TECHNISCHE UNIVERSITÄT DRESDEN

FAKULTÄT INFORMATIK
INSTITUT FÜR SOFTWARE- UND MULTIMEDIATECHNIK
PROFESSUR FÜR COMPUTERGRAPHIK UND VISUALISIERUNG
PROF. DR. STEFAN GUMHOLD

Bachelor

3D-Interface für das Naked-Objects-Framework

Albrecht Uhlig
(Mat.-Nr.: 2805055)

Betreuer: Dipl.-Inf. (FH) Benjamin Neidhold, Dipl.-Inform. Steffen Gemkow

Dresden, 23. September 2006

Aufgabenstellung

Für das NakedObjects Framework soll eine neue Visualisierung entstehen. Insbesondere sind dabei die Möglichkeiten von 3D-Grafik sinnvoll einzusetzen und zu bewerten. Hierfür gibt es bereits verschiedene Desktopsysteme, die 3D-Grafik unterstützen, wovon eines verwendet werden soll.

Inhaltsverzeichnis

1 Einleitung

Fast jeder Computer verfügt heutzutage über 3D-Beschleunigungshardware. Abgesehen von einigen Speziallösungen (AutoCAD) wird diese oft nur in Computerspielen richtig genutzt. Die neueren Desktop-Betriebssysteme nutzen bereits den Grafikprozessor. Können Geschäftsanwendungen davon profitieren? Diese Frage soll anhand eines Frameworks für Geschäftsanwendungen untersucht werden.

Die Bedienung von Computern hat sich im Laufe der Zeit stark verändert. Die ersten Computer konnten nur von Spezialisten bedient werden. Heute soll jeder, der ein fachliches Problem lösen muss, in der Lage sein, den Computer mit all seinen Möglichkeiten zu nutzen. Das bedeutet, dass Anwendungsprogramme intuitiv, konsistent und logisch sein müssen. Studien haben gezeigt, dass Anwender die Hälfte ihrer Zeit an schlechte Oberflächen verlieren [CLB$^+$]. Es ist umstritten, ob dreidimensionale Bedienschnittstellen hierbei einen Vorteil gegenüber zweidimensionalen Ansätzen bringen können.

Speziell unter dem Gesichtspunkt der neuen Möglichkeiten durch 3D-Beschleunigungshardware, soll in dieser Arbeit nach Verbesserungen der Oberfläche des NakedObjects-Frameworks gesucht werden. Wie kann die Anwendung von den neuen Möglichkeiten profitieren, um es dem Anwender einfacher und angenehmer zu machen? Er soll seine Aufgaben mit sowenig Einarbeitung wie möglich erfüllen können, ohne dass er viel Zeit beim Suchen einer Funktionalität verliert. Das ausgewählte Framework (NakedObjects) verfolgt einen neuen, noch wenig erforschten, objektorientierten Ansatz, der sich gut für eine dreidimensionale Schnittstelle zu eignen scheint.

2 Gliederung der Arbeit

Die ersten drei großen Kapitel (3, 4 und 5) behandeln die Grundlagen, die zur Arbeit gehören. Jedes dieser Kapitel könnte auch zu einer eigenen Arbeit ausgedehnt werden, kann also nur einen Einstieg und Überblick bieten. In Kapitel 6 wird mit dem erarbeiteten Wissen ein Konzept für die NakedObject-Oberfläche erarbeitet.

Kapitel 3 behandelt Grundlagen von Mensch-Maschine-Schnittstellen. Es wird auf die psychologischen und die technischen Grundlagen eingegangen. Der psychologische Teil untersucht, wie Menschen wahrnehmen und handeln. Der technische Teil stellt Hardware und Methoden (Interaktionsmodelle, Metaphern) vor.

Neuere Desktopsysteme integrieren 3D-Funktionalität. Kapitel 4 beschreibt drei solche Systeme. Die Windows-Presentation-Foundation von Windows-Vista (kurz WPF), Quartz von Mac-OSX und Looking-Glass. Alle drei verwenden 3D-Grafikhardware. Während WPF und Quartz die neue Grafikhardware hauptsächlich nutzen, um den klassichen 2D-Fensteransatz zu verschönern, versucht Looking-Glass einen echten 3D-Desktop zu erzeugen. Deshalb eignet es sich besonders als Plattform für den praktischen Teil der Arbeit.

Das NakedObjects-Framework setzt einen sehr interessanten Ansatz für Geschäftsanwendungen um. Welche Ideen dahinter stecken und wie es arbeitet, erklärt das Kapitel 5. Es wird mit anderen Lösungen verglichen und die praktischen Einsatzmöglichkeiten werden diskutiert.

Das letzte Kapitel 6 untersucht zunächst die Schwächen der bisherigen NakedObjects-Oberfläche. Danach werden bestehende 3D-Systeme vorgestellt. Im letzten Teil wird ein Konzept vorgestellt, das versucht die Schwächen zu beseitigen unter Verwendung von dreidimensionaler Visualisierung.

3 Grundlagen von Mensch-Maschine-Schnittstellen

In dieser Arbeit wird oft von **intuitiven** Schnittstellen gesprochen. Es ist sehr wichtig, den Begriff Intuition richtig zu benutzen, da er leicht zum falsch benutzten Schlagwort wird. Allgemein versteht man unter Intuition unbewusste richtige Eingebungen [wikn]. Bei Computern wird der Begriff aber eher im Sinne von „bereits bekannt, oder ähnlich zu etwas Bekannten“ eingesetzt [Ras]. Die meisten Menschen, die meinen, etwas sei selbstverständlich und intuitiv, haben nur vergessen, dass sie es irgendwann gelernt haben. Wann eine Schnittstelle intuitiv ist, hängt davon ab, was der aktuelle Anwender für Vorkenntnisse besitzt. Diese Vorkenntnisse können auch von schlechten Schnittstellen stammen, intuitiv ist also nicht immer auch gut.

Kapitel 3.1 gibt einen kurzen Einblick die psychologischen Grundlagen, die für Mensch-Maschine-Schnittstellen interresant sind. Zuerst wird betrachtet wie Menschen wahrnehmen. Daraus lassen sich einige Gesetze für gute Gestaltung ableiten. Kapitel 3.1.2 gibt einen Überblick dazu wie Menschen handeln und aus welchen Motivationen herraus sie es tun. Damit der Anwender beim Arbeiten nicht behindert wird, ist es wichtig, ihm die richtigen Handlungsmöglichkeiten verständlich anzubieten. So kann er seine Aufgaben und Probleme ohne Hilfe lösen.

3.1 Psychologische Grundlagen

Die psychologischen Grundlagen von Mensch-Maschine-Schnittstellen sind sehr kompliziert und vielfältig. Eine ausführliche Betrachtung würde den Rahmen dieser Arbeit sprengen. Einige Grundlagen sind jedoch ebenso einfach wie wichtig, wenn man intuitive Schnittstellen entwickeln will. Im folgenden geht es um Wahrnehmung, Aufmerksamkeit und Handlung. Sehr gute Quellen für einen umfangreicheren Einstieg sind die Internetseite [kom] und das Buch [Wir04]. Einen wissenschaftlicheren Einstieg bietet z.B. [SS97].

Abbildung 3.1 zeigt schematisch das Vorgehen eines Menschen vor einer unbekannten grafischen Schnittstelle. Es erscheint extrem einfach und trivial, aber genau dieser Kreislauf ist es, der immer wieder abläuft. Zuerst wird beobachtet, dann gehandelt. Nach dem Handeln wird wieder beobachtet, ob oder welches Resultat das Handeln hatte. Der Antrieb für diesen Kreislauf kann ein bestimmtes Ziel sein

(z.B. Laden einer Datei) oder auch einfach Neugier. Für Geschäftsanwendungen ist besonders der erste Fall, ein konkretes Ziel, interessant[1]. Die Schnittstelle der Anwendung muss so entwickelt sein, dass der Nutzer möglichst wenig Kreisläufe benötigt, um sein Ziel zu erreichen.

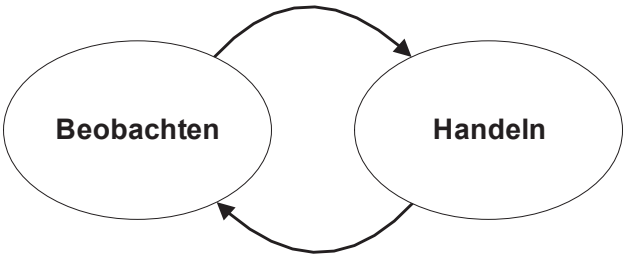

Abbildung 3.1: Handlungen eines Nutzers vor einer unbekannten grafischen Schnittstelle.

Der folgende Abschnitt untersucht die linke Seite des Kreislaufes genauer. Wie nehmen Menschen wahr und was lenkt ihre Aufmerksamkeit. Danach wird die rechte Seite näher betrachtet. Wie kommt es über Motivation zum Entscheiden und Handeln? Der letzte Abschnitt gibt ein Zusammenfassung.

3.1.1 Wahrnehmung und Aufmerksamkeit

Es gibt eine Reihe erstaunlich einfacher Regeln zur Wahrnehmung von Menschen. Viele davon empfindet man als selbstverständlich, wenn man von ihnen liest. Trotzdem werden sie oft nicht beachtet. Wahrscheinlich liegt das daran, dass ein Entwickler den Distanzblick zu seinem Produkt verliert, wenn er lange damit arbeitet. Es hilft in diesem Fall, die Grundlagen in Gesetzen zu formulieren. So ist es leichter zu prüfen, ob man keine groben Fehler gemacht hat. Hinzu kommt, dass ein wissenschaftlich belegtes Gesetz mehr wert ist, als eine plausibel klingende Regel.

Im folgenden werden einige Gesetze aus der Wahrnehmungspsychologie aufgeführt, die für grafische Schnittstellen sehr wichtig sind. Diese Gesetze sind der Gestaltpsychologie zuzuordnen und entstanden anfang des 20. Jahrhunderts in Berlin [wikg].

Gesetz der Nähe

Gesetz 1 (Nähe) *Räumlich naheliegende Dinge werden als zusammengehörend erkannt. Ungenutzter Platz ist also nicht unbedingt Verschwendung, sondern kann auch ein Gestaltungsmittel sein.*

[1] Es gibt auch Artikel, die sich der Frage stellen, wieso seriöse Anwendungen immer ernst sein müssen, z.B. [Sch04].

Abbildung 3.2 zeigt eine Tabelle über Mitarbeiterzahlen eines Unternehmens, aufgeteilt nach Abteilung und Geschlecht. Die Zahlen, die zu einer Abteilung (z.B. Entwicklung) gehören, sind weit voneinander entfernt, wärend der Abstand zwischen den Abteilungen (z.B. die W-Spalte von Entwicklung und die M-Spalte von Vertrieb) gering ist. Das Gesetz der Nähe ist damit verletzt, weil zusammengehörende Zahlen einen grösseren Abstand haben, als nicht zusammengehörende. Abbildung 3.3 zeigt die gleiche Tabelle mit einem besseren Layout. Dadurch erhört sich die Lesbarkeit der Tabelle.

Jahr	Entwicklung		Vertrieb		Support	
	M	W	M	W	M	W
2000	1	0	1	0	0	0
2001	2	1	2	0	0	2
2002	3	1	2	1	1	2
2003	3	0	3	1	2	4
2004	2	0	1	2	3	5

Abbildung 3.2: Eine Tabelle über Mitarbeiterzahlen eines Unternehmens. Das Gesetz der Nähe wird hier verletzt.

Jahr	Entwicklung		Vertrieb		Support	
	M	W	M	W	M	W
2000	1	0	1	0	0	0
2001	2	1	2	0	0	2
2002	3	1	2	1	1	2
2003	3	0	3	1	2	4
2004	2	0	1	2	3	5

Abbildung 3.3: Die gleiche Tabelle wie in Abbildung 3.2. Hier wird das Gesetz der Nähe nicht verletzt. Die Lesbarkeit ist deutlich besser.

Gesetz der Ähnlichkeit

Gesetz 2 (Ähnlichkeit) *Dinge, die sich in wichtigen Merkmalen ähneln, werden als zusammengehörend wahrgenommen. So ein Merkmal kann die Form sein oder auch die Farbe.*

Abbildung 3.4 zeigt die gleiche Tabelle wie beim Gesetz der Nähe. Durch Einfärben[2] der Spalten wird

[2]Beim Arbeiten mit Farbe muss man sehr vorsichtig sein. Farben sind auf manchen Ausgabegeräten (z.B. Beamer) schwer

das Gesetzt der Ähnlichkeit verwendet, um zu verdeutlichen, welche Zahlen zusammengehören. Obwohl das Gesetz der Nähe immer noch verletzt ist, ist die Tabelle damit besser lesbar, als in Abbildung 3.2.

Jahr	Entwicklung		Vertrieb		Support	
	M	W	M	W	M	W
2000	1	0	1	0	0	0
2001	2	1	2	0	0	2
2002	3	1	2	1	1	2
2003	3	0	3	1	2	4
2004	2	0	1	2	3	5

Abbildung 3.4: Bei dieser Tabelle sind die zusammengehörenden Spalten eingefärbt. Dadurch heben sie sich von den benachbarten besser ab, obwohl das Gesetz der Nähe verletzt wird.

Gesetz der Geschlossenheit

Gesetz 3 (Geschlossenheit) *Dinge, die von Linien umschlossen werden, erscheinen zusammengehörend. Das kann auch eine einzelne Trennlinie sein. Man muss also vorsichtig sein, wenn man aus optischen Gründen eine Linie einfügt, und prüfen, ob die Linie nicht etwas zusammengehörendes spaltet.*

Ein einfaches Beispiel ist jede Tabelle. Die bisherigen Beispieltabellen hatten bewusst keine Trennlinien. Abbildung 3.5 zeigt die Tabelle aus Abbildung 3.2 mit Trennlinien. Das Gesetz der Nähe wird immer noch verletzt, die Trennlinien sorgen aber dennoch für Übersicht. An diesem Beispiel wird bereits sichtbar, dass jedes der Gesetze für sich alleine sehr einfach ist. Wenn sich die Effekte aber gegenseitig überlagern, ist nicht sofort klar, wie das Layout verbessert werden kann.

Es müssen nicht immer Linien zur Trennung verwendet werden. Viele kleine Objekte können auch als Linie wahrgenommen werden. Abbildung 3.6 zeigt links eine Menge von Punkten, die aber als zwei Linien wahrgenommen werden. Rechts wird ein Beispiel für das Phänomen der subjektiven Konturen gezeigt. Es werden Teile der Umrisse eines Dreiecks dargestellt. Diese unvollständige Information reicht aus, um ein weißes Dreieck sehen. Durch dieses Phänomen kann man, z.B. durch angedeutete Konturen, Objekte gruppieren, ohne sie komplett einzurahmen.

lesbar, haben in verschiedenen Kulturen unterschiedliche Bedeutung und können von vielen Menschen (z.B. mit Rot-Grün-Schwäche) nicht richtig wahrgenommen werden. Mehr Informationen über Farben sind bei [kom] und [Wir04] zu finden.

