch gezählt werden. Hier ist schon ein Nachdenken und Lesen der Beschriftungen nötig.

Abbildung 7.4 schließlich zeigt ein vom Autor gefundenes Gegenbeispiel, das alle Regeln eines logischen Mappings verletzt. Weder sind die Stockwerke zeilenweise geordnet, noch spaltenweise. Hinzu kommt, dass der fragliche Aufzug zwei Türen auf beiden Seiten hatte, von denen sich in jedem Stockwerk nur eine öffnete. Die Zuordnung der Seiten entspricht jedoch auch nicht diesen Türen. Das fehlende 1. Stockwerk existierte an dieser Stelle des Gebäudes tatsächlich nicht, da die Eingangshalle 2 Geschosse hoch war. Das unbeschriftete oberste Stockwerk war nur für Personal und mit einem passenden Schlüssel oder Chip auszuwählen. Hier bleibt keine andere Erklärung, als dass dem Aufzugstechniker einfach die (vermutlich freie) Anordnung der Tasten im Detail egal war, zumal der benachbarte Aufzug ein leicht anderes Layout aufwies. Bewusste oder unbewusste Verletzung von Mappings machen Benutzerschnittstellen schwieriger zu bedienen und führen zu vermehrten Eingabefehlern. Wo immer sich ein natürliches **Mapping** finden lässt, sollte dies auch verwendet werden.

Abbildung 7.4: Links: Eine Aufzugssteuerung, die in eklatanter Weise sämtliche denkbaren logisch erklärbaren Mappings verletzt. Rechts: eine Tür mit sehr fragwürdigen Affordances.

mmibuch.de/a/7.4

7.4 Konsistenz und Vorhersagbarkeit

Neben Affordances, Constraints und Mappings ist eine weitere wichtige Eigenschaft von Benutzerschnittstellen deren **Konsistenz**. Sie erlaubt uns, das Vorhandensein und die Funktion bestimmter Bedienelemente vorherzusagen, sowie sie überhaupt in verschiedenen Kontexten zuverlässig wiederzufinden. In Kapitel 14 werden wir sehen, dass diese **Vorhersagbarkeit** sogar ein psychologisches Grundbedürfnis des Menschen anspricht.

In Anlehnung an die Sprachwissenschaft unterscheidet man verschiedene Ebenen der Konsistenz: **Syntaktische Konsistenz** bezeichnet die Tatsache, dass Dinge immer strukturell (also syntaktisch) gleich funktionieren. Ein Beispiel dafür ist, dass Buttons mit gleicher Funktion immer an der gleichen Stelle sind, z.B. der *Back*-Button in einer mobilen Applikation oder die Suchfunktion auf Webseiten (zumeist oben rechts). **Semantische Konsistenz** bedeutet umgekehrt, dass ein bestimmtes Kontrollelement, das in verschiedenen Kontexten auftritt, immer auch die gleiche Funktion (also Bedeutung oder Semantik) besitzt. Ein Beispiel hierfür ist, dass der *Back*-Button überall genau einen Bearbeitungsschritt zurück geht und die *Undo*-Funktion auch überall existiert. Se-

mantische Konsistenz bedeutet beispielsweise auch die immer gleiche Verwendung von Farben, z.B. Grün für *OK* und Rot für *Nein*. Ein Gegenbeispiel ist die Tatsache, dass das Bewegen einer Datei von einem Ordner in einen anderen in einer grafischen Benutzerschnittstelle diese entweder verschiebt, wenn beide Ordner auf dem selben physikalischen Laufwerk sind, oder sie kopiert, wenn die beiden Ordner sich auf verschiedenen physikalischen Laufwerken befinden. Dieses Verhalten ist inkonsistent und führt dazu, dass wir uns über technische Gegebenheiten Gedanken machen müssen, um die Auswirkungen einer Operation korrekt vorherzusagen (siehe hierzu auch Kapitel 5). **Terminologische Konsistenz** schließlich bedeutet, dass die gleiche Funktion immer auch mit dem gleichen Begriff benannt wird. Die Operation *Einfügen* heisst beispielsweise überall genau so, und nicht etwa plötzlich *Einsetzen* oder *Hinzufügen*. Das Disketten-Icon hat sich als universelles Symbol für die Funktion *Abspeichern* gehalten und wird auch immer noch konsistent dafür verwendet, obwohl die technische Entsprechung, nämlich die 3 1/2 Zoll Diskette längst verschwunden ist.

Man unterscheidet außerdem zwischen **innerer Konsistenz** und **äußerer Konsistenz**. Die innere Konsistenz bezeichnet dabei die Konsistenz innerhalb einer einzelnen Anwendung (z.B. immer gleicher *Back*-Button) und die äußere Konsistenz gilt über verschiedene Anwendungen und womöglich auch Geräte hinweg. Hierbei gibt es verschiedene Reichweiten der äußeren Konsistenz. Meist werden Konventionen innerhalb eines Betriebssystems oder innerhalb einer Geräteklasse eingehalten, nicht jedoch darüber hinweg. Ein frappierendes Beispiel dafür sind die Nummernblöcke unserer Computer-Tastaturen und Mobiltelefone. Ohne gleich weiterzulesen beantworten Sie bitte spontan die Frage: wo befindet sich die Null? Wo die Eins?

Abbildung 7.5 zeigt links die Tastatur eines Telefons, auf der die Null unten und die Eins oben links ist. Dagegen befindet sich die Eins im Zehnerblock einer Computertastatur unten links. Die beiden Layouts erfüllen zwar beide die gleiche grundlegende Funktion, kommen jedoch aus verschiedenen technologischen Welten, nämlich der Welt der Großrechner und der Welt der Telefone. Taschenrechner verwenden das Computer-Layout, Mobiltelefone (auch in Zeiten des Touchscreens) das Telefon-Layout. In beiden Welten sind die Gründe für eine interne Konsistenz mit dem historisch gewachsenen Standard stärker als mögliche Gründe für einen Wechsel zur externen Konsistenz.

Das Beispiel zeigt gleichzeitig ein weiteres Risiko bei fehlender Konsistenz: Menschen merken sich Zahlenkombinationen wie PINs oder Telefonnummern oft grafisch als Form auf der Tastatur. Die PIN 2589 ergibt beispielsweise auf der Zehnertastatur der meisten Geldautomaten die Form eines großen L. Hat man sich diese Form gemerkt und gibt die PIN mit wachsender Übung blind ein, dann kann es passieren, dass man an einem Geldautomaten mit anderem Layout plötzlich scheitert ohne zu wissen warum.

7.5 Feedback

Eine wichtige Anforderung bei der Gestaltung verständlicher und bedienbarer Benutzerschnittstellen ist die Bereitstellung von passendem **Feedback**. Wenn wir eine Operation ausgelöst haben, wollen wir in irgendeiner Form auch eine Rückmeldung, dass diese tatsächlich ausgeführt wurde. Bei einem Lichtschalter besteht diese Rückmeldung ganz

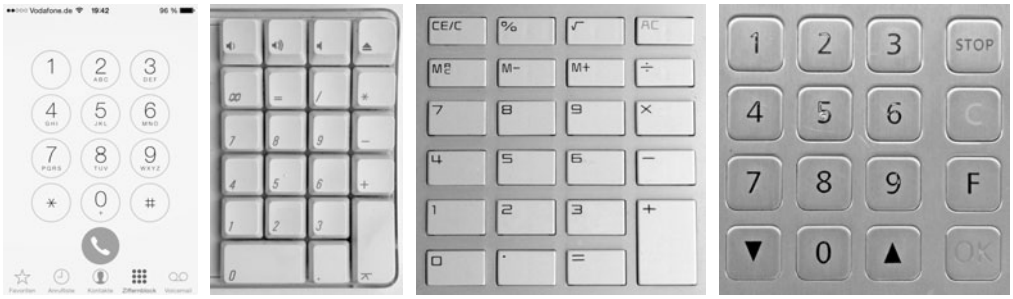


Abbildung 7.5: Inkonsistenz zwischen häufig verwendeten Arten von Zehnerblocks. Die Bilder zeigen von links nach rechts die Tastaturen eines Handys, eines Desktop-Computers, eines Taschenrechners und eines Geldautomaten.

mmibuch.de/a/7.5

einfach darin, dass das Licht an- oder ausgeht. Wenn wir ein Fenster am Bildschirm schließen indem wir die entsprechende Schaltfläche anklicken, dann ist die Rückmeldung das Verschwinden des Fensters. Feedback kann aber nicht nur visuell sein, sondern auch in verschiedenen Wahrnehmungskanälen vermittelt werden: Akustisches Feedback sind beispielsweise Signaltöne am Computer oder die Tastaturtöne bei Mobiltelefonen. Solche akustischen Signale ersetzen beispielsweise das fehlende haptische Feedback bei Bildschirmtastaturen (fehlender Tasten-Klick) oder helfen uns, auch ohne genauen Blick auf ein kleines Display eine Telefonnummer fehlerfrei einzutippen. Haptisches Feedback bekommen wir beispielsweise vom ABS unseres Autos, indem das Bremspedal vibriert, sobald die ABS Funktion eingreift. Diese ursprünglich technisch begründete Form des Feedback wird mittlerweile auch dann künstlich erzeugt, wenn das Pedal selbst keine mechanische Verbindung mehr zum entsprechenden Regelsystem hat.

Eine wesentliche Funktion des Feedback ist, dass wir verstehen, was das System oder Gerät gerade tut, bzw. in welchem Zustand es sich befindet. Eine Sanduhr oder ein rotierender Ball als Cursor vermitteln beispielsweise, dass das zugehörige Computersystem nicht etwa abgestürzt ist, sondern gerade eine Operation ausführt, die noch länger dauert (siehe Abbildung 7.6). Ein aussagekräftigeres Feedback wäre in dieser Situation z.B. ein Fortschrittsbalken, der uns abschätzen lässt, wie viel Wartezeit noch vor uns liegt. Andere Formen des Feedback sind angezeigte Hinweise oder Fehlermeldungen. Dabei gilt die Grundregel, immer verständliches und im Kontext sinnvolles Feedback anzubieten, sowie – falls möglich – einen Hinweis, wie man den Fehler beheben könnte. Eine Meldung der Art *Fehler -1* war vermutlich der wenigste Aufwand für den Entwickler, ruft aber beim Benutzer lediglich Frustration hervor.

Damit wir das Feedback mit der auslösenden Operation verbinden, müssen bestimmte Zeitlimits eingehalten werden. Einen direkten kausalen Zusammenhang nehmen wir beispielsweise bis zu einer Verzögerung von 100ms wahr. Längere Verzögerungen, die jedoch noch im Bereich bis etwa eine Sekunde liegen, führen zwar zu einer spürbaren Unterbrechung des Ablaufs, sind jedoch noch zuzuordnen. Dauert das Feedback meh-

mmibuch.de/a/7.6

Abbildung 7.6: Verschiedene Formen von Feedback, die uns in einer grafischen Benutzerschnittstelle mitteilen, dass wir warten müssen: Links die Sanduhr aus frühen Windows Versionen, in der Mitte der *Beachball* aus OSX, Rechts ein Fortschrittsbalken.

rere Sekunden oder gar noch länger, dann nehmen wir keinen direkten Zusammenhang mehr zwischen Operation und Rückmeldung wahr. Die Rückmeldung wird für sich interpretiert und die ursprüngliche Operation bleibt scheinbar ohne Wirkung. Grundregel ist es, die Antwortzeiten zu minimieren (siehe auch Abschnitt 6.1.2).

7.6 Fehlertoleranz und Fehlervermeidung

In Abschnitt 5.3.3 wurde **Murphys Gesetz** erläutert. Demnach müssen wir als Entwickler eines interaktiven Systems oder Gerätes immer fest damit rechnen, dass der Benutzer jeden möglichen **Fehler** irgendwann einmal machen wird. Dem können wir auf zwei verschiedene Arten begegnen: zunächst sollten unsere Systeme immer so robust und **fehlertolerant** wie möglich sein, also mit Fehlern des Benutzers sinnvoll umgehen und nicht etwa einfach abstürzen. Außerdem können wir durch bestimmte Techniken zur **Fehlervermeidung** beitragen und damit gleichzeitig sowohl die Frustration des Benutzers reduzieren, als auch den Aufwand, den wir selbst in die **Fehlertoleranz** investieren müssen.

Zur Fehlertoleranz gehört zunächst die konsequente Validierung aller Eingaben. Bei Eingabemasken für Dinge wie Datum oder Kreditkarten-Information können dies einfache Tests sein (legales Kalenderdatum? Richtige Stellenzahl nach Entfernen aller Leerzeichen und Bindestriche?). Bei komplexeren Eingaben, wie z.B. dem Quelltext einer Webseite kann es die komplette Validierung des HTML-Codes bedeuten. Wird ein Fehler gefunden, so sollte dieser möglichst genau lokalisiert werden und idealerweise auch gleich ein Korrekturvorschlag angeboten werden. Zeitgemäße Entwicklungsumgebungen bieten solche Funktionalität bereits standardmäßig an. Das Verhindern einer syntaktisch oder gar semantisch falschen Eingabe baut so etwas wie ein **logisches Constraint** auf und trägt damit zur **Fehlervermeidung** bei.

Lässt sich die semantische Fehlerfreiheit nicht so einfach bei der Eingabe schon überprüfen (z.B. Eingabe eines syntaktisch richtigen Servernamens, zu dem aber kein Server

existiert), dann muss in späteren Verarbeitungsschritten sichergestellt werden, dass die entsprechenden Fehler sinnvoll abgefangen werden. Dies kann z.B. durch eine aussagekräftige Fehlermeldung (Server mit dem Namen xyz nicht ansprechbar) und einen Korrekturdialog erfolgen. Einfrieren oder Abstürzen des Programms sind offensichtlich inakzeptable Verhaltensweisen. Dies bedeutet für den Entwickler, bereits während der Programmierung über mögliche Arten von Eingabefehlern sowie deren sinnvolle Behandlung nachzudenken.

Eine sehr wichtige Funktion im Zusammenhang mit menschlichen Bedienfehlern ist auch die **Undo** Funktion (engl: *to undo sth.* = etwas rückgängig machen). Eine Funktion, die es ermöglicht, jede gemachte Eingabe rückgängig zu machen, erlaubt dem Benutzer automatisch, auch jeden gemachten Fehler zurückzunehmen und zu korrigieren. Das sichere Gefühl aber, jeden Fehler rückgängig machen zu können, ermutigt den Benutzer zu experimentieren. Wenn das Resultat einer Eingabe oder einer Operation unklar ist, kann man sie einfach ausprobieren, denn wenn das Ergebnis nicht das gewünschte war, kann man sie jederzeit zurücknehmen und etwas anderes ausprobieren. Dies ist der Grund für sehr mächtige Undo Funktionen in Softwaresystemen wie z.B. Text- oder Bildverarbeitung.

Grundsätzlich dienen eigentlich alle in diesem Kapitel genannten Prinzipien der **Fehlervermeidung**: **Affordances** signalisieren uns die richtige Operation für ein Bedienelement, so dass die falschen Operationen gar nicht erst ausgeführt werden. **Constraints** schränken die möglichen Operationen auf die sinnvollen und erlaubten ein. **Mappings** erhöhen die Verständlichkeit von Bedienelementen und sorgen ebenfalls dafür, dass auf Anhieb richtige Operationen ausgeführt werden. Gleiches gilt für **Konsistenz** und sinnvolles **Feedback**.

7.7 Interface Animation

Seit die Rechenleistung bei der Gestaltung von Benutzerschnittstellen kein begrenzender Faktor mehr ist, müssen Veränderungen der grafischen Darstellung nicht mehr wie früher in einem Schritt vorgenommen werden. Sie können stattdessen auch graduell durchgeführt oder animiert werden. Diese **Animationen** können – richtig eingesetzt – das Verständnis der Vorgänge sehr stark unterstützen, indem sie verhindern, dass die Veränderung der Darstellung der in Abschnitt 2.1.1 erwähnten **change blindness** zum Opfer fallen.

Ein Beispiel soll dies verdeutlichen: Angenommen, der Bildschirm zeigt ein Brettspiel an, bei dem es viele gleiche Steine gibt, beispielsweise Mühle, Dame oder Backgammon. Nun ist der Computer am Zug und verschiebt 2 Spielsteine. Nehmen wir zunächst an, dass diese schlagartig versetzt werden, der Benutzer also von einem Augenblick zum nächsten eine neue Spielsituation vorfindet. In diesem Fall muss er diese neue Spielsituation zunächst neu analysieren und dabei auch rekonstruieren, wie sie aus der vorherigen (an die er sich hoffentlich noch erinnert) hervorging, wenn er etwas über die vermeintlichen Absichten des Spielgegners herausfinden will. Ist die Bewegung der Spielsteine hingegen animiert, dann wird dieser zusätzliche kognitive Aufwand hinfällig, da der Benutzer flüssig mitverfolgen kann, wie die neue Spielsituation aus der alten hervorgeht.

Der Benutzer muss nicht rekonstruieren, an welchen Stellen des Spielbrettes sich etwas verändert haben könnte, sondern sieht dies anhand der Animation. Ein anderes Beispiel für den sinnvollen Einsatz von Animationen ist das Verkleinern von Fenstern in modernen Desktop-Interfaces: Ein Fenster, das geschlossen wird, verschwindet hier nicht einfach schlagartig vom Bildschirm und taucht in der verkleinerten oder symbolischen Ansicht am unteren Bildschirmrand auf. Stattdessen verkleinert es sich kontinuierlich während es sich auf seine Zielposition zu bewegt. Durch diese Animation wird eine visuelle **Kontinuität** geschaffen, und der Benutzer kann sich leicht merken, wohin sein verkleinertes Fenster verschwunden ist. Allgemein können Interface-Animationen also stark das Verständnis visueller Abläufe unterstützen, genauso können sie aber auch – falsch eingesetzt – ablenken und verwirren.

Exkurs: Fehlende Animation

Das Fehlen einer Animation im Interface kann auch darüber entscheiden, ob eine Interaktion überhaupt verständlich bleibt oder nicht. In einem Interfaceentwurf der Autoren aus dem Jahre 2001 für ein mobiles Museums-Informationssystem (im Bild rechts) war eine Leiste mit Thumbnails enthalten, sowie zwei angedeutete Pfeile links und rechts, die zeigen sollten, dass es in beide Richtungen noch mehr Bilder zur Auswahl gab. Die Leiste war visuell angelehnt an einen Filmstreifen. Der Entwurf sah vor, dass man durch Anklicken der Pfeile oder durch Greifen mit dem Stift den Streifen nach rechts oder links verschieben konnte. In der technischen Umsetzung auf dem mobilen Gerät stand dann nicht genügend Rechenleistung zur Verfügung, um eine solche Animation flüssig darzustellen.

Der Programmierer entschied damals kurzerhand, auch die Pfeile mangels Platz einzusparen, da sie eh zu klein und daher schwer zu treffen waren. Stattdessen implementierte er eine Logik, die das momentan selektierte Thumbnail-Bild mit einem roten Rahmen versah und beim click auf ein benachbartes Bild dieses schlagartig zum aktuellen Bild machte. Die Leiste verschob sich damit immer schlagartig um ein ganzes Bild und die Veränderung war überhaupt nicht mehr als Verschiebung zu erkennen, sondern lediglich als Neuanordnung der Bilder. Damit war die ursprüngliche Analogie zum verschiebbaren Filmstreifen völlig verloren und die jeweilige Neuanordnung der Bilder erschien willkürlich, da nicht nachvollziehbar war, wo das zuletzt angesehene Bild nun hingewandert war.

Das gesamte Interface wurde später zugunsten eines anderen Konzeptes verworfen. Weitere amüsante Einsichten aus dem Projekt finden sich in Butz [16].

7.8 Physikanalogie

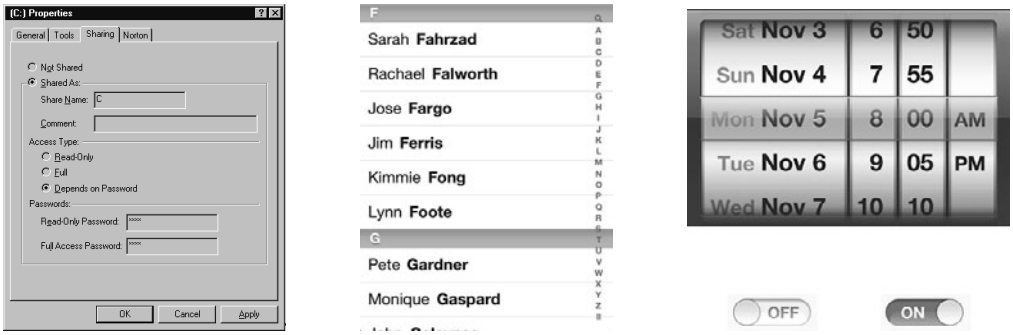
Ein sehr spezifisches Rezept zur Gestaltung verständlicher Benutzerschnittstellen ist es, diesen bis zu einem gewissen Grad physikalisches Verhalten mitzugeben. Den Umgang mit der physikalischen Welt haben wir buchstäblich von Kindesbeinen an gelernt. Wenn nun eine grafische Benutzerschnittstelle Verhaltensweisen zeigt, die wir in der physikalischen Welt kennengelernt haben, dann verstehen wir diese Verhaltensweisen intuitiv, also ohne langes Nachdenken. Die digitale Welt verhält sich an dieser Stelle genau so, wie es die physikalische Welt unseres Alltags täte, und wir kommen einer der Kernthesen **Mark Weisers** ein Stückchen näher, der schreibt:

> The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it. [125]

Physikanaloges Verhalten oder **Physikanalogie** stellt ein sehr mächtiges Mittel zur Kommunikation dar. Das intuitive Verständnis des physikalischen Verhaltens macht es fast so effektiv wie grafische Darstellungen, die die **präattentive Wahrnehmung** ausnutzen, die wir in Abschnitt 2.1.3 kennengelernt haben. Physikanalogie kann uns sowohl bestimmte **Affordances** als auch **Constraints** vermitteln. Der Benutzer kann Teile seines **mentalen Modells** von der physikalischen Welt auf die digitale Welt übertragen. Das ermöglicht ihm die Wiederverwendung von Wissen und Fähigkeiten sowie Analogieschlüsse.