Jahr	Entwicklung		Vertrieb		Support	
	M	W	M	W	M	W
2000	1	0	1	0	0	0
2001	2	1	2	0	0	2
2002	3	1	2	1	1	2
2003	3	0	3	1	2	4
2004	2	0	1	2	3	5

Abbildung 3.5: Eine Tabelle mit Linien zur Trennung der Spalten. Das Gesetz der Nähe wird immer noch verletzt. Die Linien sorgen jedoch trotzdem für gute Lesbarkeit.

Abbildung 3.6: Links: Viele kleine Objekte können auch als Linie wahrgenommen werden. Rechts: Teile einer Kontur sind ausreichend, um die Illusion eines weißen Dreiecks zu erzeugen.

Gesetz der Einfachheit

Gesetz 4 (Einfacheit) *Dinge werden so wahrgenommen, dass sie einfach erscheinen und leicht zu beschreiben sind. Wenn es also 2 mögliche Interpretationen eines Bildes gibt, wird wahrscheinlich die einfachere gewählt.*

Das Gesetz der Einfachheit lässt sich sehr leicht an zusammengesetzten Bildern zeigen. Es gibt viele Möglichkeiten, ein zusammengesetztes Bild zu beschreiben. Meist wird die einfachste zuerst erkannt. Abbildung 3.7 zeigt ein Beispiel. Links ist eine aus zwei Objekten zusammengesetzte Figur zur erkennen. Rechts sind zwei Möglichkeiten gezeigt, aus welchen Objekten das Bild aufgebaut sein könnte. Die obere Variante lässt sich sehr einfach in Worte fassen, ein Rechteck und ein Kreuz. Die untere Variante lässt sich nur sehr schwer beschreiben und wäre wahrscheinlich, selbst mit mehreren Sätzen als Beschreibung schwer vorstellbar. Wie stark der Drang zur einfachen Wahrnehmung ist, kann man sich verdeutlichen, indem man nur die linke Zeichnung betrachtet. Es fällt sehr schwer, dort kein weißes Kreuz zu sehen.[3].

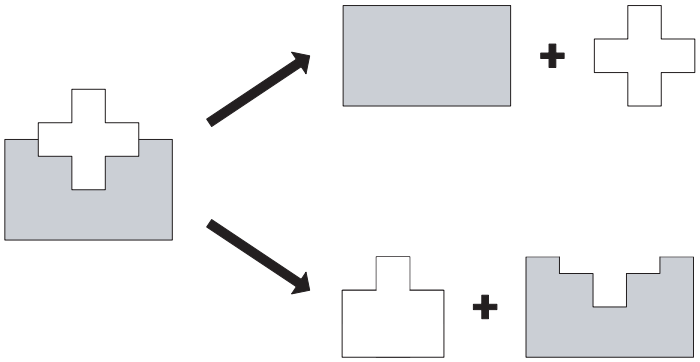

Abbildung 3.7: Es gibt verschiedene Möglichkeiten, die linke Figur zusammenzusetzen. Zwei davon sind rechts abgebildet. Das Bewusstsein entscheidet sich für die einfachste Möglichkeit, oben rechts dargestellt.

Das hier beschriebene Phänomen ist schwieriger zu verstehen und zu nutzen als die bisherigen. Es zeigt, dass das Sehen kein objektiver Vorgang ist, sondern man sich nicht gegen eine Interpretation des Gesehenen wehren kann. Abbildung 3.5, die bereits beim Gesetz der Geschlossenheit besprochen wurde, ist auch ein Beispiel dafür, dass man Dinge sieht, die eigentlich nicht da sind. Das Bewusstsein entscheidet sich automatisch für genau eine Interpretation des Gesehenen. Es kann nicht zwei Bedeutungen gleichzeitig verarbeiten.

[3]Mit einem Bleistift lässt sich das Phänomen weiter beobachten. Es hilft z.B. die Farben zu tauschen, dann fällt es leichter, das Kreuz als hinter dem Rechteck liegend zu sehen. Genauso hilft es, das Kreuz zu verfremden, oder zuerst das treppenförmig ausgeschnittene Rechteck zu zeichen.

Eine sehr interessante Schlussfolgerung für diese Arbeit ergibt sich aus dem Gesetz der Einfachheit. Auch auf zweidimensionalen Medien kann es sehr nützlich sein, 3D-Objekte darzustellen. Obwohl man nur ein zweidimensionales Bild sieht, entsteht im Bewusstsein eine dreidimensionale Vorstellung. Diese Erkenntnis ist wichtig, weil man durchaus den Sinn von 3D-Darstellungen am Computerbildschirm anzweifeln kann. Der Bildschirm ist ein 2D-Medium. Warum soll man nicht warten, bis 3D-Ausgabgeverfahren, wie sie Kapitel 3.2.1 vorstellt, zum Standard werden? Das Gesetz der Einfachheit gibt einen Grund, der zumindest rechtfertig, dass 3D-Darstellungen auf 2D-Bildschirmen einen Vorteil haben können. Auto-CAD-Software und 3D-Modellierungstools sind Beispiele für sinnvollen Einsatz der dritten Dimension. Man könnte aber z.B. auch die Rückgängig-machen und Wiederholen-Funktionalität einer Anwendung in die dritte Dimension legen. Indem der Anwender sein Dokument nach vorne und hinten verschiebt, bewegt er es in der Zeit vor und zurück.

Es lassen sich noch mehr Schlussfolgerungen aus dem Gesetz der Einfachheit finden. Wenn man beim Design einer Oberfläche ein Ziel erreicht hat, sollte man keine weiteren Effekte hinzufügen. Das klingt trivial, wird aber von Designern, die mehr den optischen Eindruck im Blick haben, oft missachtet. Wenn z.B. ein Objekt selektiert wird, soll der Benutzer es auch als selektiert erkennen, es muss sich von anderen Objekten unterscheiden. Üblicherweise reicht eine andere Farbe oder ein Rahmen aus, um das zu erreichen. Ein Designer könnte alle seine Fähigkeiten einsetzen, indem er das Objekt animiert, leuchten lässt, bunt macht, vergrössert, mit einem Schatten versieht und vieles mehr. Damit würde er weit über sein Ziel, dem Nutzer zu zeigen, welches Objekt selektiert ist, hinausgehen. Wahrscheinlich würde sogar die Aufmerksamkeit zu sehr auf das Objekt gelenkt, was den Anwender von seinem eigentlichem Ziel ablenkt.

Das Gesetz der Einfahheit scheint universell zu sein und sich nicht nur auf Wahrnehmung zu beschränken. Bücher wie der Pragratische Programmierer [HT03] enthalten ähnliche Kernaussagen. Slogans wie „Simple is beautiful“ oder „Keep it simple“ sagen im Prinzip das gleiche aus. Es lässt sich schwer beweisen, aber praktische Erfahrungen zeigen immer wieder, dass einfache Lösungen besser sind.

Aufmerksamkeit

Im diesem Abschnitt geht es darum, warum und wie man sich entscheidet, manche Dinge wahrzunehmen und andere zu ignorieren. Besonders bei Anwendungen, die der Nutzer nicht gut kennt, ist es wichtig zu wissen, wie man die Aufmerksamkeit steuert. Er soll Wichtiges finden, ohne permanent abgelenkt zu werden. Besondere Aufmerksamkeit will man normalerweise für Dinge wie selektierte Objekte, Warnhinweise, Fehlermeldungen oder Tips erreichen. Fast noch wichtiger ist es, den Nutzer nicht abzulenken, also seine Aufmerksamkeit von seiner Arbeit abzuziehen, indem z.B. Unwichtiges hervorgehoben bleibt

oder unerwartetes passiert.

Was Aufmerksamkeit genau ist und wie sie funktioniert, kann hier nur oberflächlich beantwortet werden. Das Modell der präattentiven Prozesse ist ausreichend, um die meisten Effekte zu erklären und ist in Abbildung 3.8 dargestellt. Der blaue Kasten links, stellt alles dar, was die Sinne wahrnehmen. Diese Eindrücke werden automatisch analysiert (gelber Kasten). Auf diese Analyse hat man keinen Einfluss. Wenn z.B. jemand den eigenen Namen ruft, kann man das nicht ignorieren. Der rote Kasten ist das, was man auch als Aufmerksamkeit bezeichnen kann. Man wählt Wahrnemungen aus, die dann vom Bewusstsein verarbeitet werden. Auf welche Art und Weise dieses Auswählen geschieht, hängt von den Zielen ab die man hat. Wenn keine konkreten Ziele vorhanden sind, wird eine grosse Menge an Information oberflächlich verarbeitet. Psychologen nennen das schwebende Aufmerksamkeit. Ist ein klares Ziel vorhanden, spricht man von fokusierter Ausmerksamkeit. Es wird dann gezielt nach bestimmten Merkmalen gefiltert, wobei alles andere ignoriert wird. Ein Beispiel für die Leistungsfähigkeit dieses Filters ist eine Party, bei der sehr viele Menschen gleichzeitig reden. Durch das Ziel, einer bestimmten Person zuzuhören, ist man in der Lage, alle anderen Stimmen zu ignorieren und nur eine zu verstehen. Trotzdem arbeitet die automatische Analyse weiter. Wenn z.B. der eigene Name gerufen wird oder das eigene Handy klingelt, verringert sich die Aufmerksamkeit dem Gesprächspartner gegenüber. Dieses Modell der Aufmerksamkeit wird auch Cocktail-Party-Effekt genannt und geht auf Arbeiten von Colin Cherry und Donald Broadbent in den 50ern zurück [**?**].

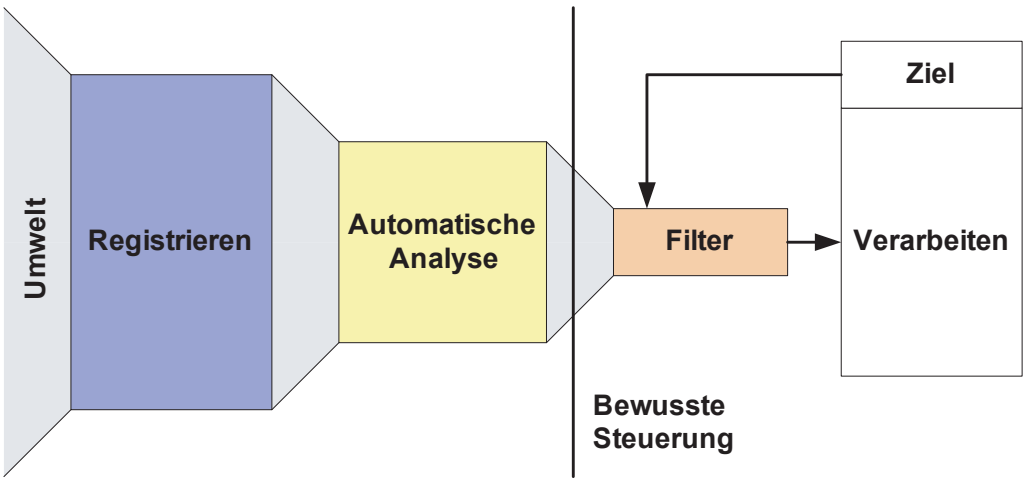

Abbildung 3.8: Modell der präattentiven Prozesse: Vor dem bewusst gesteuerten Filtern der Wahrnehmung, gibt es eine automatische Analyse.

In Abbildung 3.8 kann man auch den einfachen Kreislauf aus Abbildung 3.1, vom Anfang des Kapitels, wiederfinden. Das Handeln resultiert aus dem Verarbeiten der Wahrnehmung. Die Wahrnehmung registriert dann die Ergebnisse des Handelns.

Die Aufmerksamkeit lässt sich nicht vollständig steuern. Es ist eine Frage der Gewohnheiten, was man automtisch wahrnimmt. Das bedeutet, im schlimmsten Fall ist es möglich, die Aufmerksamkeit eines Nutzers ständig zu stören, ohne dass er etwas dagegen tun kann. Ein animiertes Werbebanner, zum Beispiel, stört die Konzentrationsfähigkeit des Anwenders. Jeder Internet-Nutzer lernt zwar, Werbebanner zu ignorieren, die biologisch programmierte Gewohnheit, dass man auf Bewegung reagiert, hat sich aber über Jahrtausende entwickelt und kann nicht so leicht unterdrückt werden. Obwohl das Bewusstsein also lernt, dass Animation im Internet meist Unwichtiges darstellt, wird man trotzdem beim Erreichen der Ziele gestört. Es gibt andere Gewohnheiten, die weniger fest sitzen. Wenn z.B. etwas gelernt wird und sehr oft angewendet, wird es zur Gewohnheit. Ein typisches Beispiel ist das Lesen. Wenn man ein einzelnes Wort sieht, muss man es lesen, selbst wenn man es nicht will. Das liegt daran, dass das Lesen überlernt ist. Auf der Internetseite [kom] kann man sich bei einer Demonstration des sogenannten Strop-Effektes davon überzeugen, das es solche überlernte Fähigkeiten gibt. Wenn man also die Aufmerksamkeit des Anwenders vorhersagen und lenken will, muss man wissen, was für Gewohnheiten er hat. Ausserdem wird es einfacher, je weniger Information man gleichzeitig darstellt und je einfacher die Struktur der Anwendung ist.

Für die Paxis gibt es einige Gesetze, was Aufmerksamkeit erregt. Die wichtigsten sind die der Intensität, der Ausnahme und der Dissonanz:

Gesetz 5 (Intensität) *Je intensiver ein Reiz ist, desto mehr Aufmerksamkeit erzeugt er.*

Gesetz 6 (Ausnahme) *Alles was von der Regel abweicht, fällt auf*

Gesetz 7 (Dissonanz) *Nicht erfüllte Erwartungen erzeugen „wundern“ und damit Aufmerksamkeit.*

Da sich diese Effekte gegenseitig beeinflussen, empfiehlt es sich, sie nicht gleichzeitig einzusetzen. Sie könnten miteinandern konkurieren. Das wird vom Betrachter dann meist als verwirrend empfunden. Wenn man einheitliche Mittel zur Aufmerksamkeitssteuerung einsetzt, spricht man von einer Aufmerksamkeits-Sprache. Unter Windows werden selektierte Objekte z.B. oft mit einem Blauen Hintergrund versehen. Es wird also die Intensitäts-Regel (durch höheren Kontrast und Farbe) und die Ausnahme-Regel (alle anderen Objekte sind nicht blau) benutzt. Ein weiteres Beispiel ist das Ausgrauen von inaktiven Buttons und Menueinträgen (Intensitäts-Regel). Eine Anwendung kann eine vorhande Aufmerksamkeits-Sprache verwenden oder eine eigene erzeugen. Wichtig ist vor allem, dass es konsistent (einheitlich) ist.