Einige Beispiele hierzu: Physikanalogie beginnt bereits in der angedeuteten dreidimensionalen Darstellung grafischer Bedienelemente in den frühen grafischen Benutzerschnittstellen der 90er Jahre des letzten Jahrhunderts (siehe Abbildung 7.7 links). Die Tatsache, dass Buttons und andere Bedienelemente plastisch aus ihrer Umgebung hervortraten, gab ihnen das Aussehen von physikalischen Druckknöpfen, die in der echten Welt aus der sie umgebenden Fläche heraustreten. Genau wie ihre physikalischen Vorbilder sanken auch diese Buttons (visuell) ein, wenn sie gedrückt wurden. Mit einem Blick konnte man damals also schon interaktive Elemente (3D) von nicht interaktiven (flach) unterscheiden. Die interaktiven Elemente vermittelten damit ihre **Affordances**.

In Abbildung 7.7 Mitte sehen wir ein Bedienelement, das vor allem im Verhalten ein hohes Maß an Physikanalogie besitzt: Die Standard-Listenansicht in iOS lässt sich mit dem Finger auf- und abwärts schieben. Sie folgt dabei dem Finger genau, solange dieser auf dem Bildschirm bleibt. Verlässt der Finger den Bildschirm in Bewegung, so bleibt auch die Liste in Bewegung und verschiebt sich mit langsam abnehmender Geschwindigkeit weiter, bis sie entweder von selbst zum Stehen kommt oder an ihrem Ende angelangt ist. Dieses Verhalten entspricht einer gleitend gelagerten Scheibe, die die Listeneinträge enthält und unter dem Sichtfenster hindurch gleitet. Die Trägheit ihrer Masse würde dafür sorgen, dass sie sich weiter bewegt, die Reibung dafür, dass die Geschwindigkeit abnimmt. Durch die **Physikanalogie** ergibt sich automatisch, dass eine solche Liste langsam oder auch sehr schnell durchgeschoben werden kann, und wir schieben eine sehr lange Liste auch mehrfach mit dem Finger an, um ihr mehr Schwung zu geben. Schiebt man die Liste schließlich an ihr Ende, so stößt sie dort nicht ganz hart an,



mmibuch.de/a/7.7

Abbildung 7.7: Physikanaloge Bedienelemente aus 20 Jahren: Links: Buttons in Windows 95, Mitte: Tabellenansicht aus iOS mit Trägheit, Reibung und elastischem Anschlag, Rechts: Datumsdialog und Schalter in iOS

sondern federt noch etwas nach. Damit wird klar vermittelt, dass wir nur am Ende der Liste angelangt sind und nicht etwa das Gerät eingefroren ist. Die Liste vermittelt damit ein **logisches Constraint** so, also ob es ein **physikalisches Constraint** wäre. Auch die in Abbildung 7.7 rechts unten gezeigten iOS Schalter vermitteln den Eindruck einer mechanischen Funktion. Sie entsprechen einem physikalischen Schiebeschalter in Funktionslogik und visueller Erscheinung. Durch ihr Aussehen vermitteln sie die **Affordance**, dass sie seitlich geschoben werden können, und das **Constraint**, dass sie nur in einem von zwei Zuständen sein können.

Abbildung 7.7 Rechts oben zeigt schließlich ein komplett physikanalog zu Ende gedachtes Bedienelement, den Datums- und Uhrzeitdialog in iOS 6. Obwohl es für dieses konkrete Element kein direktes physikalisches Vorbild gibt, erschließt sich seine Funktion doch sofort: die einzelnen Ringe können gedreht werden, um den zugehörigen Wert einzustellen und unter dem durchsichtigen Kunststoffstreifen in der Mitte steht das aktuell eingestellte Datum. Auch in diesem Beispiel sind die einzelnen Einstellringe mit Trägheit und Reibung ausgestattet und erzeugen zusätzlich beim Einrasten auf einen bestimmten Wert noch ein klickendes Geräusch. Ganz nebenbei lässt sich hier das **logische Constraint** verschiedener tatsächlich möglicher Tageszahlen für die verschiedenen Monate mit einbauen, was wiederum bei der **Fehlervermeidung** hilft. Interessanterweise stört der angedeutete Kunststoffstreifen in der Mitte die Bedienung der Ringe nicht, was in der physikalischen Welt sehr wohl der Fall wäre. Hier wird also gezielt auch von der physikalischen Realität abgewichen, wo eine hundertprozentige Analogie nur störend wäre. Der Benutzer nimmt dies zwar irgendwann wahr, stört sich aber nicht an der Abweichung, da sie hilfreich ist.

Schaut man sich die Umsetzung der Physikanalogie auf technischer Ebene genauer an, so findet man weitere Abweichungen: das physikalische Modell, das beispielsweise den iOS Listen unterliegt, ist genau gesehen ein Modell, das das Verhalten von Körpern in Flüssigkeiten beschreibt, nicht weil dies physikalisch korrekter wäre, sondern einfach weil

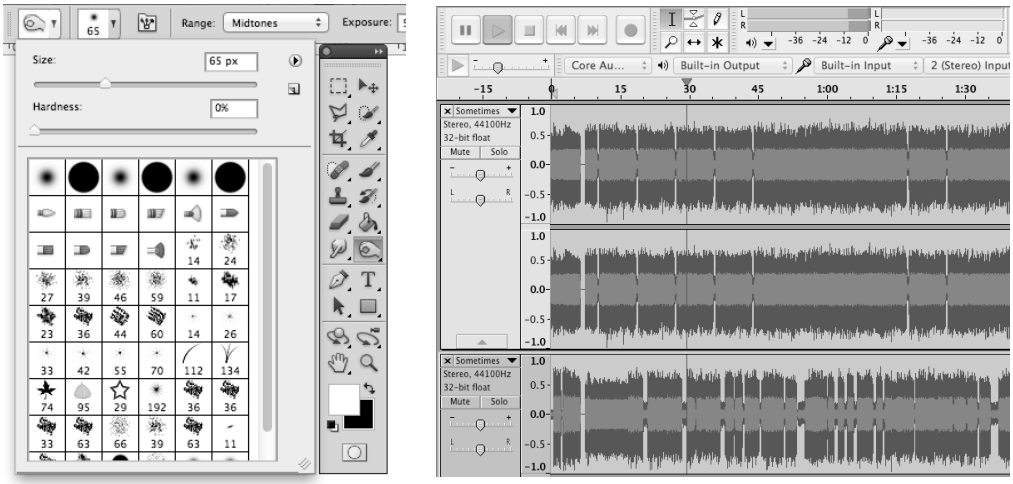
es besser aussieht. Außerdem werden Effekte und Verhaltensweisen aus dem Bereich der Animation und des Trickfilms eingesetzt, wie sie z.B. in Thomas und Johnston [113] als Animationsprinzipien beschrieben sind. Effekte auf Basis von Übertreibung oder ansprechendem Timing sorgen dafür, dass solche Schnittstellen nicht nur physikalisch plausibel erscheinen, sondern sogar einen gewissen *Charakter* entwickeln. Schließlich wird der Gedanke der Physikanalogie in Jacob et al. [57] auf das generellere Konzept realitätsbasierter Interfaces (**RBI**) ausgeweitet, bei denen außer der Physik auch noch der räumliche, logische und soziale Kontext aus der Realität übertragen wird.

7.9 Metaphern als Basis für UI

Die oben eingeführte Physikanalogie ist ein besonders prominentes Beispiel für eine allgemeinere Vorgehensweise bei der Strukturierung und Gestaltung von Benutzerschnittstellen, nämlich für die Verwendung von **Metaphern**. Die Mächtigkeit von Metaphern als Basis unseres Denkens und Verstehens der Welt wird von den Psychologen **George Lakoff** und **Mark Johnson** in Lakoff und Johnson [71] eindrücklich dargestellt, und die Verwendung einer Metapher zwischen zwei Diskursbereichen bedeutet letztlich einfach, dass wir Teile des zugehörigen mentalen Modells von einem Bereich in den anderen übertragen können.

Das wohl prominenteste Beispiel ist die **Desktop Metapher** in den heute noch vorherrschenden grafischen Benutzerschnittstellen am **Personal Computer** (siehe Kapitel 15 sowie Abbildung 15.1 auf Seite 158). Sie wurde entworfen, um die Gedankenwelt von Sekretärinnen und Büro-Angestellten möglichst gut abzubilden, damit diesen der Umgang mit dem Computer leichter fallen sollte. Gemeinsam mit dem Bedienprinzip der **Direkten Manipulation** bildet diese Metapher noch heute die Grundlage unseres Denkens und Redens über den Personal Computer. Weil im Büro schon immer mit *Dokumenten* gearbeitet wurde, die ihrerseits in *Mappen* oder *Akten* (engl. *files*) und diese wiederum in *Ordnern* (engl. *folder*) abgelegt sind, reden wir heute von elektronischen Dokumenten in Ordnern, statt von Dateien in Verzeichnissen. Die grafische Darstellung des Dokuments in Form eines **Icons** ist für uns gleichbedeutend mit der dazugehörigen Datei auf technischer Ebene. Alle Operationen spielen sich auf einem gedachten Schreibtisch ab. Weitere Icons symbolisieren für uns weitere Funktionen: ein Druckersymbol die Druckfunktion, ein Papierkorb die Löschfunktion. Innerhalb dieser Büroumgebung haben wir also die Möglichkeit, bekannte Elemente aus einem echten Büro wiederzuerkennen und von deren Funktion auf die Funktion des entsprechenden Icons im Computer zu schließen. Das Mentale Modell einer Büroumgebung wird übertragen auf den Computer und ermöglicht uns die Übertragung von Fähigkeiten und Aktivitäten. Wir können z.B. Ordnung schaffen, indem wir zueinander gehörende Dokumente nebeneinander auf dem Schreibtisch oder in einem gemeinsamen Ordner ablegen.

Für speziellere Tätigkeiten innerhalb der gängigen grafischen Benutzerschnittstellen haben sich andere, besser passende Metaphern etabliert: Zum Zeichnen und Bearbeiten von Bildern verwenden viele Bildbearbeitungsprogramme eine Malerei-Metapher (Abbildung 7.8 links). Die leere Zeichenfläche heißt dabei *Canvas*, wie die Leinwand, auf der Ölgemälde gemalt werden. Die Werkzeuge sind Stift, Pinsel und Airbrush, Radierer oder Lineal. Angeordnet sind sie in *Werkzeugpaletten* oder *-kästen*. Farben befinden



mmibuch.de/a/7.8

Abbildung 7.8: Links: Pinsel, Stifte, Radierer etc. als Metapher in der Bildbearbeitung. Rechts: Tonspuren, Pegel und Regler in der Audiobearbeitung

sich in *Farbpaletten*. Die grundlegenden Tätigkeiten beim Zeichnen können damit direkt übertragen werden. Gleiches gilt für die Bearbeitung von Tonmaterial: Hier wird die Metapher eines Mischpultes oder Mehrspur-Recorders ausgenutzt (Abbildung 7.8 rechts). Es ist die Rede von *Kanälen*, *Pegeln* und *Effekten*. Eine solche Metapher und die schlüssige Verwendung der zugehörigen Terminologie ermöglicht einem Tontechniker, der früher mit analogen Geräten arbeitete, einen wesentlich einfacheren Einstieg, da er zumindest relevante Teile seines mentalen Modells übertragen kann und somit im neuen Arbeitsumfeld Computer auf seine erlernten Techniken und Arbeitsweisen zurückgreifen kann.

Eine andere prominente Metapher ist der Begriff der *cloud* im *cloud computing*. Die Bezeichnung eines irgendwo vorhandenen Netzes aus Servern, die Rechen- und Speicherleistung anbieten als *Wolke* spielt bewusst mit den Begriffen *wolkig* (engl. *cloudy*) und kommuniziert damit unterschwellig, dass es gar nicht so wichtig ist, genau zu verstehen, welche technische Infrastruktur benutzt wird. Die Metapher dient hier also nicht zur Übertragung eines vertrauten **mentalen Modells**, sondern zur Vermittlung der Tatsache, dass gar kein exaktes mentales Modell notwendig oder überhaupt gewollt ist. Insbesondere soll auch das in der Regel sehr komplexe **implementierte Modell** vor dem Benutzer versteckt werden.

In den frühen 1990er Jahren präsentierte Microsoft unter dem Namen **Bob** eine Grafische Benutzerschnittstelle für PCs, die eine andere Metapher benutzte. Statt auf einer Schreibtischoberfläche waren alle Objekte nun in einem Raum angeordnet (siehe Abbildung 7.9), vermutlich um besser auf die verschiedenen Lebensbereiche der Privatnutzer eingehen zu können. Es gab verschiedene solche Räume mit unterschiedlicher Ausstat-

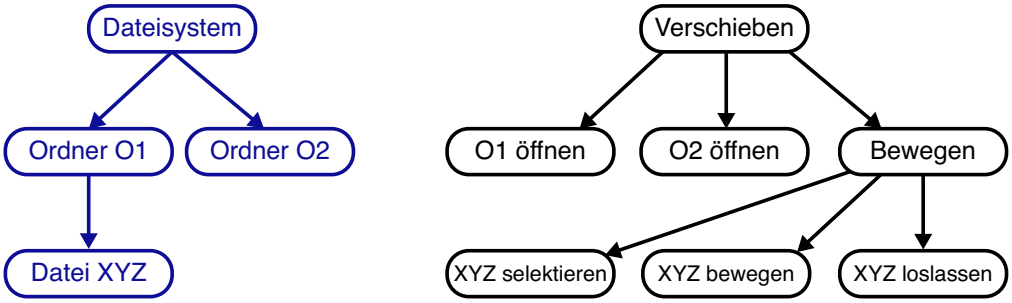
Abbildung 7.9: Links: Microsoft Bob: Metapher, die als Nachfolger von Windows 3.1 vorgestellt, jedoch nie akzeptiert und später von Windows 95 überholt wurde. Rechts: Hund als Metapher für die Suche in Windows XP

mmibuch.de/a/7.9

tung, und der Benutzer konnte sie selbst umräumen und umorganisieren. Bewohnt waren die Räume von hilfreichen Gesellen, wie z.B. dem Hund **Rover**, der – für einen Hund sehr passend – eine Suchfunktion verkörperte. Während das Bob Interface später durch Windows 95 komplett verdrängt wurde und sich nie durchsetzen konnte, hielt sich der Hund als **Metapher** für die Suchfunktion noch bis zur letzten Version von Windows XP im Jahre 2007.

7.10 Object-Action Interface Modell

Eine weitere prinzipielle Überlegung hilft uns beim grundsätzlichen Entwurf neuer Benutzerschnittstellen und insbesondere im Zusammenhang mit Metaphern: Das **Object-Action Interface Modell** oder **OAI Modell**. Es wurde von **Ben Shneiderman** in älteren Ausgaben seines Standardwerks *Designing the User Interface* [105] eingeführt, fehlt jedoch in der aktuellen Ausgabe. Benutzerschnittstellen sind demnach maßgeblich durch die in ihnen vorkommenden Objekte und Aktionen charakterisiert. In Desktop-GUIs sind die Objekte beispielsweise Dateien, dargestellt durch Icons, und die Aktionen sind Nutzeraktionen wie z.B. das Bewegen, Löschen oder Editieren einer Datei. Diese grundlegende Unterscheidung nach Objekten und Aktionen ist wesentlich für unser Denken und das Verständnis der Welt, spiegelt sich in unserer Sprache in der Unterscheidung zwischen Substantiven und Verben wieder, und ist auch in vielen UI-Konzepten finden. Beim Entwurf eines Bedienkonzeptes haben wir nun die Möglichkeit, entweder



mmibuch.de/a/7.10

Abbildung 7.10: Links: Objekthierarchie, Rechts: Aktionshierarchie zum Verschieben einer Datei

mit dem Objekt, das wir manipulieren wollen, zu beginnen, oder mit der Aktion, die wir durchführen wollen. Ein Beispiel: In einem Desktop GUI selektieren wir zuerst ein Icon mit der Maus, um es dann zum Papierkorb zu bewegen und damit zu löschen. Diese Funktionsweise entspricht damit dem **OAI Modell**. Umgekehrt geben wir in einem Kommandozeilen-Interface zuerst die Aktion (das Kommando) an und erst danach das betroffene Objekt: `rm filename.txt`. In einem Bildbearbeitungsprogramm wählen wir auch zuerst das Werkzeug (Pinsel, Schere) aus, bevor wir mit diesem das Bild selbst manipulieren. Diese umgekehrte Vorgehensweise entspricht dem sogenannten **Action-Object Interface Modell** oder **AOI Modell**. Obwohl diese grundlegende Überlegung relativ alt ist, hilft sie uns immer noch dabei, klare und schlüssige **konzeptuelle Modelle** für eine Benutzerschnittstelle zu entwerfen.

Shneiderman führt seine Überlegung weiter, indem er sowohl für Objekte als auch für Aktionen Hierarchien definiert. Objekte bestehen aus Teilobjekten, die evtl. getrennt manipuliert werden. Aktionen bestehen genauso aus Teilschritten, die bei komplexeren Vorgängen noch weiter untergliedert werden können. Ein Beispiel aus dem Alltag: Der Vorgang des Kaffeekochens besteht daraus, die Maschine zu befüllen, anzuschalten, abzuwarten, und irgendwann die Kanne mit dem gekochten Kaffee zu entnehmen. Das Befüllen der Maschine besteht wiederum aus dem Einfüllen von Filter und Pulver, sowie dem Eingießen von Wasser. Um Wasser einzugießen, muss wiederum der Deckel der Maschine geöffnet und dann Wasser in die entsprechende Öffnung gegossen werden. Wir erkennen hier also eine Hierarchie von Aktionen, die sich als Baum darstellen lässt. Genauso besteht die Kaffeemaschine aus einer Hierarchie von Objekten: sie besitzt u.a. einen Wasserbehälter, und der wiederum einen Deckel.

Ein weiteres Beispiel aus der Computerwelt: Das Verschieben eines Dokumentes von einem Ordner in einen anderen ist eine komplexe Aktion in einer komplexen und dynamischen grafischen Umgebung: Das Dateisystem des Computers enthält verschiedene Ordner, evtl. mit Unterordnern. In einem dieser Ordner befindet sich die fragliche Datei. Die Aktionsfolge besteht daraus, zunächst beide Ordner zu öffnen und dann das

Datei-Icon von einem in den anderen Ordner zu bewegen. Das Bewegen der Datei wiederum besteht aus dem Selektieren des Icons mit dem Mauszeiger, der Mausbewegung bei gedrückter linker Maustaste, sowie dem Loslassen im neuen Ordner. Die zugehörigen Objekt- und Aktionshierarchien sind in Bild 7.10 gezeigt.

Das ursprüngliche Argument für die **OAI** und **AOI Modelle** war nun, dass eine Interface-Metapher leichter verständlich wird, wenn die Hierarchie ihrer Objekte und Aktionen mit den Objekt- und Aktions-Hierarchien der echten Welt (oder jedenfalls des Diskursbereichs der Metapher) möglichst gut übereinstimmen. Diese Argumentation führt jedoch zu sehr genauen Übertragungen der physikalischen Welt auf die Benutzerschnittstelle, was wiederum nicht immer notwendig und zuträglich ist. Im Gegenteil greifen viele gängige Metaphern ihren Diskursbereich nur in bestimmten Grenzen direkt auf, um ihn dann in anderen Bereichen selbst und anders weiterzuentwickeln. Trotzdem hilft das Denken in Objekten und Aktionen sowie das systematische Gestalten von Objekt- und Aktions-Hierarchien bei der logischen Strukturierung einer Benutzerschnittstelle und unterstützt damit Lernen und Verstehen. Die Bedeutung von Objekten und Aktionen im Rahmen des Designprozesses von Benutzerschnittstellen werden wir in Kapitel 10 noch einmal aufgreifen.

Übungsaufgaben:

mmibuch.de/l/7

1. Nennen Sie fünf Arten von Sitzgelegenheiten, die sich in ihren Affordances unterscheiden und diskutieren Sie diese.
2. Gehören Sie noch zu der Generation, die Computerbefehle auswendig gelernt hat und im Schlaf beherrscht, oder verlassen Sie sich bereits darauf, dass alles überall sichtbar und erkennbar ist? Finden Sie in Ihrem Alltag eine Interaktion mit einem computergestützten Gerät (ruhig auch Wecker, Pulsuhr, Mikrowelle, etc.), die Sie mittels Kommandos ausführen, und für die Sie die Kommandos lernen und behalten müssen, weil sie nicht sichtbar oder erkennbar sind. Wie könnte diese Situation einfach und realistisch, also ohne wesentliche Designänderungen verbessert werden?
3. Wie lassen sich Dialoge generell beschleunigen? Denken Sie über Breite und Tiefe des Dialogbaumes nach, aber auch über die Häufigkeit, mit der die verschiedenen Dialogschritte jeweils ausgeführt werden. Falls Sie sich mit Codierungsverfahren auskennen, finden Sie Inspiration bei der Huffmann-Codierung. Nennen Sie Beispiele aus Ihrem Alltag, bei denen solche optimierten Dialogbäume angewendet werden!

8 Etablierte Interaktionsstile

In diesem Kapitel wollen wir in aller Kürze einige etablierte **Interaktionsstile** am **Personal Computer** (**PC**) systematisch betrachten und ihre Herkunft sowie Vor- und Nachteile diskutieren. Dieses systematische Verständnis dient uns einerseits beim gezielten Einsatz dieser Interaktionsstile in konventionellen PC-Benutzerschnittstellen. Es hilft uns andererseits aber auch beim Nachdenken über völlig neue Arten der Interaktion. Teil IV dieses Buches diskutiert einige fortgeschrittene Interaktionsparadigmen sowie die damit verbundenen Besonderheiten im Detail.

8.1 Kommandos

Computer leisten auf unterster technischer Ebene nichts anderes, als Folgen von **Kommandos** abzuarbeiten. Diese Maschinenbefehle stehen hintereinander im Arbeitsspeicher und enthalten beispielsweise mathematische Operationen, Verzweigungen oder Sprunganweisungen. Als die ersten Computer entwickelt wurden, waren ihre Konstrukteure gleichzeitig auch ihre Programmierer und ihre einzigen Nutzer. Die Bedienung eines Computers bestand darin, ein Programm zu schreiben und es dann auszuführen. Mit der Zeit trennten sich diese verschiedenen Rollen immer weiter auf und heute scheint es absurd, von einem Benutzer zu verlangen, dass er auch programmieren oder gar Computer bauen können soll. Trotzdem enthalten alle PC-Betriebssysteme noch heute sogenannte **Kommandozeilen-Umgebungen** oder **Shells** (siehe Abbildung 8.1) und sogar bei manchen Smartphone-Betriebssystemen kann man noch auf eine Kommando-Ebene zugreifen.