3.1.2 Motivation und Handeln

Was genau einen Menschen antreibt und wie er auf Situationen reagiert, kann nicht genau vorhergesagt werden. Ausgangspunkt ist meist eine Motivation. Psychologen verstehen unter einer Motivation, eine relativ stabile Persönlichkeitseigenschaft, die in einer Vorliebe für bestimmte Arten von Zielen zum Ausdruck kommt. Motivation führt also zu Zielen, die man erreichen will. Es gibt Motivationen, die auf physischen Vorgängen beruhen, z.B. Nahrungsaufnahmen, und solche, die auf psychologischen Vorgängen beruhen wie z.B. Neugier. Jeder Mensch hat eine Reihe von Basismotivationen, die sich über Jahrtausende entwickelt haben. Anhang A.2 zeigt eine Tabelle mit 16 Basismotivationen.

Für Geschäftsanwendungen ist es nicht wichtig, aus welcher Motivation heraus ein Anwender handelt. Eine Geschäftsanwendung wird vom Anwender genutzt, weil sie beim Erreichen einer bestimmten Menge von Zielen helfen soll. Es genügt also, sich mit den Zielen und den Handlungen zum Erreichen der Ziele, zu beschäftigen. Wie gross die Bereitschaft ist, Hindernisse beim Erreichen der Ziele zu überwinden, hängt von vielen Faktoren ab, z.B. das Image des Produktes oder die Müdigkeit und der aktuelle Gefühlszustand des Anwenders.

Ein Ziel verbindet sich bei einem Computerprogramm mit einem Anfangszustand, einem gewünschten Endzustand und mehreren Zwischenzuständen. Um die Zustände zu verändern, gibt es Operatoren, das sind z.B. Buttons, Menus, Slider und andere Komponenten. Wenn es Hindernisse beim Erreichen der Ziele gibt, spricht man von Problemen. Probleme kann es beim Finden der Operatoren, beim Anwenden dieser und beim Finden der Ziele geben. Wenn ein Buttons z.B. nichts Ersichtliches verändert oder aus nicht erkennbaren Gründen deaktiviert ist, entsteht so ein Problem. Der Operator Button kann nicht richtig verwendet werden und es muss ein anderer Weg gefunden werden.

Wie stark ein Benutzer bereit ist, sich mit einer Anwendung zu beschäftigen, hängt auch von seiner Gefühlslage ab. Wenn er bereits negative Erfahrungen mit dem Produkt gemacht hat, wird er schneller aufgeben, falls es Probleme gibt. Gerade bei negativen Erlebnissen ist das Gedächtnis besonders gut.

Wie hartnäckig sich so ein negatives Image halten kann, sieht man an Microsoft-Windows. Obwohl die einzelnen Versionen eindeutig besser geworden sind in den letzten Jahren, bleibt das schlechte Image erhalten. Anhang A.1 zeigt eine Tabelle, die dabei helfen kann, schlechte Erlebnisse beim Anwender zu verhindern.

Zum Lösen von Problemen gibt es verschiedene Strategien. Beim Prinzip Divide-and-Conquer (Teile und Herrsche) wird das Problem in kleine Teilprobleme zerlegt. Jedes Teilproblem wird mit Hilfe der Operatoren gelöst. Wenn es keine Hindernisse gibt, kann man jedes noch so grosse Ziel auf diese Art erreichen.

Eine andere Strategie ist das Rückwärtssuchen. Ausgehend vom Ziel, sucht man Zustände, die ihm vorrausgehen. Oft kann man Probleme auch über Analogien lösen, wenn in der Anwendung Metaphern verwendet werden. Bei der Schreibtisch-Metapher kann man z.B. ein Dokument löschen, indem man es in den Papierkorb wirft. Das Ziel Dokument-löschen wird durch eine Analogie zum bekannten Vorgehen beim Beseitigen eines Dokumentes aus der wirklichen Welt erreicht.

Die einfachste Strategie, die sogar von niederen Säugetieren verwendet wird, ist die Methode Trial-and-Error (Versuch und Irrtum). Wenn die Menge der Operatoren zu gross ist, eine systematische Strategie nicht funktioniert hat oder die Berreitschaft zum Nachdenken des Nutzers zu klein ist, wird dieses Verfahren benutzt. Es ist sozusagen die letzte Möglichkeit, sein Ziel zu erreichen, wenn alles andere scheitert. Für Geschäftsanwendungen ist diese Strategie gefährlich, weil man dabei auch sehr viel Schaden anrichten kann. Es muss verhindert werden, dass der Anwender in eine Situation gerät, in der er keine Möglichkeit sieht, sein Ziel zu erreichen. Hilfe-Funktionen und Support-Hotlines sollten normalerweise immer ein besserer Weg zur Problemlösung sein als Versuch und Irrtum.

Damit es nicht zu „Versuch und Irrtum“ kommt, muss man in jedem Programmzustand die möglichen Anwenderziele kennen und entsprechende Operatoren anbieten. In der Praxis gibt es normalerweise zu viele mögliche Ziele, um alle Operatoren übersichtlich darzustellen. Viele Anwendungen bieten deshalb Wizards an, bei denen dem Anwender eine Reihe von Fragen gestellt werden, was er machen will. Eine andere Möglichkeit ist es, verschiedene Awendungs-Modi für Anfänger und Fortgeschrittene anzubieten. Die selten benötigten Operatoren könnten dann für neue Anwender ausgeblendet werden. In der Zukunft sind sogar Anwendungen denkbar, die durch Auswertung von Mausbewegungen automatisch feststellen, wenn ein Anwender Hilfe benötigt.

Welche Operarotren man bereitstellt und wie man sie kennzeichnet, ist wahrscheinlich die grösste Herausfoerderung bei grafischen Oberflächen. Es erfordert genaue Kenntnis über mögliche Ziele und Eigenschaften der Anwender. Menschen lösen jedoch gerne Probleme, wenn es geeignete Operatoren gibt und die Schnittstelle fehlertolerant ist. Fehlertolerant bedeutet z.B., dass Operationen rückgängig gemacht werden können. Die grafische Schnittstelle von Naked-Objects, um die es in dieser Arbeit geht, nutzt diese Erkenntnis. Es werden Operatoren angeboten, mit denen jedes Ziel, das die Anwendung lösen kann, erreichbar ist.

3.2 Technische Grundlagen

In diesem Kapitel geht es um die technischen Umsetzungen von Mensch-Maschine-Kommunikation. Zum einen muss der Mensch dem Computer mitteilen, was er zu machen hat, zum anderen will der

Mensch auch das Resultat wissen. Der erste Teil geschieht, indem man eine Aktion (ein Verb) und ein Objekt (ein Substantiv) auf dem die Aktion ausgeführt wird, wählt. Das geschieht z.B. durch Selektion des Objektes mit einem Zeigegerät und Drücken einer Taste für die gewünschte Aktion. Die Ergebnisse werden durch ein Ausgagegerät präsentiert.

Der erste Unterabschnitt wird Ein- und Ausgabegeräte vorstellen. Das sind vor allem die Tastatur und die Maus. Es werden aber auch einige Alternativen erwähnt. Danach geht es um Interaktionsmodelle. Also wie man den Ablauf der Kommunikation aufbaut. Genauso wie in der Sprache zwischen Menschen, werden auch zwischen Mensch und Maschine Metaphern benutzt. Wie das funktioniert, wird im letzten Abschnitt erklärt.

3.2.1 Ein- und Ausgabegeräte

Viele Ein- und Ausgabegeräte, wie die Maus und die Tastatur, werden von den meisten Nutzern als selbstverständlich angesehen. Es gibt jedoch Alternativen und Verbesserungsmöglichkeiten, die in Spezialberreichen benutzt werden und sich in der Zukunft vielleicht als Standard durchsetzen. Dieses Kapitel gibt einen Überblickt über verschiedene Arten von Tastaturen, Zeigegeräten und neuen experimentellen Lösungen.

Eingabegeräte werden meist mit den Händen bedient und basieren auf Druck oder Bewegung. Spracheingabe wird in speziellen Berreichen, z.B. beim Autofahren, eingesetzt. Visuelle Sensoren werden für Eye-Tracking [wikm] oder Gestenerkennung (z.B. GoMonkey [gom]) eingesetzt. Neuere experimentelle Geräte messen Gehirnströme über Implantate [geh].

Ausgabegeräte sprechen meist das Sehen oder Hören an. Der Tastsinn wird bei Force-Feedback-Geräten und bei Ausgabegeräten für Blinde angesprochen. Geruchs- und Geschmackssinn spielen in der Computerwelt keine Rolle.

Die Tastatur mit dem bekannten QWERTY-Layout [wikk] wird als Haupteingabegerät für Texteingabe bleiben. Es werden aber neue Eingabegeräte hinzukommen und es wird sinnvolle Kombinationen geben. Man kann sich z.B. einen Text vorlesen lassen oder ein Zeigegerät zum Selektieren und ein Sprachbefehl für die Aktion verwenden. Besonders bei mobilen Kleingeräten und in der Unterhaltungs-Industrie (z.B. Spielekonsolen) gibt es viele Neuerungen geben. Beispiele sind projezierte Tastaturen [pro] oder dreidimensionale Zeigegeräte [nin].

Tastaturen

Die Tastatur ist das Haupteingabegerät für Text und wird es auch noch eine Weile bleiben [SP05]. Tastaturen gab es schon lange vor Computern z.B. in Form von Tasteninstrumenten. Gerade bei den Instrumenten ist erkennbar, wie sehr trainingsabhängig das Arbeiten mit Tastaturen ist. Beim Computer reicht die Spanne etwa von einer Taste pro Sekunde bis zu 15 Tasten pro Sekunde.

Es gibt verschiedene Arten von Tastaturen. Bei Kleingeräten werden sie oft nur mit dem Daumen bedient. Für öffentliche Geräte, die sehr robust sein müssen, werden Touchscreen-Tastaturen verwendet. In Spezialgebieten werden Chord-Tastaturen verwendet (Akkord-Tastaturen), die nur durch intensives Training bedient werden können. Tastaturen gibt es in vielen verschiedenen Layouts. Am verbreitesten ist das QWERTY-Layout, das mit kleinen Unterschieden auf der ganzen Welt zu finden ist (Deutschland QWERTZ-Layout, Frankreich AZERTY-Layout,...). Ziel beim QWERTY war es, häufig aufeinanderfolgende Buchstaben weit voneinander zu trennen und möglichst abwechselnd die rechte und die linke Hand zu benutzen. Andere Layouts, wie das DVORAK-Layout, die den Abstand zwischen Buchstabenpaaren verringern, erreichen höhere Wortraten, setzten sich aber aufgrund der Verbreitung von QWERTY und dem Umlernaufwand nicht durch.

Tabelle 3.1 zeigt verschiedene Tastaturarten und die erreichbaren Wort-Pro-Minute Werte nach Training. Die Zahlen stammen alle aus [SP05]. Zahlen aus anderen Quellen können nicht wirklich als Vergleich genommen werden, da es stark vom Training und den Testbedingungen abhängig ist, was für Werte erreicht werden. Die ersten beiden Tastaturen sind Daumentastaturen für Handys. Multitap ist auf fast jedem Mobiltelefon verfügbar. Jede Taste hat mehrer Buchstabenbelegungen, die durch Mehrfachdrücken ein und derselben Taste erreicht werden. Wenn ein Buchstabenpaar auf derselben Taste liegt, muss eine gewisse Zeit gewartet werden. Diese Pause fällt bei LetterWise weg, wodurch es etwas höhere Wortraten erreicht. Beim LetterWise gibt es eine Next-Taste zum Erreichen der Mehrfachbelegung einer Taste. Obwohl Touchscreen-Tastaturen auch meist ein QWERTY-Layout haben, erreichen sie nicht die Wortraten von normalen Tastaturen. Es gibt allerdings sehr viele verschiedene Arten, z.B. auch projezierte Tastaturen, die man schlecht alle mit einem Zahlenwert belegen kann. Der Wert hier von 20-30 Wörtern pro Minute wurde an Touchscreen-Tastaturen mit einer Größe von 7 bis 25 cm, die man mit oder auch ohne Stift bedienen kann, ermittelt. Das DVORAK-Layout ist leicht effizienter als das QWERTY-Layout, wobei es auch Studien gibt, die aussagen, dass es keinen Unterschied in der Effizienz gibt [wike][wikk]. Die höchsten Wortraten lassen sich mit Chord-Tastaturen erreichen, bei denen mehrere Tasten gleichzeitig für einen Eingabewert gedrückt werden müssen. Allerdings sind sie auch nut mit intensivem Training benutzbar.

Wörter pro Minute	Tasttatur
15	Handy mit Multitap [wikj]
20	Handy mit LetterWise [wiki]
20-30	Touchscreen-Tastatur
150	QWERTY [wikk]
150-200	DVORAK [wike]
300	Chord-Tastatur [wikc]

Tabelle 3.1: Wörter pro Minute bei Texteingabe auf verschiedenen Arten von Tastaturen. Die Zahlen stammen aus [SP05]

Zeigegeräte

Mit dem Aufkommen grafischer Oberflächen kamen auch die Zeigegeräte. Objekte können durch Zeigen direkt ausgewählt werden. Der Nutzer muss dabei keine Kommandos lernen, kann sich nicht vertippen und kann den Bildschirm im Blick behalten. Das Zeigen auf Gegenstände ist ein sehr natürlicher Vorgang, einfach zu erlernen, schnell und wenig fehleranfällig.

Man unterscheidet direkte und indirekte Zeigegeräte. Direkt bedeutet, dass man den Bildschirm berührt, fast so, als ob man das dargestellte Objekt wirklich anfassen will. Das hat den Nachteil der Verschmutzung des Bildschirms, ist dafür aber sehr einfach zu erlernen. Beispiele sind der Touchscreen, Lightpen oder der Stylus bei den Palm Kleingeräten. Indirekte Zeigegeräte gibt es in verschiedenen Formen und benötigen etwas mehr Lernaufwand. Man bedient hier ein Gerät mit der Hand und hält den Blick dabei auf dem Bildschirm. Das erfordert Hand-Augen-Koordination, mit der vor allem Kinder und alte Menschen Schwierigkeiten haben. Beispiele für indirekte Geräte sind die Maus, Trackballs, Joysticks, Trackpoints, Touchpads oder Grafik-Tablets. Es gibt neben diesen verbreiteten Geräten noch viele experimentelle Lösungen für Spezialanwendungen wie z.B. Eye-Tracking, Handschuhe, digitales Papier, 3D-Zeigegeräte usw.