Das Denkmodell, das diesen Kommandozeilen-Umgebungen zugrunde liegt, ist also das eines Computerprogramms: der Benutzer gibt einen Befehl ein und der Computer gibt das Ergebnis dieses Befehls am Bildschirm aus. Grundlegend muss der Benutzer daher zunächst einmal wissen, welche Befehle er überhaupt zur Verfügung hat. In manchen Kommandozeilen-Umgebungen gibt es hierfür Hilfe, nachdem man das Kommando *help* oder *?* eingegeben hat, was man aber auch zuerst einmal wissen muss. Ein grundlegendes Problem der Kommandozeile ist also, dass der Benutzer sich an Kommandos samt Syntax und Optionen *erinnern* muss, statt sie irgendwo *erkennen* zu können (vgl. Abschnitt 3.1.2 zu den Konzepten *recall* und *recognition*).

In Kommandozeilen-Umgebungen lassen sich sehr komplexe Abläufe als Folgen von Kommandos spezifizieren und auch automatisieren, indem man diese Kommandofolgen wiederum in ausführbare Dateien, die sogenannten **shell scripts** schreibt. Das Unix Betriebssystem in seinen verschiedenen Spielarten wird komplett durch solche Scripts gesteuert und konfiguriert. Für den geübten Benutzer stellen Kommandozeilen und scripts also sehr mächtige Werkzeuge dar, mit deren Hilfe komplexe Vorgänge ange-

mmibuch.de/a/8.1

Abbildung 8.1: Kommandozeilen-Umgebungen in Windows, OSX und Linux

stoßen werden können, die sich in dieser Form in rein grafischen Benutzerschnittstellen nicht ausdrücken lassen (Einfaches Beispiel: eine programmierte Schleife für wiederholte Vorgänge).

Kommandozeilen sind zwar das bekannteste, jedoch bei weitem nicht das einzige Beispiel für Kommando-basierte Interaktion. Auch die Steuerung einer Stereoanlage mittels der Knöpfe einer Fernbedienung ist letztlich nicht anderes. Jeder Knopf steht für ein bestimmtes Kommando, z.B. Lautstärke erhöhen, Sender wechseln, CD anhalten. Immerhin sind die Knöpfe in diesem Fall beschriftet und der ahnungslose Benutzer kann sehen, welche Kommandos zur Verfügung stehen. Weitere Beispiele sind ein klassisches Autoradio, bei dem es für jede Funktion einen eigenen Knopf oder Regler gibt, sowie bestimmte Fahrkartenautomaten, wie z.B. die älteren Modelle des Münchner Verkehrsverbundes (siehe Abbildung 8.2 Rechts). Bei diesen Automaten existiert für jede Fahrkartenart jeweils ein eigener Knopf, mit dem diese in einem Schritt ausgewählt wird. Eine Kommando-basierte Bedienung bietet sich immer dann an, wenn es eine überschaubare Anzahl von Optionen gibt und diese dem Benutzer entweder vertraut oder klar erkennbar sind und keiner weiteren Interaktion bedürfen.

8.2 Dialoge

Sobald zu einer Anweisung an den Computer komplexere Rückfragen kommen, entsteht ein regelrechter **Dialog** zwischen Benutzer und Computer. Diese Form der Bedienung ist überhaupt erst mit dem Computer entstanden und war vorher bei einfacheren Maschinen gar nicht möglich. Wenn wir uns wieder das Beispiel der Fahrkartenautomaten ansehen, dann sind die neueren Modelle des MVV oder der Deutschen Bahn (siehe Abbildung 8.2 links) gute Beispiele für komplexe Dialoge. Die Fahrkarte wird nicht (wie bei der Kommando-Variante) in einem Schritt ausgewählt, sondern verschiedene Aspekte wie Fahrziel, Zahl der Personen und Gültigkeitsdauer des Tickets werden in mehreren Schritten abgefragt. Die Tatsache, dass oft beide verschiedenen Automaten nebeneinander stehen, stellt übrigens ein gewisses Maß an **Flexibilität** (siehe Abschnitt 5.2.2) in dieser Benutzerschnittstelle sicher: Jeder kann das bevorzugte Modell verwenden.

Abbildung 8.2: Dialog- und Kommando-orientierte Fahrkartenautomaten des Münchner Verkehrsverbundes: am linken Automaten findet ein Bildschirmdialog statt, am rechten entspricht eine Taste genau einem Ticket.

mmibuch.de/a/8.2

Die Vor- und Nachteile gegenüber rein kommandobasierter Bedienung sind damit offensichtlich: Es kann eine sehr viel größere Auswahl an Funktionen gesteuert werden (Sondertickets, Sondertarife). Dafür erfordert die Bedienung jedoch mehrere Schritte. Eine gute Gestaltung des Dialogs ist hierbei von großer Bedeutung, wie die öffentliche Diskussionen bei Einführung der Bahn-Automaten gezeigt haben. Es gilt hier vor allem, Quellen für Fehler und Frustration zu vermeiden, sowie eine gute Balance zwischen **Effizienz** und **Mächtigkeit** des Bedienablaufs zu finden: Ein Dialog, der immer alle Optionen abfragt, ist mächtig aber langsam. Ein Dialog, der nur wenig abfragt ist schnell, bedeutet aber weniger Variationsmöglichkeit. Dialoge mit ihren Verzweigungen kann der Informatiker als Baum beschreiben und durch Umsortieren dieser Bäume lassen sich Dialoge bzgl. der durchschnittlich benötigten Anzahl an Dialogschritten optimieren. Ob sie dabei verständlich bleiben und dem Benutzer als logisch erscheinen, lässt sich jedoch nur durch Benutzertests (vgl. Kapitel 13) verlässlich nachprüfen.

Andere Beispiele für eine dialogbasierte Interaktion sind Installationsdialoge für Software oder **Sprachdialogsysteme**, die sog. *Telefoncomputer*. Letztere haben das weitere Problem, dass sie aus dem gesprochenen Sprachsignal zunächst einmal die richtigen Worte und dann deren Bedeutung im Kontext ableiten müssen. Einfache Kommandoeingaben zur Navigation in Menüs am Telefoncomputer sind mittlerweile sehr robust machbar. Die Verarbeitung komplexerer Sprache stellt jedoch weiterhin ein aktives Forschungsgebiet dar und ist stets mit erheblichem Rechenaufwand verbunden. Die Analogie zu einem Dialog zwischen zwei Menschen weckt hierbei immer wieder falsche Erwartungen: Während wir bei einem menschlichen Gegenüber davon ausgehen können, dass der Dialog mit einem Mindestmaß an Intelligenz und *common sense* geführt wird, wird diese Annahme beim Dialog mit einem Computer regelmäßig enttäuscht. Die Geschichte der Sprachdialogsysteme von Eliza [127] bis Siri[1] ist daher gepflastert mit spöttischen Gegenbeispielen, die ein Versagen des Dialogs provozieren.

8.3 Suche und Browsen

Wenn wir es mit größeren Datenmengen zu tun haben, verwenden wir als Interaktionsform oft die gezielte Suche (engl. **search**) oder wir stöbern weniger zielgerichtet (engl. **browsing**). Die Suche selbst kann ihrerseits zwar wieder als Dialog verlaufen oder sogar in Form eines Kommandos spezifiziert werden (z.B. SQL-Anfrage an eine Datenbank). Charakteristisch ist jedoch, dass wir oft nicht mit dem ersten Ergebnis zufrieden sind oder gar eine Menge von Ergebnissen zurückbekommen, die eine weitere Verfeinerung der Suche erfordern. Der Begriff *browsing* hat sich vor allem für das Stöbern im World Wide Web etabliert und als *browsen* mittlerweile sogar seinen Weg in den Duden gefunden. Er bezeichnet das mehr oder weniger zielgerichtete Bewegen in einem großen Datenbestand wie dem Web, einer Bibliothek oder Musiksammlung (Abbildung 8.3).

Der Übergang zwischen *suchen* und *browsen* ist fließend und wir Menschen wechseln übergangslos zwischen diesen beiden Aktivitäten hin und her. Wir gehen beispielsweise in ein Geschäft, um gezielt etwas zu kaufen, lassen uns dabei aber auch inspirieren von anderen Dingen, die wir unterwegs sehen, und kommen am Ende oft mit ganz anderen Dingen aus dem Geschäft. Während dieser Sachverhalt im Kaufhaus einfach kommerzielle Interessen unterstützt, ist er doch oft eine wünschenswerte Eigenschaft für Benutzerschnittstellen. Suchmaschinen im Web bieten beispielsweise Ergebnisse an, die nur entfernter etwas mit der Suchanfrage zu tun haben. Interfaces zur Verwaltung von Musiksammlungen bieten oft ein Suchfeld, in dem bestimmte Titel, Alben oder Künstler direkt nach Namen gesucht werden können. Gleichzeitig stellen sie aber auch CD-Cover grafisch dar oder bieten uns (nach bestimmten Kriterien) *ähnliche* Musikstücke an. Dadurch lassen wir uns als Menschen inspirieren und schweifen vom eigentlichen Suchziel ab (siehe z.B. Hilliges und Kirk [48]). Ein weiterer Effekt ist das sogenannte **satisficing**, das aus den englischen Wörtern *satisfying* und *sufficing* zusammengesetzt ist: Wir beginnen mit einem bestimmten, möglicherweise vagen Suchziel, suchen jedoch nicht, bis wir eine exakte Lösung gefunden haben, sondern lediglich so lange, bis ein gefundenes Ergebnis unseren Ansprüchen eben genügt. Dabei lassen wir uns durchaus auch ablenken (**sidetracking**). Gute Interfaces für Suche und Browsen sollten diese beiden

[1] http://www.siri-fragen.de

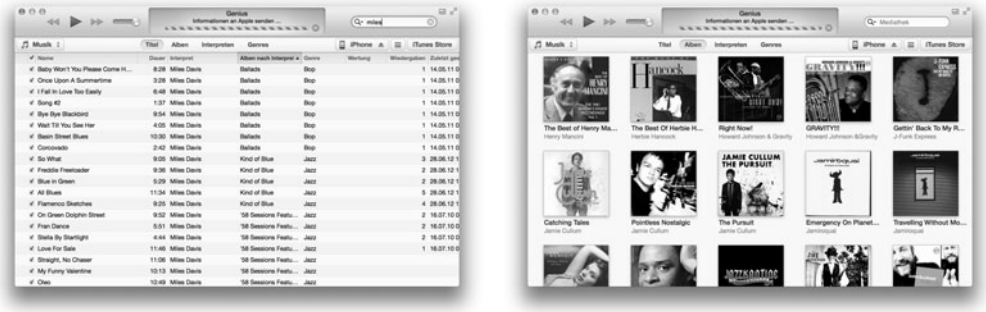


Abbildung 8.3: Suchen und Browsen innerhalb der gleichen Benutzerschnittstelle, hier zum Durchsuchen oder Durchstöbern einer Musiksammlung

mmibuch.de/a/8.3

Mechanismen *sidetracking* und *satisficing* unterstützen, um in großen Datenbeständen eine fruchtbarere Suche sowie eine individuelle Balance zwischen Qualität der Ergebnisse und Schnelligkeit der Suche zu ermöglichen.

8.4 Direkte Manipulation

Direkte Manipulation ist der übergeordnete Interaktionsstil mit heutigen grafischen Benutzerschnittstellen am Personal Computer. **Ben Shneiderman** [104] definierte diesen Interaktionsstil 1983 durch folgende Eigenschaften:

1. Visibility of Objects and Actions (Sichtbarkeit aller Objekte und Aktionen)
2. Rapid, reversible, incremental actions
   (schnelle, umkehrbare, schrittweise ausführbare Aktionen)
3. Replacement of complex command-language syntax with direct, visual manipulation of the object of interest
   (direkte, visuelle Manipulation der Objekte statt komplizierter Kommandos)

Diese Definition ist vor allem vor dem Hintergrund des damaligen Stands der Technik, nämlich Kommandozeilen zu sehen und wird in nachfolgenden Publikation wie Shneiderman [105] auch mit aktualisierter Schwerpunktsetzung neu formuliert. Im Gegensatz zur Kommandozeile werden im **Direct Manipulation Interface** (**DMI**) die zu manipulierenden Objekte sowie die damit ausführbaren Aktionen sichtbar gemacht (1). Dateien werden beispielsweise durch Icons repräsentiert, und Programme oder einfache Kommandos ebenfalls. So kann man mit der Maus ein Datei-Icon nehmen und es zum Papierkorb-Icon ziehen um die Aktion *löschen* damit auszuführen. Die auf diese Art

ausführbaren Aktionen sind schnell, schrittweise, und umkehrbar (2). Das Verschieben einer Datei von einem in den anderen Ordner ist einfach umkehrbar, indem die Datei zurück verschoben wird. Kommandos oder Dateinamen muss der Benutzer also nicht auswendig wissen. Er kann sie stattdessen auf dem Bildschirm sehen, auswählen und direkt manipulieren (3).

Interaktion mit DMI ist daher vor allem für neue oder gelegentliche Benutzer einfach, da diese sich (zumindest im idealen Bild von DMI) nichts merken müssen, sondern den kompletten Funktionsumfang erkennen können. Das gleiche Prinzip wird auch in grafischen Computerspielen verfolgt, in denen alle Spielelemente grafisch dargestellt werden und ihre Funktion erkennbar ist. Eine wesentliche Beschränkung von DMI ist gleichzeitig auch die strikt inkrementelle Ausführung relativ einfacher Aktionen (2). Hierdurch dauern komplexere Vorgänge, wie z.B. das Löschen aller Dateien, die einem bestimmten Kriterium genügen, ggf. sehr lange, während sie auf der Kommandozeile sehr einfach zu spezifizieren wären. Als erstes Direct Manipulation Interface wird gemeinhin das der Xerox Star Workstation genannt (siehe Abbildung 15.1 auf Seite 158). Den kommerziellen Durchbruch schaffte dieser Interaktionsstil jedoch erst 1984 mit dem **Apple Macintosh**. Interessanterweise wird die Interaktion mittels Maus oder anderer Zeigegeräte mittlerweile schon als *indirekt* bezeichnet, wenn man sie mit der Interaktion durch Berührung interaktiver Oberflächen vergleicht (siehe hierzu Kapitel 17).

8.5 Interaktive Visualisierungen

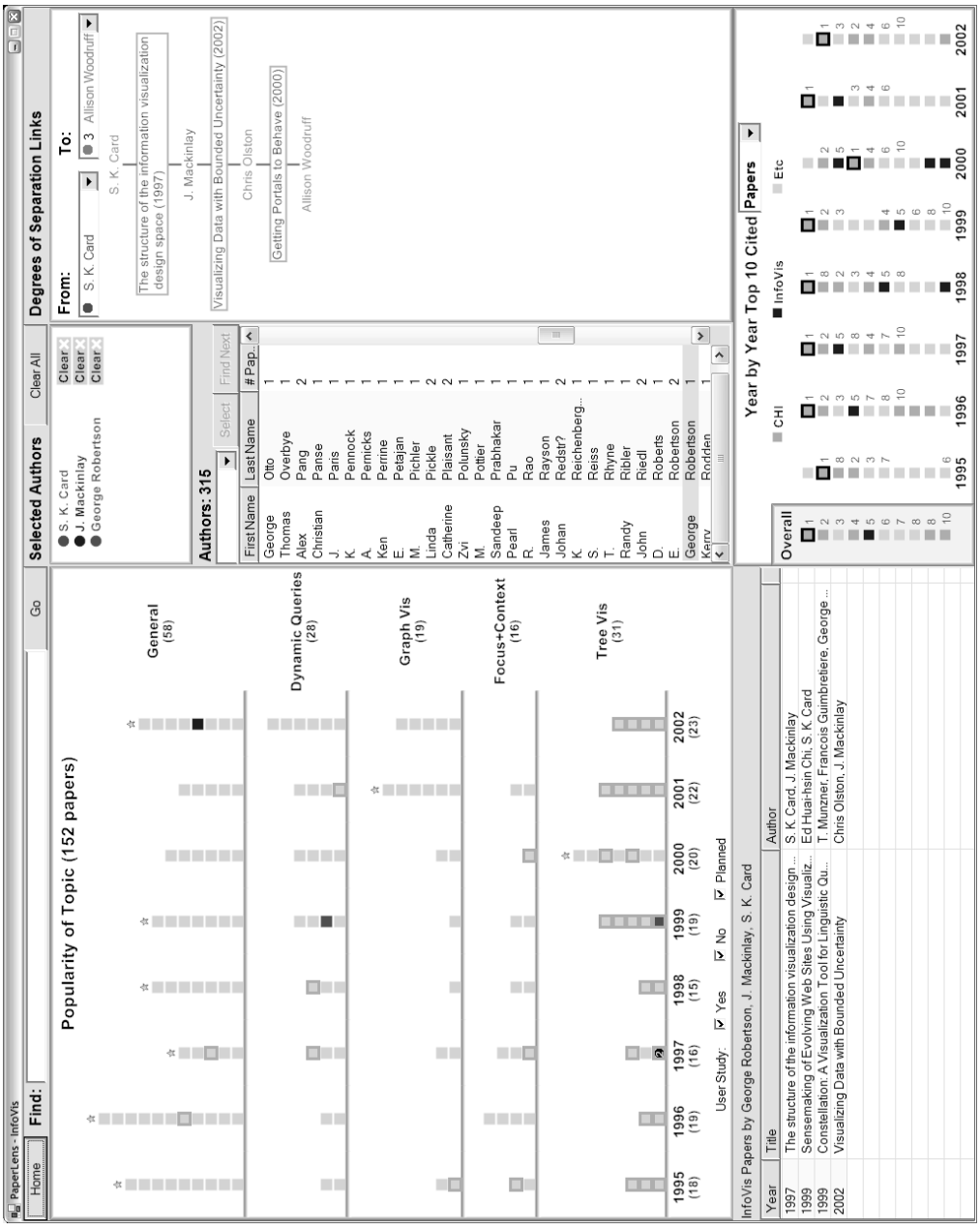
Mit wachsender Rechenleistung der verfügbaren Computer und gleichzeitig rapide wachsenden Datenbeständen haben sich interaktive **Visualisierungen** als Interaktionsstil etabliert. Die grafische Darstellung großer Datenmengen oder komplexer Sachverhalte ist per se keine Errungenschaft des Computerzeitalters, wie **Edward Tufte** [117] eindrucksvoll dokumentiert. Der Computer ermöglicht es jedoch erst, aus veränderlichen Datenbeständen solche visuellen Darstellungen jederzeit neu zu berechnen und damit diese Datenbestände in Echtzeit am Bildschirm zu durchsuchen und zu manipulieren. Hierbei werden viele Effekte der visuellen Wahrnehmung ausgenutzt, wie z.B. die in Kapitel 2 beschriebenen **Gestaltgesetze** oder die **präattentive Wahrnehmung**. Darüber hinaus gibt es ein etabliertes Repertoire an Techniken zur Interaktion mit Visualisierungen, aus dem hier nur zwei beispielhaft genannt werden sollen.

Eine einzelne Visualisierung bietet eine bestimmte Sicht (engl. **view**) auf einen Datenbestand. Zum gleichen Datenbestand können wiederum verschiedene Sichten berechnet werden, beispielsweise nach verschiedenen Filter- oder Ordnungskriterien. Sind mehrere solcher Sichten miteinander koordiniert, spricht man von einer **multiple coordinated views** (**MCV**) Visualisierung. Einfachstes Beispiel ist eine Kartendarstellung, die einen Ausschnitt im Detail zeigt, sowie eine grobe Übersichtskarte, aus der hervorgeht, welcher Ausschnitt eines größeren Gebietes gerade detailliert sichtbar ist (Abbildung 9.3 auf Seite 97). Sind die Sichten derart miteinander koordiniert, dass in der einen Sicht selektierte und hervorgehobene Objekte auch in der anderen Sicht hervorgehoben werden, dann spricht man von **linking**. Solche koordinierten Sichten können beispielsweise benutzt werden, um in einer Sicht Filter- oder Ordnungskriterien anzuwenden und dann in einer anderen Sicht interessante Muster im Ergebnis zu suchen. Diese Technik nennt

man **brushing**. Zusammen bilden *brushing* und *linking* auf MCV Visualisierungen eine mächtige Technik, um in hochdimensionalen Datenbeständen interessante Zusammenhänge herauszufinden. Solche Interfaces wie das in Abbildung 8.4 entfalten ihren vollen Nutzen erst in der aktiven Manipulation. Eine umfassende Einführung in das Gebiet der Informationsvisualisierung gibt beispielsweise **Robert Spence** [108].

Übungsaufgaben:

1. Flexibilität in der Benutzerschnittstelle: Finden und beschreiben Sie auf Ihrem bevorzugten Computersystem mindestens drei Wege, eine Datei zu löschen. Diese drei Wege sollen zu unterschiedlichen (evtl. Kombinationen) der oben genannten Interaktionsstile gehören. Was ist der Grund dafür, überhaupt verschiedene Wege anzubieten, und welche Gründe sprechen jeweils für die einzelnen Wege?
2. Vergleichen Sie die Interaktion mittels Kommandozeile mit der Direkten Manipulation und benutzen Sie dazu Erkenntnisse zum Langzeitgedächtnis aus Abschnitt 3.1.2. Diskutieren Sie, warum Kommandozeilen besser für Experten geeignet sind und die Direkte Manipulation eher für Laien.
3. Informieren Sie sich über das Konzept **Faceted Search** und nennen Sie einige Beispiele aus dem Internet. Inwiefern werden hier Suche und Browsen miteinander kombiniert?



mmibuch.de/a/8.4

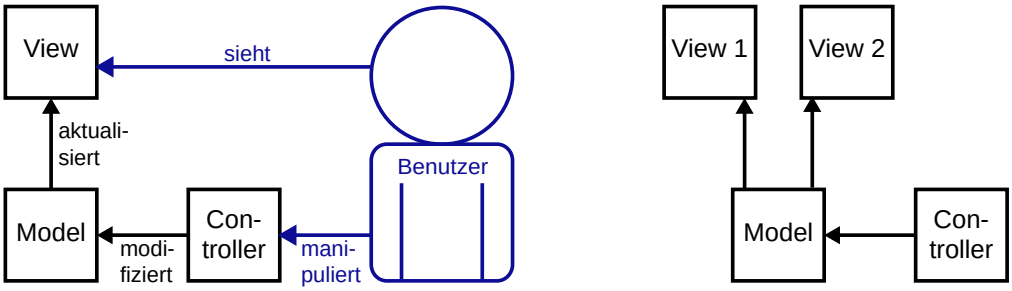
Abbildung 8.4: Ein MCV Interface mit Brushing und Linking: PaperLens, Gewinner des Infovis Contest 2004

9 Einige Grundmuster grafischer Benutzerschnittstellen

Nachdem wir in Kapitel 7 einige grundlegende Regeln für die Gestaltung von Benutzerschnittstellen und in Kapitel 8 einige etablierte Interaktionsstile gesehen haben, befasst sich dieses Kapitel nun mit strukturellen Mustern, also gewissermaßen Rezepten zum Bau von Benutzerschnittstellen. Die hier präsentierte Auswahl ist alles andere als vollständig. Sie umfasst jedoch das derzeit wohl häufigste Entwurfsmuster und einige darauf aufbauende Interface-Konzepte und soll ein Gefühl dafür vermitteln, wie mit solchen Mustern umzugehen ist.