Mit Zeigegeräten lassen sich verschiedene Aufgaben lösen. Bereits erwähnt wurde das Selektieren von Objekten. Es lassen sich aber Positionen, Ausrichtung, Pfade und Mengen (z.B. mit Schiebern) eingeben. Dabei hat der Nutzer meist eine direkte visuelle Rückmeldung. Über Menus lassen sich auch Komandos eingeben. Der Ablauf ist dabei normalerweise, Selektieren des Objektes und danach Auswahl des Kommandos aus einem Menu. Professionelle Anwender wählen das Kommando meist über Shortcuts aus, was deutlich schneller geht. Bei Strategie-Computerspielen, wo es wichtig ist, viele Objekte zu selektieren und ihnen Kommandos zu geben, wird der Unterschied zwischen Anfänger, die Kommandos mit der

Maus geben, und Profis, die Shortcuts benutzen, deutlich. Geübte Spieler schaffen es, bis zu 4 mal mehr Aktionen pro Minute auszuführen als ungeübte [wikb].

Beretis bevor es Computer gab, wurden Untersuchungen über die Geschwindigkeit von Handbewegungen gemacht. Paul Fitts veröffentlichte 1954 ein Gesetz, mit dem sich diese Zeit abschätzen lässt [wikf]. Obwohl es damals noch keine Computer gab, lassen sich die Ergebnisse auch für Zeigegeräte an Computer verwenden. Formel (3.1) zeigt eine mathematische Formulierung des Gesetzes. Die Zeit T hängt von der Entfernung D und der Ausdehnung W des Zielobjektes ab. Die beiden Konstanten a und b müssen experimentell gefunden werden und hängen vom Zeigegerät ab. Im Internet gibt es ein Programm, dass in Form eines Spiels diese Werte ermittelt [fit]. Bei über 10000 Messungen wurden dort a = 0.45 und b = 0.13 ermittelt. Aus den Resultaten lässt sich z.B. ablesen, dass ältere Menschen langsamer sind, Dreiecke schlechter zu treffen sind als Kreise und die Farbe des Zielobjektes die Konstante a, also die Zeit zum Finden und reagieren, beeinflusst.

$$T = a + b \log_2 \left(\frac{D}{W} + 1 \right) \tag{3.1}$$

Ausgabegeräte

Das wichtigste Ausgabegerät ist der Bildschirm. Es gibt CRT-Röhrenbildschirme, die langsam von den LCD-Flachbildschirmen abgelöst werden. Für Fernsehen werden PDP-Plasmabildschirme und Beamer beliebter. Bei grossen Werbeanzeigen werden oft LED-Bildschirme verwendet. Weiterhin gibt es auch Bildschirme für Blinde, die ein Relief abbilden und den Tastsinn ansprechen. Eine relativ neue Entwicklung ist das digitale Papier, bei dem Schwarz-Weiß-Bilder mit Auflösungen bis zu 200 dpi darstellbar sind [ein]. Drucker erstellen im Prinzip nur Momentaufnahmen von Bildschirmen und werden mit der Weiterentwicklung von digitalem Papier evtl. an Bedeutung verlieren.

Für diese Arbeit ist es besondern interessant, die Möglichkeiten von 3D-Ausgebgeräten zu kennen. 3D-Drucker existieren inzwischen, wichtiger sind aber 3D-Displays. Es gibt verschiedene Ansätze, 3D-Bilder zu erzeugen. Echte 3D-Bildschirme versuchen ein echtes 3D-Bild darzustellen, während viele ältere Ansätze versuchen zwei Halbbilder, für jedes Auge eins, darzustellen. Dazu muss eine 3D-Brille getragen werden [wika]. Stereogramme, die durch die Buchserie „Das Magische Auge“ bekannt wurden, stellen beide Halbbilder in einem dar. Einen 3D-Effekt erreicht man durch Fokusieren eines weiter hinten liegenden Punktes [wikl]. Dauerhaftes Schielen und 3D-Brillen eignen sich nicht für Geschäftsanwendungen, da sie das Auge mehr ermüden als normale Bildschirme und für stundenlanges Arbeiten nicht verwendbar sind. Echte 3D-Bildschirme sind noch nicht für den Massenmarkt verfügbar, werden aber ganz neue Anwendungserlebnisse ermöglichen.

Sprache und andere Möglichkeiten

Sprache ist die natürlichste Form der Kommunikation. Allerdings ist die Anwendung für die Mensch-Maschine-Kommunikation sehr schwierig. Sprache ist mehr als nur Worte. Betonung und Mimik des Sprechers enthalten auch Informationen die kein Computerprogramm in absehbarer Zeit verstehen kann. Bis man mit einem Computer reden kann, wie man es aus Sient-Fiction-Filmen kennt, wird also noch viel Zeit vergehen. Ein weiterer Nachteil ist, dass Sprechen beim Problemlösen ablenkt, was bei der Hand-Augen-Koordination nicht der Fall ist. Sehr nützlich ist Sprachsteuerung und Sprachausgabe, wenn der Anwender seine Augen und Hände für andere Aufgaben benötigt, z.B. beim Autofahren. Navigationssysteme arbeiten schon lange mit Sprachausgabe, und neue Autos können auch Wortkommandos z.B. um jemanden anzurufen, verstehen.

Sprachausgabe ist in Form von Audioguides in Museen oder Hörbüchern sehr beliebt. Der Nutzer kann nebenbei andere Aufgaben erfüllen. Bei Geschäftsanwendungen ist es meist schneller, einen Text visuell darzustellen als ihn vorzulesen. Sinnvoll können aber kurze Töne, sogenannten Soundicons und Earcons, sein. Um z.B. das Ende eines Vorgangs oder neue Ereignisse anzuzeigen, können Geräusche verwendet werden.

Bei speziellen Anwendungen ist Sprachein- und Ausgabe sinnvoll. Im Allgemeinen sind Tastatur, Maus und Bildschirm aber schneller für Geschäftsanwendungen.

Auf dem Markt der Ein- und Ausgabegeräte gibt es ständig Neuentwicklungen. Eine Firma aus Österreich hat z.B. ein System namens GoMonkey entwickelt, bei dem sich Anwendungen durch Gesten steuern lassen [gom]. Es existiert sogar eine Anbindung an Looking-Glass, ein 3D-Desktopsystem, das in Kapitel 4.4 vorgestellt wird. Die neue Spielekonsole von Nintendo wird mit einem neuartigen Controller ausgeliefert, der als 3D-Zeigegerät funtkioniert und gleichzeitig Feedbeck über Vibration und Sound bietet [nin]. In Spanien wurde ein Musikinstrument entwickelt, bei dem Gegenstände auf einem Tische bewegt werden. Es gibt eine sichtbare Reaktion auf dem Tisch selber und eine Hörbare [rea].

3.2.2 Interaktionsmodelle

Interakionsmodelle lassen sich in drei grosse Gruppen einteilen. Besonders wichtig für diese Arbeit ist Direct-Manipulation. Direct-Manipulation zeichnet sich durch die Sichtbarkeit von Objekten und Akionen aus. Es wird in kleinen sichtbaren Schritten gearbeitet, die einfach rückgängig gemacht werden können. Ein älterer Ansatz ist das Arbeiten mit Menus, Forumularen und Dialogen. Hier ist der Aufbau der Menus besonders entscheident dafür, wie einfach die Anwendung zu benutzen ist. Die älteste Form der Kommunikation mit dem Computer ist die Kommandosprache. Es werden Kommandos in Textform ein-

gegeben, die der Computer direkt verarbeiten kann. Besonders für Experten ist diese Form sehr effizient. Für andere Anwender ist der Lernaufwand bei dieser Form zu gross.

Direct-Manipulation

Der Begriff Direct-Manipulation wurde von 1982 von Ben Schneiderman eingeführt und in nachfolgenden Veröffentlichungen konkretisiert [Sch83]. Es gibt viele verschiedene Beschreibungen von Direct-Manipulation ([SP05] Seite 231). Schneiderman selbst sagt, es geht um direktes Verändern der interessanten Objekte. Das bedeutet, jedes Objekt wird einzeln visuell dargestellt, und ist als solches veränderbar. Die nächsten Absätze stellen die Prinzipien und die Vor- und Nachteile genauer vor.

Es lassen sich drei grundprinzipien für Direct-Manipulation formulieren:

1. visuelle Präsentation der Objekte und Aktionen durch Metaphern und visuelle Sprache
2. physische Aktionen (Tastendrücke) führen direkt zu Veränderung, keine komplizierte Syntax
3. arbeiten in kleinen, schnellen, umkehrbaren Schritten, die direkt sichtbar werden

Aus diesen Prinzipien ergeben sich einige Vorteile. Neue Benutzer lernen den Umgang mit der Oberfläche sehr schnell. Meist reicht es aus, wenn sie einmal sehen, wie es funktioniert. Durch die visuellen Darstellungen fällt es auch leichter, sich später wieder zu erinnern. Das liegt daran, dass sich visuelle Dinge einfacher merken lassen als Wörter. Durch die direkt sichtbaren und umkehrbaren Aktionen, sieht der Nutzer, ob er sich seinem Ziel nähert. Fehlbedienungen und Fehlermeldungen sind selten, da sich alle Aktionen direkt umkehren lassen. Alles im allem, enmpfinden Nutzer Direct-Manipulation-Schnittstellen meist als angenehm, da sie sich vorhersehbar verhalten und dem Nutzer ein Gefühl von Kontrolle geben. Einige Studien deuten darauf hin, dass Direct-Manipulation auch für geübte Nutzer effizienter als Kommandoeingaben sein können [MS87].

Bie der Umseztung von Direct-Manipulation gibt es einige Schwierigkeiten. Zuerst müssen geeignete Metaphern und eine visuelle Sprache gefunden werden. Da hier die Vorkenntnisse des Nutzers eine wichtige Rolle spielen, kann es bei einem großem Nutzerkreis schnell zu Fehlinterpretationen kommen. Besonders Icons, mit ihrer begrenzten Größe, sind oft schwer zu deuten. Es gibt aber auch noch rein technische Probleme. Die visuellen Repräsentationen benötigen viel Bildschirmplatz. Auf älteren Schwarz-Weiß-Bildschirmen oder Kleingeräten reicht der Platz nicht. Da jede Aktion sofort sichtbar sein soll, müssen sie auch entsprechend schnell ausführbar sein. Bei verteilten Anwendungen oder komplexen Datenbankanfragen ist das oft nicht möglich. Auch die Umkehrbarkeit von Aktionen ist teilweise nur sehr schwer umsetzbar. In der Praxis findet man also selten ideale Direct-Manipulation Anwendungen[4].

[4] In Computerspielen findet man meist ideale Direct-Manipulation. Allerdings haben Spiele auch andere Anforderungen und

Vorteile:

- einfach und schnell zu lernen
- Fehlbedienung einfacher zu vermeiden, wenig Fehlermeldungen nötig
- visuelle Repräsentationen einfach zu merken
- wird als angenehm vom Nutzer empfunden, da vorhersehbar, nachvollziehbar und steuerbar

Probleme:

- visuelle Sprache und Metaphern nicht immer eindeutig
- benötigt viel Bildschirmplatz
- benötigt viel Rechenleistung
- Experten können gebremst werden

Menus und Formulare

Menus und Formulare können, wenn sie richtig benutzt werden, sehr gut benutzbar sein. Sie sind einfacher zu erzeugen als ein Direct-Manipulation-Ansatz und trotzdem auch für nicht Experten gut geeignet. Das Ausfüllen von Formularen ist Teil vieler bürokratischer Abläufe und wird von vielen Nutzern deswegen schnell verstanden und erlernt. Auch Menus lassen sich bei guter Organisation ohne Lernaufwand benutzen, und können durch Shortcuts auch für Vielnutzer attraktiv sein. Im folgenden werden kurz einige Menuarten und Selektionsmechanismen vorgestellt.

Ein einfaches Menus ist eine Auswahl von zwei oder mehr Menupunkten. Diese Auswahl kann in verschiedenen Formen dargestellt werden, z.B. einer Dialogbox mit Ja/Nein-Buttons, Radio-Buttons, Checkboxes oder Pulldown-Menus. Mehrere einfache Menus können auf verschiedene Arten verknüpft werden. Lineare Menus stellen unabhängig von der Eingabe des Nutzers mehrere einfache Menus hintereinander dar. Bei baumartige Strukturen können Menupunkten wieder andere Untermenus zugeordnet werden. Wenn man einzelne Menupunkte dabei über mehrere Wege erreichen kann, kommt man zu Netzwerken mit oder ohne Kreisen, je nachdem ob man unendlich lange Wege durch die Verknüpfungen finden kann. Am weitesten verbreitet sind die Baumstrukturen. Das Internet ist ein Beispiel für Netzwerkmenus mit Kreisen.

Dargestellt werden Menus meist durch Radio-Buttons, Checkboxen, Comboboxen, Pulldown-Menus oder Popup-Menus. Bei grossen Mengen an Menupunkten können scrollbare Menus oder Fisheye-Menus, bei denen Menupunkte kleiner werden je weiter sie vom Cursor entfernt sind, verwendet werden. Es gibt

Rahmenbedingungen als Geschäftsanwendungen.

viele weitere Möglichkeiten, Menus darzustellen. Kreisförmige Menus lassen sich schneller und einfacher bedienen bei wenig Menupunkten [wikh]. Ausserdem erlauben sie schnelle Auswahlmöglichkeiten durch Mausgesten. Verschiedene Arten von 3D-Menus, z.B. auf einem drehbaren Würfel oder einer Scheibe sind nicht sehr verbreitet und meist langsamer zu bedienen als die herkömmlichen Ansätze. Wie man Menupunkte am besten anordnet und benennt, wird in Kapitel 7 bei [SP05] oder Schritt 4 bei [Gal02] ausführlich beschrieben.

Für den Aufbau von Formularen gibt es viele Richtlinien. Besonders [Gal02] gibt eine sehr praktische Beschreibung. Wichtig sind ausagekräftige Feldbeschreibungen, logische Anordnung der Felder und Fehlerresistenz. Wenn möglich sollte auf Fehleingaben sofort hingewiesen werden. Auf Seite 297 in [SP05] werden weitere Richtlinien erläutert.

Abschliesend noch einmal die Vor- und Nachteile bei der Verwendung von Menus und Formularen.