9.1 Ein Entwurfsmuster: Model-View-Controller

Die Idee bei der Definition und Verwendung eines **Entwurfsmusters** ist es, eine gute, getestete und etablierte Struktur zum Entwurf von Software zu formalisieren und wiederverwendbar zu machen. Bestimmte Entwurfsmuster werden durch bestimmte Programmiersprachen oder Programmierumgebungen besonders unterstützt. So bietet beispielsweise Java (und viele andere Programmiersprachen) das **Observer** Muster zur leichteren Implementierung der unten beschriebenen Model-View-Controler Architektur. Entwurfsmuster bieten für bestimmte, immer wieder auftauchende Probleme einen vordefinierten Lösungsweg, der in vielen Fällen zu einer optimalen Lösung führt. Außerdem sorgen sie für einen gewissen Wiedererkennungswert und beschreiben so etwas wie ein **konzeptuelles Modell** der Software für den Entwickler. Das Entwurfsmuster **Model-View-Controller** (**MVC**) beschreibt eine Softwarearchitektur, in der die Daten, um die es geht (*Model*), ihre Darstellung (*View*), sowie ihre Manipulation (*Controller*) klar voneinander getrennt sind. Baut man eine Software nach diesem Muster modular auf, dann können ihre Einzelteile unabhängig voneinander modifiziert oder ausgetauscht werden. Es können auch mehrere *Views* auf das gleiche *Model* definiert werden, oder verschiedene *Controller* für verschiedene Möglichkeiten der Manipulation. Abbildung 9.1 zeigt das MVC Muster, und wie ein Benutzer mit den verschiedenen Komponenten interagiert. Ein praktisches Beispiel soll das Zusammenspiel der verschiedenen Komponenten verdeutlichen: Nehmen wir an, das *Model* sei ein Text als Zeichenkette im Arbeitsspeicher des Rechners. Dann ist der *View* das Stück Software zur Darstellung dieses Textes am Bildschirm, beispielsweise mithilfe einer bestimmten Schriftart im Fenster eines Texteditors. Der *Controller* wäre in diesem Fall das Stück Software, das Eingaben von der Tastatur entgegen nimmt und den Text im Arbeitsspeicher entsprechend modifiziert. Immer, wenn sich der Text im Speicher ändert, muss die Darstellung am Bildschirm aktualisiert werden. Daneben könnte ein weiterer *View* existieren, der den Text in einer Gliederungsansicht zeigt, und es so erlaubt, den Überblick über sehr



mmibuch.de/a/9.1

Abbildung 9.1: Links: Das grundlegende MVC Entwurfsmuster, Rechts: mehrere *Views* auf das gleiche *Model*

lange Texte zu behalten. Ein anderes Beispiel wäre ein 3D-Modellierungsprogramm, das das gleiche 3D-Objekt in verschiedenen Fenstern von verschiedenen Seiten zeigt. In diesem Fall wären die Softwarekomponenten für die verschiedenen Views sogar identisch, lediglich ihre Parameter (insbesondere die Kameraposition) verschieden. In der Praxis verwendet der *Controller* oft Teile des *View* mit und ist mehr oder weniger eng in diesen integriert. Um beispielsweise den Texteditor aus dem ersten Beispiel benutzbar zu machen, muss eine aktuelle Cursor-Position im *View* angezeigt werden, auf die sich alle Eingaben des *Controller* beziehen. Um im 3D-Modellierungsprogramm Objekte selektieren zu können, muss im Fenster des *View* eine Clickposition ermittelt und diese mit dem 3D-Modell geschnitten werden. In jedem Fall ist es jedoch wichtig, das *Model* von den beiden anderen Komponenten sauber zu trennen.

Ein weiteres täglich verwendetes Stück Software, das nach diesem Prinzip arbeitet, ist die zentrale Ansicht des Dateisystems (Windows Explorer, MacOS Finder, Linux KDE Dolphin, ...). Hierbei ist das *Model* das tatsächliche Dateisystem. Es kann sich auf der Festplatte, einem USB-Stick oder im Netzwerk befinden, also physikalisch völlig unterschiedlich beschaffen sein, bietet aber in jedem Fall die gleiche (Software-) Schnittstelle zu den anderen Komponenten an. Als *View* dienen in diesem Fall die verschiedenen Ansichten (Ordner und Icons, Baumdarstellung, Spaltenansicht, ...) mit den jeweils integrierten *Controller* Komponenten, die es ermöglichen, Dateien mit der Maus zu verschieben, mit der Tastatur umzubenennen etc. Als weiteres Beispiel zeigt Abbildung 8.3 in Abschnitt 8.3 auf Seite 89 zwei verschiedene *Views* (Liste von Songs bzw. Album Covers) auf das gleiche *Model*, in diesem Fall eine Musiksammlung.

Aus dem Bereich der Informationsvisualisierung kommt das bereits in Abschnitt 8.5 eingeführte Konzept der **multiple coordinated views** (**MCV**). Hierbei handelt es sich um verschiedene *Views* auf den gleichen Datensatz, die untereinander koordiniert sind. Diese Koordination ermöglicht zusätzliche Formen der Interaktion, insbesondere beim Suchen und Filtern innerhalb des Datensatzes. Abbildung 8.4 auf Seite 92 zeigt ein Beispiel für ein solches Interface. Interaktion kann also, wie in Abschnitt 8.5 bereits beschrieben, dabei helfen, mit großen Informationsmengen umzugehen.

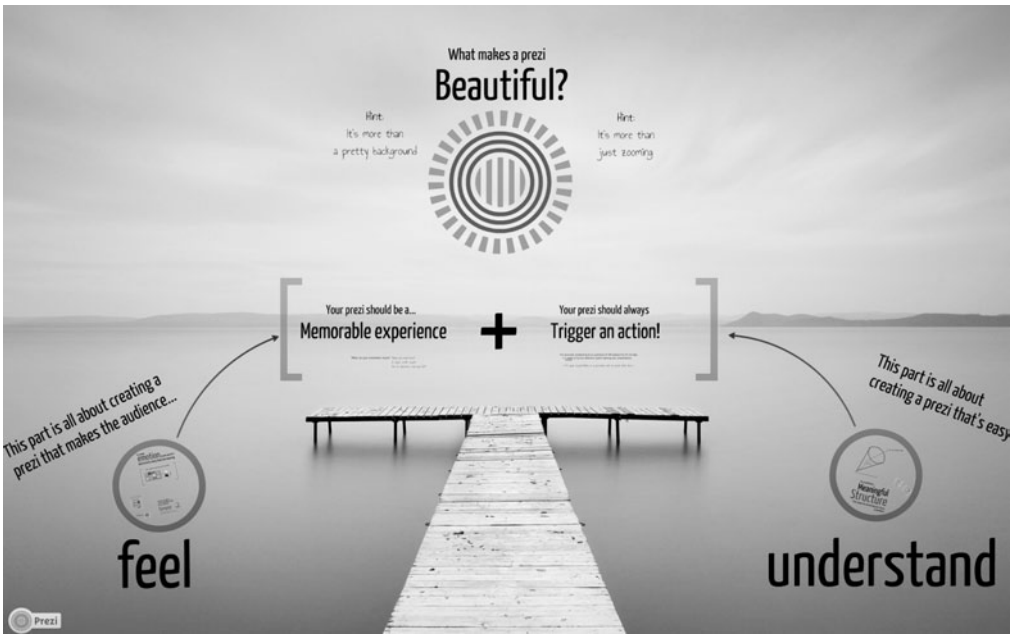
9.2 Zoomable UIs

Ein typisches Problem bei großen Informationsmengen ist naturgemäß der Platzmangel auf dem Bildschirm. Werden alle Bestandteile der Information detailliert dargestellt, dann benötigt die gesamte Darstellung mehr Platz als auf dem Bildschirm verfügbar ist. Verkleinert oder vereinfacht man umgekehrt die Darstellung soweit, dass alles auf den Bildschirm passt, dann reicht der Detailgrad für viele Aufgaben nicht mehr aus. Dieses Problem lässt sich auch nicht durch eine beliebige Erhöhung der Bildschirmauflösung beseitigen (Abschnitt 6.1.1), da dann immer noch die Auflösung des Sehsinns der begrenzende Faktor wäre (Abschnitt 2.1). Für große Datensätze ist es daher notwendig, verschiedene Skalierungen oder Detailgrade anzuzeigen. Dies kann man entweder durch kontinuierliche Veränderung der Skalierung (**geometrischer Zoom**) oder durch die gleichzeitige Darstellung verschiedener Skalierungen (Fokus & Kontext) erreichen.

Das Konzept des **Zoomable User Interface** (**ZUI**) stammt aus den 1990ern und wurde durch **Ken Perlin** und **David Fox** vorgestellt [92] und später durch **George Furnas** und **Ben Bederson** [36] formal genauer gefasst. Während man in den 1990ern das Konzepts eines ZUI noch ausführlich erklären musste, darf man heute getrost davon ausgehen, dass jeder Leser intuitiv damit vertraut ist, da vermutlich jeder schon einmal eine interaktive Kartenanwendung in einem Navigationssystem oder z.B. **Google Maps** verwendet hat. Eine Landkarte ist auch gleichzeitig das perfekte Beispiel für den Nutzen eines ZUI: Landkarten auf Papier werden in verschiedenen Maßstäben verkauft: 1:5.000.000 für den Überblick über gesamte Kontinente, 1:500.000 für ganze Länder und zur groben Routenplanung mit Auto oder Zug, 1:100.000 für Radfahrer und 1:25.000 für Wanderer. Dabei nimmt der Detailgehalt kontinuierlich zu, während das dargestellte Gebiet immer kleiner wird. Ein ZUI mit einer voll detaillierten Weltkarte, wie wir es mit Google Maps benutzen, ermöglicht es nun, alle diese Aufgaben mit einem einzigen System zu erledigen, und beispielsweise auch den exakten Weg durchs Wohngebiet am Ende einer langen Autobahnfahrt zu planen.

Die grundlegenden Operationen in einem ZUI sind die Veränderung des Maßstabes (**geometrischer Zoom**) und die Verschiebung des gezeigten Ausschnitts (**Pan**). Mithilfe dieser beiden Operationen können wir uns durch den gesamten Datenbestand auf allen verfügbaren Detailgraden bewegen. Wir haben auch gelernt, dass die Veränderung des Ausschnitts auf höheren Maßstäben schneller geht, und zoomen daher aus dem Wohngebiet mindestens auf Ebene der Stadt oder des Landkreises, bevor wir den Ausschnitt zu einer anderen Stadt bewegen, in der wir dann wieder auf die Ebene einzelner Straßen hinunterzoomen. Eine genauere Diskussion der möglichen Operationen sowie der dahinter stehenden Mathematik findet sich in Furnas und Bederson [36].