Vorteile:

- gut für neue Anwender und Gelegenheitsanwender
- können auch für Experten gut sein, wenn schnelle Auswahlmöglichkeiten vorhanden
- kein auswendig Lernen erforderlich
- vereinfacht Dateneingabe
- wenig Fehleranfällig

Nachteile:

- unübersichtlich bei zuvielen Menupunkten oder schlechter Organisation
- kann Experten bremsen bei der Nutzung
- benötigt viel Platz auf dem Bildschirm bei größeren Menus oder Formularen
- meist nicht sehr flexibel, an vorgegeben Prozesse gebunden

Kommandos und Sprache

Das Steuern des Computers über Kommandos oder eine bestimmte Sprache gibt einem Experten die volle Kontrolle über den Computer. Theoretisch sollte er jedes Problem in einer Sprache beschreiben und lösen können. Praktisch muss aber meist das Problem erst in die ensprechende Programmiersprache übersetzt werden, was sehr viel Aufwand bedeuten kann. Neue Tools und Methoden versuchen genaus diese Hürde zu beseitigen [Dmi]. Im folgenden wird kurz auf die bestehenden Formen von Kommandoeingaben und Sprachen eingegangen.

Es gibt verschiedene Möglichkeiten, Kommandos zu organisieren. Die einfachste ist, eine Menge von Kommandos bereit zu stellen, bei der jedem Kommando genau eine Aufgabe zugeordnet ist. Solange die Menge der Aufgaben klein ist, funktioniert das gut. Die zweite Möglichkeit sind Kommandos mit Argumenten und Optionen. Diese Form wird bei den meisten Konsoloen für Betriebssysteme verwendet. Ein dritte Möglichkeit ist eine hierarchische Struktur. Aus wenigen Einzelkommandos können so sehr viele Kombinationen gebildet werden. Ein hierarchischer Aufbau ist einfacher zu lernen, da weniger Kommandos auswendig gelernt werden müssen.

Die Namenswahl und Struktur einer Kommandosprache ist entscheidend für die Benutzbarkeit und den Lernaufwand. Konsistente Namensgebung und Argumentanordnung sind genauso wichtig wie die Identifikation der zu erledigenden Aufgaben und Auswahl entsprechender Kommandos. Einen detaillierten Überblick bieten Kapitel 8.4 und 8.5 in [SP05].

Wie bereits in Kapitel 3.2.1 angesprochen, ist die natürliche Sprache, wie wir sie zum Kommunizieren mit anderen Menschen benutzen, in absehbarer Zukunft nicht geeignet, um mit einem Computer zu „sprechen“. Bei den meisten praktikablen Ansätzen muss der Mensch sich an bestimmte Regeln halten, so dass ein Kompromiss zwischen natürlicher Sprache und Kommandosprache entsteht. Die nächste grosse Weiterentwicklung wird wahrscheinlich die sprachorientierte Programmierung sein, wie sie in [Dmi] beschrieben wird. Probleme werden dann in ihrer eigenen Sprache beschrieben, so dass z.B. ein Mathematiker seine Formeln direkt einem Computer übergeben kann. Die folgenden Auflistungen zeigen die Vor- und Nachteile von Kommandosprachen noch einmal in Kurzform.

Vorteile:

- sehr flexibel, grosse Menge an Problemen lössbar
- Automatisierung durch Scripte und Macros
- wenig Rechenressourcen zur Verarbeitung da nur Textein- und Ausgabe

Nachteile:

- aufwendig zu lernen, nur für Experten geeignet
- schnell wieder verlernbar
- hohe Fehlerrate durch Vertippen
- für viele (z.B. ältere Menschen) kaum erlernbar

3.2.3 Metaphern

Der Begriff Metapher kommt aus dem Griechischen und bezeichnet Worte, die in einer übertragenen Bedeutung gebraucht werden [wiko]. Zwischen der übertragenen Bedeutung und dem verwendeten Wort

besteht normalerweise eine Analogie, also eine Ähnlichkeit. Die Sprache ist voll von Metaphern und gewinnt durch sie an Ausdrucksmöglichkeit. Viele Metaphern werden so selbstverständlich benutzt, dass sie nicht mehr als solche auffallen. Der Fuß des Berges zum Beispiel, ist nicht wirklich ein Fuß.

Metaphern, und besonders ihre dargestellten Analogien, sind gut geeignet um Neues zu lernen, oder komplizierte Sachverhalte zu erklären. Fachleute benutzen meist Analogien, um Vorgänge zu erklären. Zum Beispiel wird von einem Computer oft so gesprochen, als ob er eine Person ist. Die Aussage „Der Computer schaut nach, ob ein CD-Laufwerk da ist“, ist ein Beispiel für so eine Analogie. Durch Metaphern können also Zusammenhänge verstanden und Neues gelernt werden.

Was bei der Sprache funktioniert, funktioniert auch bei Mensch-Maschine-Schnittstellen. Ein Papierkorb dient z.B. dazu Dinge wegzuwerfen, deshalb gibt es auf dem Desktop ein Papierkorb-Icon um Objekte zu löschen. Jeder, der einen Papierkorb kennt, versteht diese Analogie[5]. Wenn man also Analogien zu Dingen, die dem Nutzer bekannt, sind richtig verwendet, erspart man ihm Lernen, Nachdenken und Anstrengung. Genauso kann auch das Gegenteil eintreten. Wenn die Anwendung sich anders verhält, als der Nutzer es durch die Analogie erwartet, kann es zu Missverständnissen kommen.

Es wurde bereits erwähnt, dass Metaphern immer eine Ähnlichkeit benötigen. Diese Ähnlichkeit kann visuell, funktionell oder strukturell sein. Man kann für die drei Typen auch kurze Wortgruppen verwenden, „sieht aus wie“, „macht das gleiche wie“ und „ist aufgebaut wie“. Tabelle 3.2 zeigt einige Beispiele für die 3 Typen. Meist verwendet man mehrere der Typen zugleich. Wenn etwas ähnlich aussieht, dann sollte es auch ähnlich funktionieren.

Im folgenden werden 2 sehr oft vorkommende Metaphern kurz erläutert. Zum einen die Desktopmetapher und zum anderen die Werkzeugmetapher.

Die Desktopmetapher

Die Desktopmetapher wird zusammen mit dem sogenannten „Windows, Icons, Menus, Pointing interfaces“, kurz WIMP, benutzt. Das Konzept wurde berreits 1968 auf einer Konferenz vorgestellt [DW68] und weiterentwickelt, bis es 1985 zuerst von Apple Macintosh kommerziell eingesetzt wurde.

Die Desktopmetapher benutzt folgende funktionelle Analogien. Der Bildschirm ist der Schreibtisch. Auf ihm liegen Dokumente in Form von Icons und Foldern. Geöffnete Dokumente sind wie Papierzettel, die übereinander liegen können, dargestellt durch Fenster. Dass man Fenster vergrössern und verkleiner kann, ist eine Erweiterung gegenüber der Realität. Die Hand, mit der man sonst auf einem Schreibtisch

[5] Hier zeigt sich, dass es wichtig ist, zu wissen aus welchem Kulturkreis und mit welcher Vorbildung die Anwender kommen. Nicht jeder Mensch weiß was ein Papierkorb ist, Metaphern funktionieren also nicht immer.

Analogie	**Beispielmetapher**	**Bedeutet z.B.**
Visuell sieht aus wie...	ein CD-Player	es gibt ein Display, Schaltknöpfe und einen CD-Schacht
	Analoguhr	rundes Zifferblatt, bewegende Zeiger
Funktionell macht das gleiche wie...	ein Papierkorb	Objekte können hineingelegt werden
	CD-Player	es gibt Play, Stop, Eject, Skiptasten
	Versandkatalog	es gibt Produktbilder, Preislisten, Bestellmöglichkeiten
Strukturell ist aufgebaut wie...	eine Stadt	es gibt Häuser, Strassen, Wegweiser,...
	ein Baum	es gibt Verzweigungen und Blätter

Tabelle 3.2: Metaphern können in drei Typen unterteilt werden, abhängig davon, welche Art von Analogie sie benutzen.

agiert, wird durch den Mauscursor dargestellt. Das Nehmen und Ablegen von Objekten ist nichts anderes als Drag and Drop. Der berreits erwähnte Papierkorb zum Löschen von Dokumenten wird durch ein Icon dargestellt. desktopmetapher:

An einigen Stellen unterscheidet sich das WIMP-Interface von der Realität. Normalerweise arbeitet man mit 2 Händen. Beim Computer hat man aber nur eine Maus als Handersatz. Oft benutzt man aber beim Arbeiten an einem Schreibtisch die 2. Hand nur um etwas festzuhalten, was beim Computer nicht nötig ist. Eine zweite Maus ist also nicht nötig. Ein weiterer Unterschied sind die Menus. Normalerweise arbeitet man direkt mit Dokumenten und muss nicht erst in einem Menu nach dem richtigen Kommando suchen. Der in Kapitel 3.2.2 beschrieben Direct-Manipulation-Ansatz entspricht eher dem realen Arbeiten.

Die Werkzeugmetapher

Wenn man in der wirklichen Welt arbeitet, benutzt man oft Wekrzeuge. Bei Dokumenten kann das z.B. Stift, eine Schere, Leim oder ein Lineal sein. Die Werkzeugmetapher benutzt eine visuelle und funktionelle Analogie. Es werden Icons dargestellt, die wie das entsprechende Werkzeug aussehen und auch ähnlich funktionieren. Das Werkzeug kann selektiert und benutzt werden. Das entspricht dann z.B. dem Nehmen eines Stiftes, wenn man das Stift-Icon anklickt. Teilweise wird der Mauscursor dann zum gewählten Symbol, so als ob man das Werkzeug in der Hand hält. Die Werkzeugmetapher entspricht dem realen Arbeiten mit Objekten.

3.3 Zusammenfassung

Um intuitive grafische Schnittstellen zu erstellen, ist es wichtig, möglichst viel über das Verhalten des Anwenders zu wissen. Es wurden Gesetzmässigkeiten genannt, wie Menschen wahrnehmen und was ihre Aufmerksamkeit lenkt (Kapitel 3.1.1). Besonders wichtig ist es, eine Aufmerksamkeitssprache zu entwickeln, die der Nutzer schnell versteht und lernen kann. Dabei sollten möglichst wenige und einfache gestalterische Mittel verwendet werden. Nur so lässt sich Verwirrung vermeiden und einigermaßen vorhersagen, wie der Benutzer den Programmzustand wahrnimmt.

Anwender verfolgen meist klare Ziele, die sie mit Hilfe von Operatoren, z.B. Buttons, Menus, usw., erreichen. Es ist wichtig, die Operatoren klar zu kennzeichnen, und ihre Menge zu begrenzen. Das bedeutet z.B., dass es keine undeutlichen Icons ohne Tooltip gibt und anklickbare Objekte als solche gekennzeichnet sind. Um die Menge der Operatoren zu begrenzen, bieten sich verschiedene Anwendungs-Modi für Anfänger und Fortgeschrittene an, oder Wizards die dem Anwender eine Reihe von Fragen stellen.

Die grafische Oberfläche von Naked-Objects, um die es in dieser Arbeit geht, verfolgt einen anderen Ansatz. Es wird eine kleine Menge von allgemeinen Operatoren benutzt, mit denen das Anwendungsprogramm komplett manipuliert werden kann. Der Anwender lässt sich dadurch schlecht steuern und einschränken, wird aber motiviert, seine Ziele durch systematisches Problemlösen zu erreichen, z.B. durch Zerlegen in Teilziele. Da die Menge der Operatoren überschaubar bleibt, ist die Wahrscheinlichkeit gering, dass der Anwender nicht mehr weiß, wie er seine Ziele erreichen soll und es mit Trial-and-Error versucht.

Im Kapitel 3.2 wurden einige technische Grundlagen erläutert, die für das Entwickeln von grafischen Oberflächen interessant sind. Es gibt viele verschiedene Ein- und Ausgabegeräte. Am verbreitesten sind die Tastatur und die Maus. Sinnvoll ist es, Tastatureingaben und Mauseingaben zu kombinieren. Mit der Maus wählt man ein Objekt aus und mit der Tastatur eine Aktion. Für Anfänger sollte auch eine reine Maussteuerung möglich sein. Einige Daten zu Maus und Tastatur, sowie alternative Ein- und Ausgabegeräte werden in Kapitel 3.2.1 vorgestellt.

Um die Kommunikation zwischen Mensch und Maschine zu organisieren, gibt es verschiedene Modelle (3.2.2). Das älteste ist die Kommandosprache. Der Nutzer gibt Kommandos mit einer bestimmten Syntax ein, die vom Rechner verarbeitet werden. Um auch Nicht-Technikern die Benutzung von Computern zu ermöglichen, entstanden Anwendungen mit Menusteuerung und Formularen. Die modernste, und auch aufwändigste Form der Interaktion ist Direct-Mainpualtion (direkte Manipulation). Objekte werden hier visuell dargestellt und können direkt verändert werden. Damit der Nutzer eine Anwendung schneller versteht, können Metaphern verwendet werden. Metaphern benutzen Analogien (Ähnlichkeiten) zu rea-

len Dingen. Wenn der Nutzer diese realen Dinge kennt und den Bezug versteht, sind Metaphern sehr nützlich. Kapitel 3.2.3 stellt verschiedene Arten von Metaphern vor.

4 Desktopsysteme mit integrierter 3D-Unterstützung

Seit Jahren ist auf jedem Heimrechner leistungsfähige 3D-Grafik möglich. Meist wird die Grafikhardware aber nur von Spielen und Spezialsoftware (z.B. 3D-Modeller oder Physiksimulationen) genutzt. Die verbreiteteten älteren Window-Systeme (Windows, MacOS, X-Windows) haben bisher nur 2D-Beschleunigerfunktionen genutzt. Mac-OSX, Windows-Vista und andere kleinere Systeme bilden eine neue Generation von Arbeitsoberflächen. Es wird intensiv 3D-Grafikhardware genutzt. Der Desktop entwickelt sich vom einfachen 2D-Bild aus Pixeln, zu einer 3D-Szene. Das macht viele optische Effekte möglich, wie Beleuchtung, Rotation, Transparenz und stufenloses Zoomen, die das Arbeitsgefühl (User Experience) deutlich erweitern werden. Dabei wird nicht nur die optische Qualität verbessert, sondern auch technische Aspekte wie Stabilität und Sicherheit.

Mit der Integration von 3D-Funktionaliät in die Desktopsysteme ergeben sich neue Möglichkeiten für Anwendungen. Auch in Geschäftsanwendungen werden 3D-Komponenten Einzug halten. Wie diese neuen Möglichkeiten sinnvoll genutzt werden können (am Beispiel von NakedObjects), soll in dieser Arbeit untersucht werden. In diesem Kapitel werden aber zunächst die Grundlagen der 3D-Desktopsystem vorgestellt.

Zuerst werden die größten Schwachstellen der älteren Systeme genauer beschrieben. Danach werden 3 aktuelle Systeme vorgestellt. Zuerst geht es um die beiden Grafiksysteme von Windows und Mac-OSX. Das Dritte ist ein Desktopsystem, das zu keinem bestimmten Betriebssystem gehört, Looking-Glass, eine Entwicklung von SUN. Da es sich sehr gut für experimentelle 3D-Anwendungen eignet, wird es etwas genauer vorgestellt. Der Praktische Teil der Arbeit wird dieses System benutzen. Schwerpunkt bei den Betrachtungen sind die Schnittstellen für Anwendungsprogrammierer und das Look and Feel, also der optische Gesamteindruck. Die Umsetzung der neuen Anforderungen an Grafiktreiber und Hardware werden nicht genauer beschrieben. Im Anhang B befindet sich ein Abschnitt zur Umsetzung bei Windows-Vista. Die Zusammenfassung am Ende des Kapitels gibt einen Tabellarichen Überblick zu den vorgestellten Systemen und einen Ausblick auf die Entwicklung in diesem Bereich. Für das X-Windows System gibt es Projekte zur Verbesserung des Grafiksystems, auf die hier nicht eingegangen wird. Eine Einführung dazu findet man bei [Wei05].

4.1 Schwächen alter Systeme

Windows (vor Vista), MacOS und X-Windows funktionieren alle ähnlich. Jedes Programm bekommt einen Bereich des Bildpuffers (Fenster) für den es verantwortlich ist. Wenn ein verdeckter Teil sichtbar wird, ist das Programm, dem dieser Teil gehört, verantwortlich ihn neu zu zeichnen. Das hat den Nachteil, dass jedes Programm immer dazu in der Lage sein muss, sein Fenster neu zu zeichnen, was nicht immer funktioniert. Abbildung 4.1 zeigt ein Beispiel. Das vordere Fenster wird über dem hinteren bewegt, welches gerade sein Dokument speichert. Obwohl die Office-Anwendung nicht abgestürzt ist, entstehen kurzzeitig Grafikfehler.

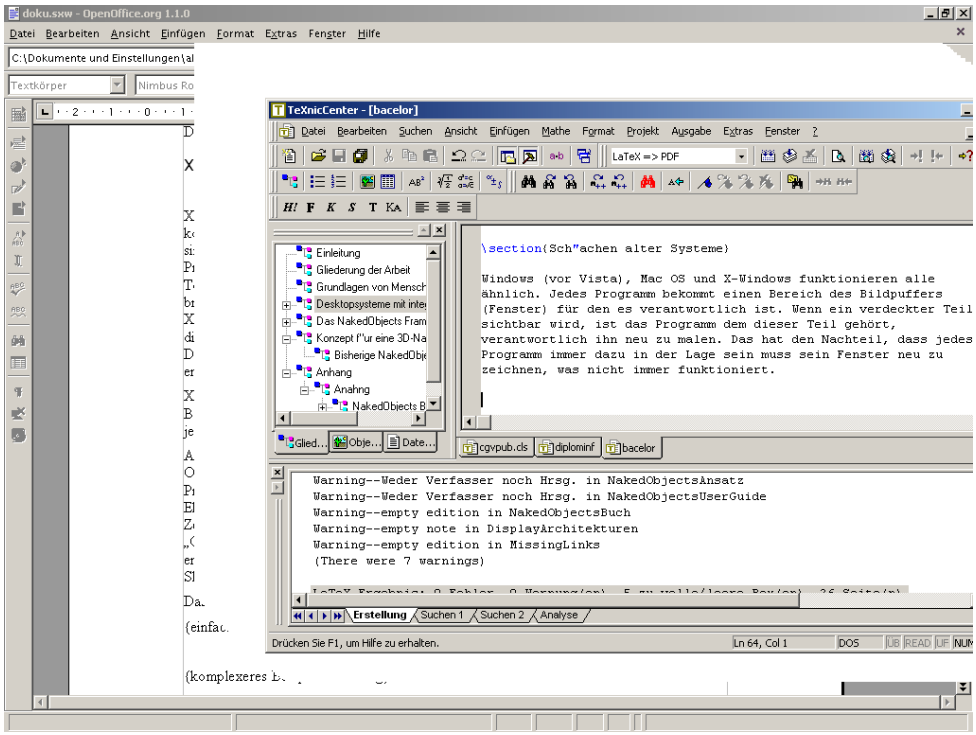

Abbildung 4.1: Typischer Repaint-Fehler. Das hintere Fenster kann die freiwerdenden Stellen nicht schnell genug nachzeichnen.

Bisher basiert die Darstellung immer auf Pixeln, wobei meist die Pixeldichte nicht beachtet wird. Da sich die Grafiksysteme weiterentwickeln und dabei immer auf die gängigen Monitore abgestimmt sind, fällt das dem normalen Nutzer nicht negativ auf. Wenn man aber Windows auf einem hochauflösenden Bildschirm startet, erkennt man zunächst nicht viel, weil die Icons und Buchstaben winzig klein sind (Abbildung 4.1). Ein ähnliches Problem ergibt sich bei einem alten Monitor mit sehr geringer Auflösung.

Neuere Systeme haben die beiden eben angesprochenen Probleme (Pixeldichte, Neuzeichnen) nicht mehr. Das Problem mit den Pixeln wird durch vektorbasierte Grafik gelöst. Dazu werden Schnittstellen wie Direct3D, OpenGL oder ähnliche benutzt. Erst am Ende des Zeichenprozesses werden die durch

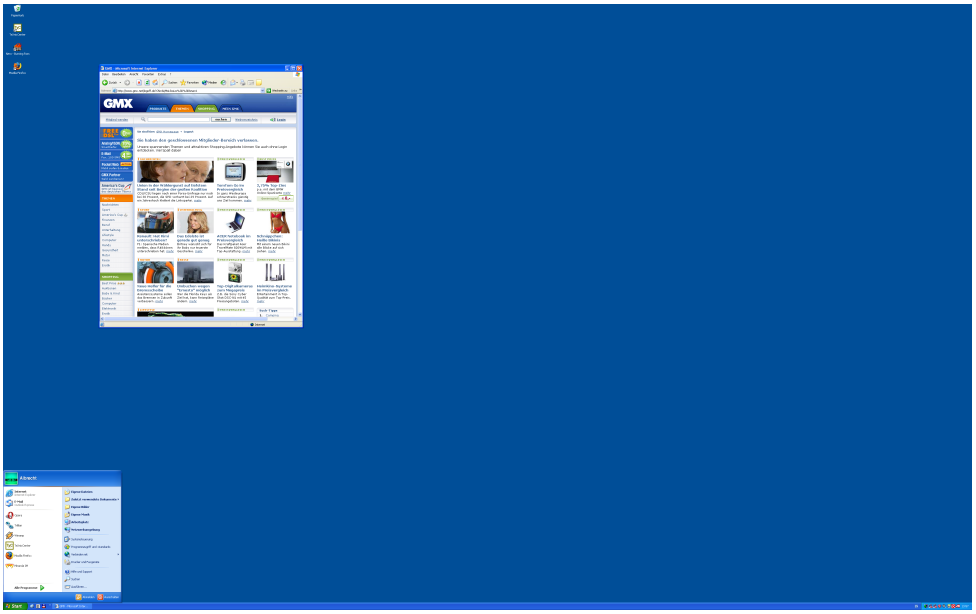

Abbildung 4.2: Windows auf einem 204dpi Monitor (3840x2400; 22 zoll), 4.5 mal soviel wie die heute üblichen 96dpi.

Vektoren beschriebenen Objekte auf ein Ausgabemedium gerastert. Auch das Neuzeichnen eines Fensters ist nicht mehr Aufgabe der Programme, die zu einem Fenster gehören. Sie zeichnen in eine Art virtuellen Grafikspeicher, der vom Desktop-System zu den einzelnen Fenstern zusammengefühgt wird. Dadurch kann auch keine Anwendung mehr in den Bereich einer fremden Anwendung zeichnen.

4.2 Windows-Presentation-Foundation (Windows-Vista)

Seit etwa 20 Jahren verwendet Windows die gleiche Grafikschnittstelle (GDI). Sie wurde zwar ständig weiterentwickelt, aber nie grundlegend überarbeitet und weist die am Anfang des Kapitels erwähnten Schwachstellen auf (Pixelbasiert, Repaintansatz). Seit etwa 10 Jahren arbeitet Microsoft an einer neuen Grafikschnittstelle, bekannt unter dem Codenamen Avalon. Der Name Avalon wird inzwischen von Microsoft nicht mehr verwendet. Es wird jetzt Windows-Presentation-Foundation, kurz WPF, genannt. Geplant ist die Fertigstellung für 2007. Dann soll es auch für Windows-XP und Windows-Server-2003 verfügbar sein.

WPF benutzt nicht nur intern neue Technologie, es ist auch für alle Entwickler ein grosser Fortschritt. Es stellt eine einheitliche und konsistente Grafikschnittstelle für User-Interfaces, 2D-Grafik, 3D-Grafik, Vektorgrafik, Video und Audio, usw. dar. Selbst Webanwendungen können die gleichen Schnittstellen benutzen. Diese Einheitlichkeit macht jedem Entwickler sehr viel Funktionalität zugänglich, ohne dass er sich Spezialwissen aneignen muss. Ein Entwickler, der 3D-Grafik verwenden will, muss nichts mehr

über 3D-Hardware oder Vektorrechnung wissen. Microsoft erhofft sich durch diese Vereinfachung sehr viel ansprechendere grafische Oberflächen für alle zukünftigen Windowsprogramme.

WPF und Direct3D bieten beide die Möglichkeit 3D-Grafik darzustellen. Sie haben aber unterschiedliche Anwendungsbereiche und Zielgruppen. Die 3D-API von WPF (Avalon 3D) ist vollständig in WPF integriert. Sie kann problemlos mit 2D-Grafik und anderen Medien (Film, Foto,..) zusammen benutzt werden. Ausserdem ist es einfach zu benutzen, ohne spezielles Wissen über 3D-Grafik zu erfordern. Direct3D hingegen bietet viel mehr Kontrolle über die 3D-Hardware. Dafür erfordert es detailliertes Spezialwissen und kann nicht ohne weiteres mit anderen darstellenden APIs zusammen benutzt werden. Die Wahl zwischen Avalon 3D und Direct3D hängt also von der Komplexität der darzustellenden 3D-Szene und dem Bedarf nach anderen WPF Features ab.

Quellen für die folgenden Abschnitte sind Folien von der WinHEC [Mic], Videos von Interviews mit Microsoftmitarbeitern [Cha] und Internetseiten wie [Thu]. Da Windows-Vista zum Entstehungszeitpunkt dieser Arbeit noch nicht fertig ist, gibt es leider keine besseren Quellen. Es besteht auch die Möglichkeit, dass sich an einigen Details noch etwas ändert. Wer die Arbeit stark zeitversetzt liest, sollte sich auch in aktuellen Quellen informieren.

4.2.1 Architektur

Wie in Abbildung 4.3 zu sehen, baut WPF auf Direct3D auf. Der Desktop wird vom Desktop-Window-Manager (DWM im folgenden) als 3D-Szene dargestellt. Die Abbildung stellt schematisch 3 verschiedene Anwendungsfälle dar. In der Mitte eine neue Windows-Vista-Anwendung. Sie benutzt die WPF-Bibliotheken. Die Anwendung wird mit Hilfe der Grafikhardware in eine Texture gerendert und dann vom DWM auf einem Polygon dargestellt. Rechts ist eine ältere Windows-Anwendung dargestellt. Sie benutzt eine Softwareimplementierung der alten GDI-Schnittstelle. Hier wird keine Grafikhardware benutzt. Links ist eine DirectX-Anwendung zu sehen. Dort wird DirectX direkt ohne Zwischenschicht benutzt. Das sind meist Computerspiele oder andere Programme mit komplexen 3D-Szenen und Effekten.

Bei jeder WPF-Anwendung läuft eine sogenannte Unified-Composition-Engine (UCE) in einem eigenem Thread. Aufgabe der UCE ist es, einen Baum aus sogenannten Visuals auszuführen und in ein Bitmap oder den Grafikspeicher direkt zu schreiben. Dieser Baum wird als Composition-Tree bezeichnet. Ein Visual enthält Informationen über Transparenz, Kinder-Visuals, Transformation, Clipping und Zeichenoperationen. Das „Unified“ im Namen der UCE hat durchaus seine Berechtigung. Es können 2D und 3D Grafik, Bilder, Video, Audio und Text verarbeitet werden. Unter Windows-XP gab es verschiedenes APIs für diese Dinge. Jetzt muss der Entwickler sich nur noch mit WPF beschäftigen.

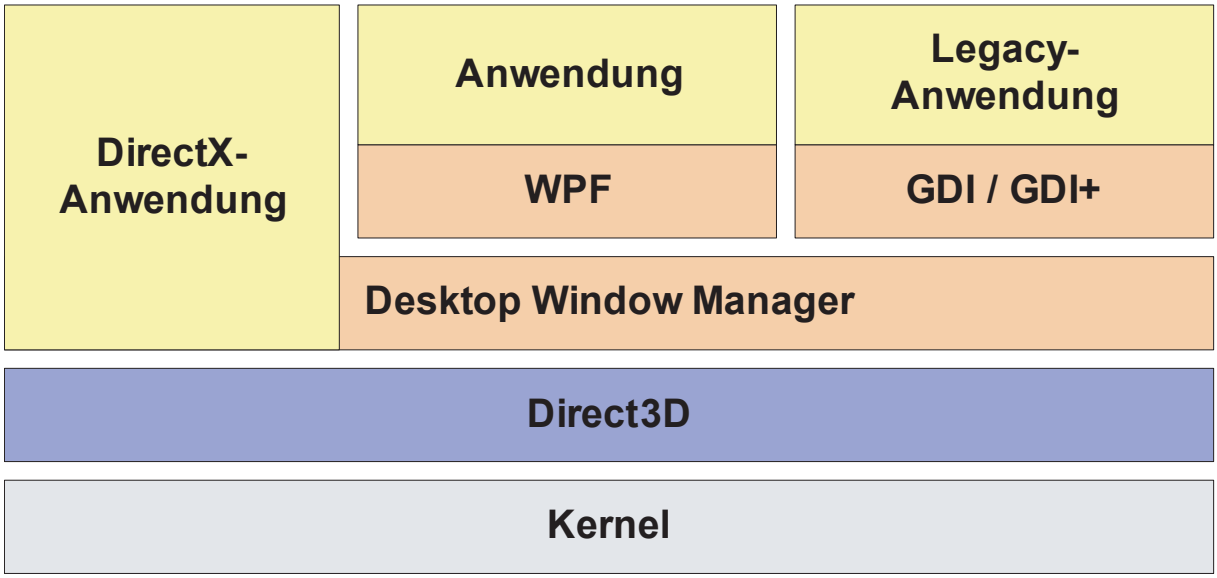

Abbildung 4.3: Windows-Presentation-Foundation Architektur

Die UCE arbeitet mit Native-Code und bekommt von der Anwendung, die mit Managed-Code läuft, einen UI-Tree zum Zeichnen. Die Kommunikation ist asynchron, wenn die Anwendung also hängt, ist die UCE und damit die grafische Darstellung nicht betroffen. Eine abgestürzte oder blockierte Anwendung wird dadurch keine grafischen Fehler mehr erzeugen. Die Trennung zwischen UI und Composition macht Windows-Vista deutlich stabiler und setzt an der Hauptursache für Systeminstabilität an [Mic]. Ausserdem kann die Anwendung mit Managed-Code nicht so einfach wie bei Windows-XP das gesamte System blockieren. Die Trennung wird auch für die Remote-Desktop-Technologie verwendet.