Während sich Landkarten inhärent zum Zoomen eignen, lassen sich auch andere Informationen – geeignet aufbereitet – mittels *Zoom* und *Pan* durchstöbern. Perlin und Fox [92] geben dazu Kalenderdaten und strukturierte Texte als Beispiel an und definieren auch bereits den Begriff des **semantischen Zoom**. Bei diesem wird im Gegensatz zum *geometrischen Zoom* nicht nur der Maßstab der Darstellung verändert, sondern es wird tatsächlich neue Information hinzugefügt oder sogar eine völlig andere Darstellung gewählt. Aus Google maps ist uns das vertraut, da beispielsweise Sehenswürdigkeiten oder Bushaltestellen erst ab einem gewissen Darstellungsmaßstab auftauchen oder die Darstellung von der höchsten Zoomstufe dann zur Ego-Perspektive (Google street view),



mmibuch.de/a/9.2

Abbildung 9.2: Ein ZUI für Präsentationen: Prezi

also einer völlig anderen Darstellung wechselt. Ein anderes zeitgenössisches Beispiel für ein ZUI ist das Präsentationstool Prezi[1] (Abbildung 9.2). Darin werden multimediale Informationen (Text, Bilder, Videos, Animationen) in einer unendlich großen Ebene auf verschiedenen Skalierungsstufen arrangiert. Diese Ebene kann nun frei exploriert werden, oder – was der Normalfall bei einer linearen Präsentation ist – entlang eines festgelegten Pfades. Dabei vermittelt beispielsweise die übergeordnete Struktur auf der obersten Zoomstufe den logischen Zusammenhang der Elemente, die präsentiert werden. Durch Hineinzoomen in einzelne Bereiche der Ebene gelangt man – geometrisch wie logisch – zu einem höheren Detailgrad. Verlässt man diese Detailstufe wieder, um zu einem anderen Bereich zu wechseln, so zoomt die Kamera zurück, verschiebt die Ebene und zoomt an einer anderen Stelle wieder zum Detail. So wird durch die grafische Darstellung gleichzeitig die logische Struktur und die Zusammenhänge zwischen Teilen der Information vermittelt.

[1] http://prezi.com

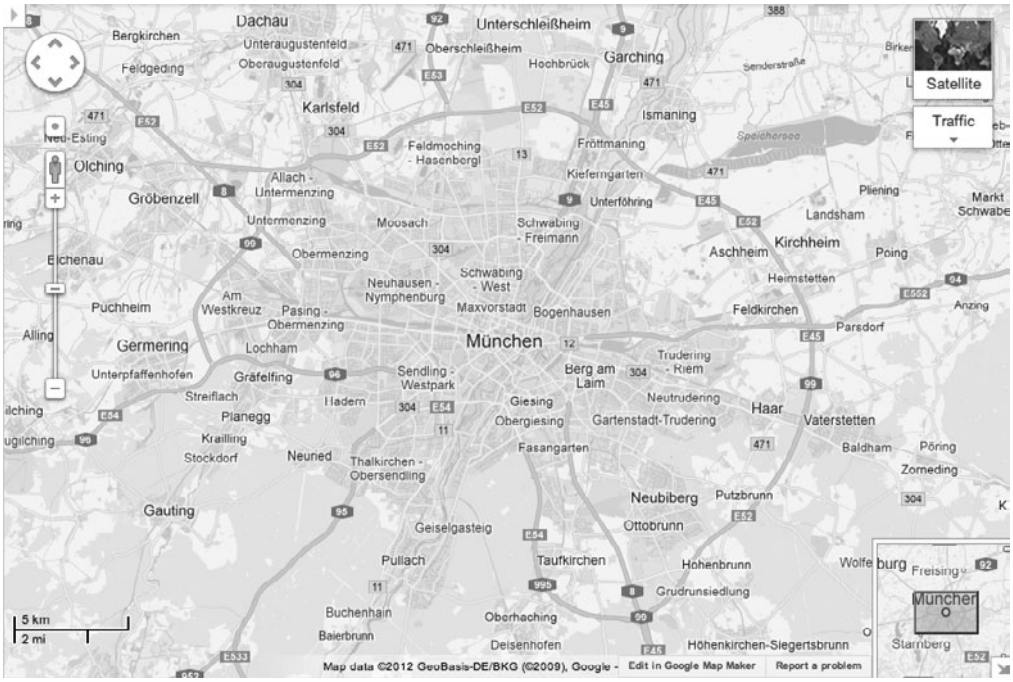
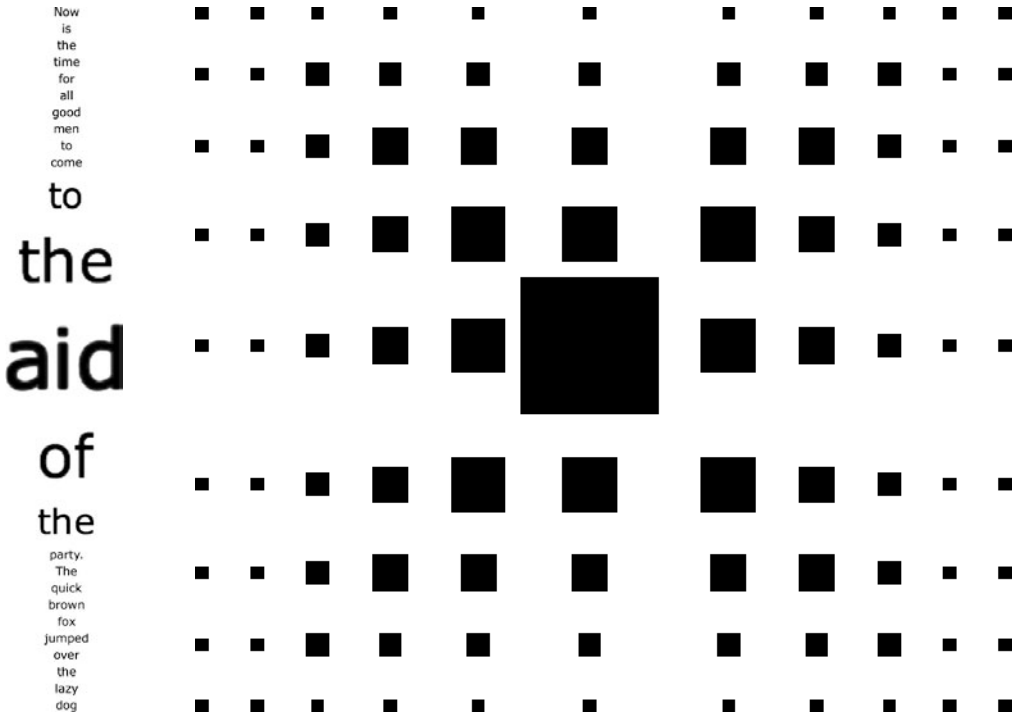


Abbildung 9.3: Einfaches Beispiel für 2 koordinierte Sichten in **Google Maps**, die Fokus (große Karte) und Kontext (Übersichtskarte Unten rechts) gleichzeitig anzeigen.

mmibuch.de/a/9.3

9.3 Fokus & Kontext

Ein typisches Problem von ZUIs ist, dass sie nicht gleichzeitig Überblick und Details vermitteln können. So muss sich der Benutzer ständig zwischen verschiedenen Zoomstufen bewegen, um den Überblick beim Ansehen von Details nicht zu verlieren. Eine andere Möglichkeit ist jedoch, immer zwei Zoomstufen gleichzeitig anzuzeigen. Dabei zeigt eine Darstellung den gegenwärtig ausgewählten Ausschnitt (**Fokus**) auf hoher Detailstufe und die andere Darstellung den gesamten **Kontext** rund um den *Fokus*. Im *MVC* Entwurfsmuster lässt sich dies einfach durch 2 verschieden parametrisierte *views* auf dasselbe *model* implementieren. Abbildung 9.3 zeigt eine solche Darstellung von Kartenmaterial: Die große Karte zeigt München mit seinen wichtigsten Verkehrsadern, während die kleine Übersichtskarte Unten rechts das Münchner Umland mit benachbarten Städten zeigt. Der Ausschnitt kann in beiden verschoben werden und das dunkle Rechteck in der Übersichtskarte zeigt, welcher Bereich in der Detailkarte gezeigt wird. Eine solche Darstellung wird auch **overview+detail** Darstellung genannt und ist eine häufig verwendete Möglichkeit, *Fokus* und *Kontext* gleichzeitig darzustellen. Eine andere Möglichkeit, *Fokus* und *Kontext* gleichzeitig darzustellen, stammt ebenfalls aus dem



mmibuch.de/a/9.4

Abbildung 9.4: Fisheye-Darstellung einer Liste von Wörtern (Links) sowie eines Rasters von Quadraten (Rechts)

Bereich der Informationsvisualisierung und wurde auf die Darstellung von Listen oder Menüs übertragen. Die **Fisheye** Darstellung erhielt ihren Namen vom fotografischen Fisheye Objektiv, einer Optik, die Dinge in der Bildmitte stärker vergrößert als am Bildrand. So entsteht ein geometrisch verzerrtes Bild, das die Welt in der Bildmitte in einem anderen Abbildungsmaßstab wiedergibt als am Rand. Diese Idee wurde zunächst auf die Visualisierung von Grafen und Netzwerken angewendet, später aber auch auf Listen und Menüs. Abbildung 9.4 Links zeigt eine Liste von Wörtern. Darin ist ein bestimmtes Wort in den *Fokus* gerückt und daher vergrößert. Es kann so besser gelesen oder ausgewählt werden. Oberhalb und unterhalb nimmt der Vergrößerungsfaktor kontinuierlich bis zu einem festen Minimum ab, wodurch die benachbarten Wörter (*Kontext*) zunehmend kleiner erscheinen. Abbildung 9.4 Rechts zeigt das gleiche Konzept, übertragen auf ein zweidimensionales Raster aus Quadraten. Die *Fisheye* Darstellung findet sich generell dort, wo es gilt, auf begrenztem Raum viel Information auf verschiedenen Detailstufen unterzubringen, beispielsweise in Menüs von Mobiltelefonen, oder bei der *Dock* genannten Startleiste in OSX.

Übungsaufgaben:

1. Benennen Sie anhand der Abbildung 9.3 die Komponenten der zu Grunde liegenden MVC-Architektur von Google Maps. Diskutieren Sie die Vor- und Nachteile des semantischen Zooms, der bei Google Maps implementiert wurde. Welche unterschiedlichen Zoom- und Pan-Kontrollmöglichkeiten wurden durch die Entwickler vorgesehen und warum?
2. Vergleichen Sie die Funktionalität von elektronischen Karten und klassischen Papierkarten. Welche Vor- und Nachteile haben beide Varianten? Benennen Sie eine Situation, in der Sie bewusst eine Papierkarten verwenden würden und erläutern Sie warum.
3. Recherchieren Sie verschiedene Arten, eine geometrische **Fisheye** Darstellung zu berechnen. Beschreiben Sie zumindest drei dieser Arten und geben Sie an, wofür diese jeweils besonders gut oder schlecht geeignet sind.