Das Resultat der UCE's der einzelnen Anwendungen wird in Form eines Bitmaps an den Desktop Windows Manager (DWM) gegeben. Die Aufgabe des DWM ist es, den Desktop auf den Bildschirm zu zeichnen. Dazu hat er einen Composition-Tree, in dem die Bitmaps der laufenden Anwendungen Knoten sind. Die UCE des DWM wird auch als Desktop-Composition-Engine (DCE) bezeichnet. Die DCE ist der letzte Schritt der Verarbeitung und zeichnet direkt in den Grafikspeicher.

Damit alle Grafikeffekte funktionieren, benötigt Windows-Vista moderne Grafikhardware. Es werden Pixel Shader benutzt, also kleine Programme, die auf dem Grafikprozessor ausgeführt werden. Um Kompatibel zu älterer Hardware zu bleiben, kann auch komplett auf Hardwarebeschleunigung verzichtet werden. Dann werden aber viele grafische Effekte des DWM abgeschaltet. Windows-Vista wird also auf einem neuem Rechner sehr viel besser aussehen. Da jede Instanz der UCE die GPU benutzt, ergeben sich neue Anforderungen bei der Verteilung von Grafikresourcen. Im Anhang B ist genauer beschrieben, wie eine Zeichenoperation abläuft.

4.2.2 API

Microsoft hat sich für eine XML-Schnittstelle entschieden. Dadurch soll eine Vielzahl von Designtools und eine klare Trennung zwischen Programmlogik und Design möglich werden. Die entsprechende XML Sprache heißt XAML. XAML steht für „eXtensible Application Markup Language“. Mit dieser Markup-Sprache können Grafiken, Fenster und vieles mehr beschrieben werden. Während der Programmierer mit VisualStudio Programmcode bearbeitet, soll ein Designer mit einem Tool seiner Wahl per Maus grafische Oberflächen bearbeiten. Laut Microsoft soll es eine breite Palette solcher Designtools geben, die alle XAML-Dateien produzieren. In diesen XAML-Dateien können Grafikelemente an Daten gebunden werden. Diese saubere Trennung macht die Zusammenarbeit zwischen Programmierern und Designer einfacher. Ein Designer kann das Aussehen einer Anwendung ändern, ohne dass er einen Programmierer braucht, der die Änderungen in das Programm aufnimmt.

XAML kann nicht nur 2D-Grafik beschreiben, sondern auch komplette Benutzeroberflächen. Sogar kleine 3D-Szenen sind möglich. Genaugenommen kann, ähnlich dem JavaBeans-Konzept, jedes beliebige Objekt mit seinen Properties beschrieben werden.

Aus jeder XAML-Datei wird automatisch Quellcode generiert, der einen Baum aus Objekten generiert (UI-Tree). Auf diese Objekte kann wärend der Laufzeit des Programmes zugegriffen werden. Dieser Baum ist eine direkte Abbildung der XAML Elemente. Er enthält noch recht abstrakte Elemente (z.B. Controls) und ist zum direkten Zeichnen nicht geeignet. Es gibt deshalb noch eine Verarbeitungsstufe, den sogenannten Composition-Tree. Die Elemente in diesem Baum werden „Visuals“ genannt und enthalten alle Informationen, die zum Zeichnen wichtig sind (z.B. Transparenz, Skalierung,...).

Die Abbildungen 4.4 und 4.5 zeigen ein einfaches XAML-Beispiel. Abbildung 4.6 zeigt mehrere Ansichten eines animierten Buttons. Eine CD mit Hülle, die sich bei einem Mausklick öffnet. Solche 3D-Buttons können von einem Designer sehr einfach erstellt werden, indem sie Keyframes vorgeben. Den XAML-Code zu diesem Beispiel und viele andere interresante Designbeispiele findet man bei [Dun].

4.2.3 Look and Feel (Aero)

Moderne Grafikhardware wird von Windows-Vista auf 2 Arten benutzt. Für besser aussehende Anwendungen und für ein verbessertes Nutzererlebnis (User experience). Die Grafikschnittstelle für Anwendung wurde im vorhergehende Kapitel behandelt, in diesem Kapitel geht es um das Nutzererlebnis.

Der Begriff Nutzererlebnis ist eine freie Übersetzung von dem was Microsoft „User experience“ nennt. Damit ist alles optische ausserhalb der Anwendungen gemeint. Also z.B. Aussehen der Fenster, des Startmenus, Animationen beim Entstehen von Fenstern, Transparenz, Lichteffekte und vieles mehr. Während

```
<Border xmlns="http://schemas.microsoft.com/winfx/avalon/2005">
   <DockPanel Background="silver" Margin="20" LayoutTransform="scale2">
      <Menu DockPanel.Dock="Top">
         <MenuItem Header="File" />
         <MenuItem Header="Edit" />
      </Menu>
      <TreeView>
         <TreeViewItem Header="root1">
            <TreeViewItem Header="child" />
         </TreeViewItem>
         <TreeViewItem Header="root2">
      </TreeView>
      <ListBox>
         <ListBoxItem>listBoxItem1</ListBoxItem>
         <ListBoxItem>listBoxItem2</ListBoxItem>
      </ListBox>
   </DockPanel>
</Border>
```

Abbildung 4.4: XAML Beispiel

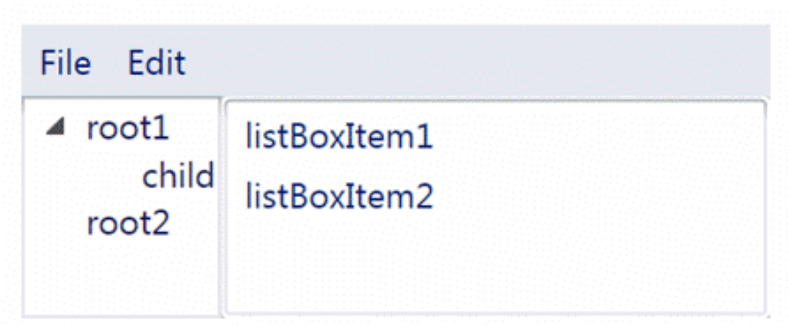

Abbildung 4.5: Screenshot vom XAML-Beispiel aus Abbildung 4.4

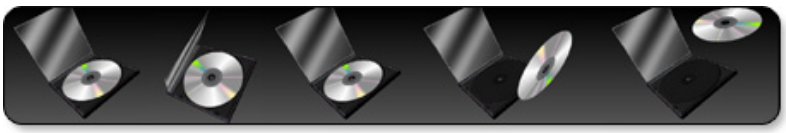

Abbildung 4.6: Mehrere Ansichten eines animierten Buttons.

sich optische Qualität und Geschwindigkeit für Anwendungen mit besserer Grafikhardware gleichmässig verbessert, wird es nur 2 verschiedene Stufen beim Nutzererlebnis geben. Die beiden Stufen werden Aero und Aero-Glass heißen, wobei Aero-Glass die höheren Hardwareanforderungen stellt. Am Windows-Logo soll direkt sichtbar sein, welche der beiden Stufen aktiv ist.

Aero wird sich optisch deutlich von Windows-XP unterscheiden. Es wird auf WPF (Avalon) aufbauen und erfordert mindestens DirectX 9, 32MB VRAM und AGP 4x Bus. Aero-Glass stellt deutlich höhere Anforderungen wie z.B. Pixel Shader 2.0 Unterstützung und Longhorn-Display-Driver-Model (siehe Anhang B). Windows wird jeden Rechner automatisch testen, ob er genug Leistung hat, um auch bei hoher Auslastung ein flüssiges Nutzererlebnis aufrechtzuerhalten. Die Desktop-Composition-Engine soll möglichst immer mit der Bildschirmwiederholrate synchron laufen, bzw. mit der halben bei sehr hohen Bildschirmfrequenzen.

Insgesammt scheint die Oberfläche auf den ersten Blick sehr bunt und etwas verspielt. Das liegt sicher auch am naheliegenden Vergleich mit dem sehr elegant aussehendem Mac-OSX. Die Abbildungen 4.7, 4.8 und 4.9 zeigen einige Besonderheiten der neuen Oberfläche.

Abbildung 4.7: Windows-Vista Desktop mit einem durchsichtigen (Glassy) Fenster. Alles hinter dem Fenster wird verschwommen dargestellt. Am rechten Rand können sogenannte Gadgets angezeigt werden.

Abbildung 4.8: Links: leuchtender Close-Button. Mitte: live Vorschaufenster. Rechts: Alt-Tab

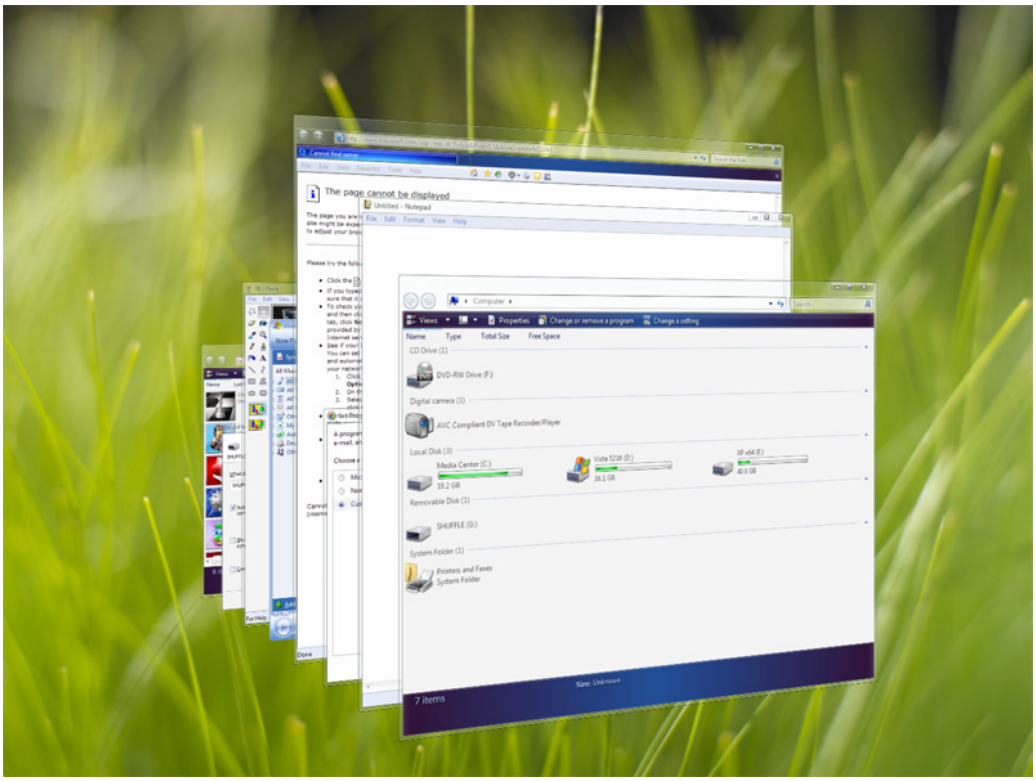

Abbildung 4.9: Eine neue Variante von Alt-Tab ist Flip3D. Die Fenster werden dreidimensional hintereinander angeordnet. Per Mausrad oder Cursortasten kann ein Fenster ausgewählt werden.

Unter dem Namen Classic wird auch das bekannte Windows 2000 Aussehen verfügbar sein. Auf Computern ohne ausreichende Grafikhardware wird Windows-Vista also auch laufen. Microsoft will dadurch die Nutzer, die kein neues Aussehen wollen oder sich davon abgelenkt fühlen, auch bedienen. Die neuen Windows-Vista Schnittstellen, bis auf die UI bezogenen, werden auch unter Classic verfügbar sein.

Bei Aero ist der Desktop genau wie jede Anwendung eine 3D-Szene, die vom Desktop-Window-Manager (DWM) verwalted wird. Allerdings unterscheiden sich die Anforderungen etwas von denen einer 3D-Anwendung. Auf dem Desktop gibt es nur wenige, dafür grosse Polygone. Anwendungen sind Texturen auf einigen Polygonen (Fenstern), die sich schnell und oft ändern. Zukünftige Grafikhardware und Treiber werden auf diese schnellen Änderungen hin optimiert werden müssen.

Insgesammt ist es das Ziel von Microsoft, dass Desktop Composition so oft wie möglich benutzt wird. Auflösungsunabhängikeit, transparente Fenster, leuchtende Buttons, live Vorschau, stufenloses Zoomen, grafische Stabilität und viele andere Features funktionieren ohne Desktop Compostion nicht.

4.3 Quartz (Mac-OSX)

Was Microsoft erst mit Windows-Vista einführt, ist bei Mac schon lange Realität. Mac-OSX verfügt über eine vektorbasierte Grafikschnittstelle mit dem Namen Quartz. Sie besteht aus den 2 Komponenten Quartz-2D und Quartz-Compositor. Quartz-2D stellt die Funktionialität zum Zeichnen bereit, wobei immer in einen Hintergrundbuffer gerendert wird. Der Compositor stellt diese Buffer als Fenster auf dem Bildschirm dar. Im Internet findet man sehr ausführliche Artikel über Architektur und Benutzung von Quartz [quab].

4.3.1 Architektur

Quartz ist ähnlich wie das Grafiksystem von Windows aufgebaut. Es besteht aus einer Laufzeitbibliothek (Quartz2D) die in Anwendungen eingebunden wird und Routinen für das Zeichnen von Grafikprimitiven zur Verfügung stellt. Dies beinhaltet Punkte, Linien, Bézier-Kurven, Bitmaps, Fonts usw. Da alle Objekte durch Vektoren beschrieben werden, sind auch Transformationen einfach möglich. Für die Anwendung ist es egal, ob am Ende auf dem Bildschirm, in ein Bitmap oder in ein PDF ausgegeben wird. Der zweite Teil von Quartz ist der Quartz-Compositor. Er ist das Kernstück der Fensterverwaltung und kopiert die virtuellen Fenster der Anwendung auf den Bildschirm. Die Anwendung selber muss dazu nicht benachrichtigt werden.

Abbildung 4.10 gibt einen groben Überblick zur Architektur. Grundlage für die Grafikdarstellung ist

OpenGL. Darauf baut der Quartz-Compositor auf, welcher ähnliche Aufgaben wie der Desktop-Window-Manager von Windows-Vista erfüllt. Eine XML-Beschreibungssprache wie XAML gibt es hier nicht.

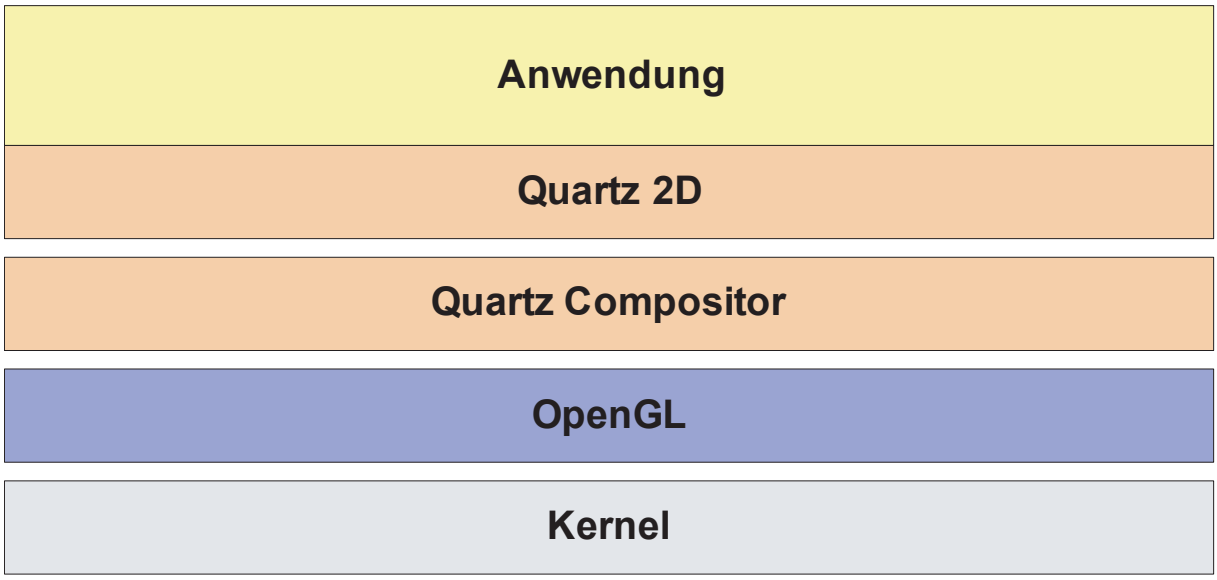

Abbildung 4.10: Quartz Architektur

Erst ab der Version Mac-OSX-10.2 ist Quartz auch in der Lage, Beschleunigungshardware zu benutzen. Der Desktop wird als OpenGL-Szene modelliert. Fenster werden zu Polygonen und der Fensterinhalt zu Texturen. Seit der Version 10.4 Tiger gibt es auch Quartz-2D-Extreme, welches die neuen 3D-Grafikkarten nutzt. Mit Hilfe von Pixel Shadern werden viele Funktionen in der GPU ausgeführt. Für den Anwender ist es transparent, ob seine Anweisungen in der GPU oder CPU ausgeführt werden. Eine weitere Neuerung in Mac-OSX 10.4 ist CoreImage, welches eine umfangreiche Sammlung von Filteralgorithmen zur Verfügung stellt. Diese Filter werden, wenn möglich, von der Grafikhardware ausgeführt. Ansonsten wird auf eine Softwareimplementierung zurückgegriffen.

4.3.2 API

Die Quartz-2D API wird als Teil eines Frameworks für Mac-Anwendungen eingebunden. Quartz-2D arbeitet mit der Metapher einer Papierseite, auf die gezeichnet wird. Objekte die gezeichnet sind, können nicht mehr verändert werden, es können aber weitere Objekte gezeichnet werden, die die vorhergehenden überdecken. Die Reihenfolge der Operationen ist also sehr wichtig. Intern arbeitet Quartz-2D dabei mit dem PDF-Format. Abhängig vom Kontext der virtuellen Papierseite, kann auf den Bildschirm, auf einen Drucker, in eine Datei oder andere Formate ausgegeben werden.

Um mit Quartz-2D etwas zu zeichnen, erzeugt man sich zuerst einen Grafik-Kontext (Graphics Context). Mit diesem führt man dann seine Zeichenoperationen aus. Es muss nicht darauf geachtet werden, zu

welchem Ausgabemedium der Kontext gehört. Quartz-2D kümmert sich um alle nötigen Berechnungen. Abbildung 4.11 zeigt Beispielcode zum Erzeugen eines Bitmapkontext und einige Zeichenoperation. Eine detaillierte Beschreibung zum Arbeiten mit Quartz-2D gibt [quac].

4.3.3 Look and Feel (Aqua)

Der gute Ruf von Apple-Betriebssystemen basiert auch auf dem sehr eleganten grafischen Aussehen. Jedes Programm folgt dem gleichen Styleguide. Dadurch wirkt es wie aus einem Guss gefertigt. Apple hat sehr früh die Wichtigkeit eines solchen Styleguides erkannt. Im Buch „Human interface guidelines: the Apple desktop interface“ [app87] wurden bereits 1987 einheitliche Richtlinien festgehalten.

Die neue Oberfläche von MaxOSX bietet viele interessante und nützliche Features. Um sich wiederholende Aufgaben zu automatisieren gibt es den Automator. Um schnell einen Überblick über alle offenen Fenster zu bekommen, kann man sie als Thumbnails nebeneinandern darstellen (Exposé). Abbildung 4.13 zeigt das sogenannte Dashboard. Es bietet Zugriff auf kleine Tools und nützlicher Information (ähnlich den Gadgets bei Windows). Der Finder und Spotlight helfen beim Organisieren und Finden von Daten auf dem Computer. Weiterhin gibt es integrierte Schrift- und Spracherkennung, Profile für jeden Nutzer und Hilfen für Menschen mit eingeschränktem Sehvermögen (Vergrößerung, Vorlesen).

4.4 Looking-Glass

Looking-Glass (LG3D) ist anders als Quartz und Avalon. Es gehört nicht zu einem Betriebssystem, sondern ist ein in Java geschriebenes 3D-Desktopsystem, also Plattformunabhängig. Unter Solaris x86 und Linux gibt es die Möglichkeit, existierende XWindows-Programme zu starten. Ausserdem gibt es eine Schnittstelle für dreidimensionale Anwendungen.

Da Looking-Glass ein opensource Projekt ist, kann man aktiv an der Entwicklung teilhaben. Es gibt eine aktive Community, die immer wieder Neues beisteuert. Allerdings ist es eher noch in einem Proof-Of-Concept Status, und nicht wirklich praktisch einsetzbar. Trotzdem scheint es für Prototypen von 3D-Anwendungen gut geeignet, weshalb es hier genauer vorgestellt wird.

Die folgenden Informationen sind aus Artikeln von der Looking-Glass-Homepage und der Java-Homepage bei Sun zusammengetragen. Da sich Looking-Glass schnell entwickelt, empfiehlt es sich für aktuelle Informationen auf der Homepage [lg3a] nachzuschauen. Es gibt immer wieder neue Dokumentationen und Beispielanwendungen. Ausserdem kann man seine Fragen im Forum direkt von den Entwicklern beantworten lassen.

```
CGContextRef MyCreateBitmapContext (int pixelsWide, int pixelsHigh) {
    CGContextRef    context = NULL;
    CGColorSpaceRef colorSpace;
    void *          bitmapData;
    int             bitmapByteCount;
    int             bitmapBytesPerRow;
    bitmapBytesPerRow   = (pixelsWide * 4);
    bitmapByteCount     = (bitmapBytesPerRow * pixelsHigh);
    colorSpace = CGColorSpaceCreateWithName(kCGColorSpaceGenericRGB);
    bitmapData = malloc( bitmapByteCount );
    if (bitmapData == NULL) {
        fprintf (stderr, "Memory␣not␣allocated!");
        return NULL;
    }
    context = CGBitmapContextCreate (bitmapData, pixelsWide, pixelsHigh, 8, bitmapBytesPerRow,
                                     colorSpace, kCGImageAlphaPremultipliedLast);
    if (context== NULL) {
        free (bitmapData);
        fprintf (stderr, "Context␣not␣created!");
        return NULL;
    }
    CGColorSpaceRelease( colorSpace );
    return context;
}

...

// Zeichenoperationen
CGRect myBoundingBox;
myBoundingBox = CGRectMake (0, 0, myWidth, myHeight);
myBitmapContext = MyCreateBitmapContext (400, 300);
CGContextSetRGBFillColor (myBitmapContext, 1, 0, 0, 1);
CGContextFillRect (myBitmapContext, CGRectMake (0, 0, 200, 100 ));
CGContextSetRGBFillColor (myBitmapContext, 0, 0, 1, .5);
CGContextFillRect (myBitmapContext, CGRectMake (0, 0, 100, 200 ));
myImage = CGBitmapContextCreateImage (myBitmapContext);
CGContextDrawImage(myContext, myBoundingBox, myImage);
CGContextRelease (myBitmapContext);
CGImageRelease(myImage);
```

Abbildung 4.11: Beispielcode für das Erzeugen eines Kontexts zum Zeichnen eines Bitmaps und einige einfache Zeichenoperationen. Abbildung 4.12 zeigt einen Screenshot.

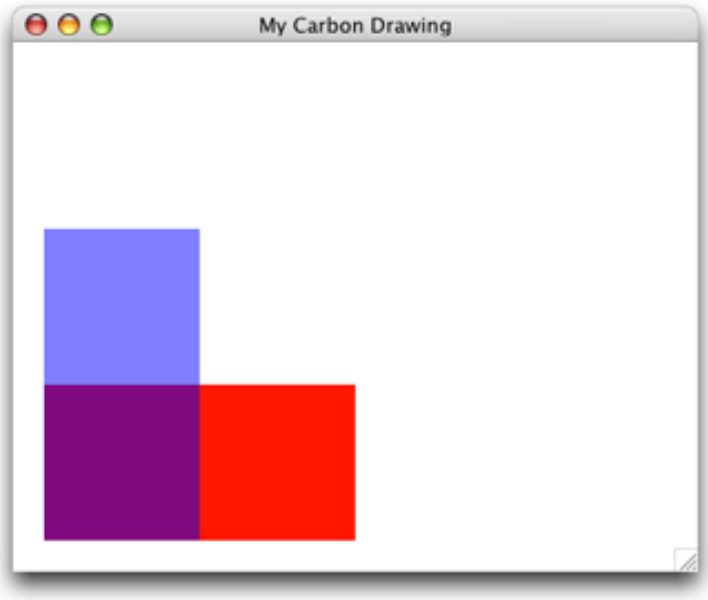

Abbildung 4.12: Screenshot des Beispielcodes aus Abbildung 4.11

Abbildung 4.13: Das Dashboard bietet Zugriff auf kleine Tools und Informationen.

4.4.1 Architektur

Looking-Glass baut auf Java-Technologie auf. Java bietet im Vergleich zu anderen Sprachen ein Produktivitätsgewinn. Es existieren sehr viele Frameworks und Bibliotheken, die man benutzen kann. Es gibt hervorragende Entwicklungsumgebungen und eine riesige Comunity, die bei Problemen helfen kann. Hinzu kommt, dass es frei verfügbar ist und eine Menge Sicherheitsfeatures anbietet. Java bietet also mit seiner Stabilität und der Menge an Bibliotheken eine solide Basis, um neue Technologien zu erforschen.

Looking-Glass baut auf Java3D auf und erweitert es um z.B. einem 3D-Windowmanager und einer einfachen Animations-Api. Java3D bietet eine einfach Schnittstelle für die Erzeugung und Steuerung von 3D-Objekten und ihrer Umgebung. Es werden jedoch nicht alle Möglichkeiten von Java3D in der LG3D-Api angeboten. Dadurch können 3D-Anwendungen später auch auf leistungsschwächeren Systemen, wie z.B. einem Autonavigationssystem, funktionieren.

Die aus Java3D stammende Client-Server Szenengraphentechnologie erlaubt es mehreren Prozessen, die gleiche 3D-Umgebung zu teilen. Anwendungen können in einem eigenen Prozess oder sogar auf einem anderen Rechner laufen, und ihr 3D-Interface in ein einziges 3D-Universum, das auf dem Server läuft, darstellen. So wird eine saubere Trennung zwischen Anwendung und Plattform möglich. Bei dieser Technologie werden Teile des Szenengraphen zwischen Client und Server ausgetauscht. Das bedeutet weniger Kommunkation als bei der X-Server Technologie. Dort werden Bereiche des Bildschirms als komprimierte Bitmaps ausgetauscht. Bei einem Szenengraphen kann eine Animation z.B. auf dem Client laufen, ohne dass Information zwischen Client und Server ausgetauscht werden müssen. Beim X-Server muss jedes Einzelbild der Animation ausgetauscht werden.

Abbildung 4.14 zeigt einen groben Überblick über die LG3D-Architektur. Die Basis bildet die virtuelle Maschine von Java mit all ihren Bibliotheken, insbesondere Java3D (läuft auch auf Clientseite, ist aus Gründen der Übersicht dort nicht noch einmal eingezeichnet). Darauf bauen die LG3D-Szenengraphklassen und der Display-Server auf. Hier erfolgt die Kommunikation zur Integration von X-Anwendungen. Das funktioniert nur, wenn man den speziell entwickelten X-Server startet. Dieser X-Server ist um eine Capture-Funtkionalität erweitert. Die visuelle Representation der unmodifizierten X-Anwendung wird an den Display Server geschickt, der die Darstellung im 3D-Raum organisiert. Für sogenannte LG3D-Aware-Anwendungen gibt es LG3D-Bibliotheken, die mit dem Displayserver kommunizieren. Wenn man auf den X-Server verzichtet, läuft LG3D auf jedem Java-fähigen Betriebssystem, dann aber nur mit LG3D-Anwendungen.

Die LG3D-Szenengraphklassen basieren auf Java3D. Da sie teilweise den gleichen Namen tragen wie die entsprechenden Java3D-Klassen, aber nicht immer das gleiche machen, muss man als Entwickler genau

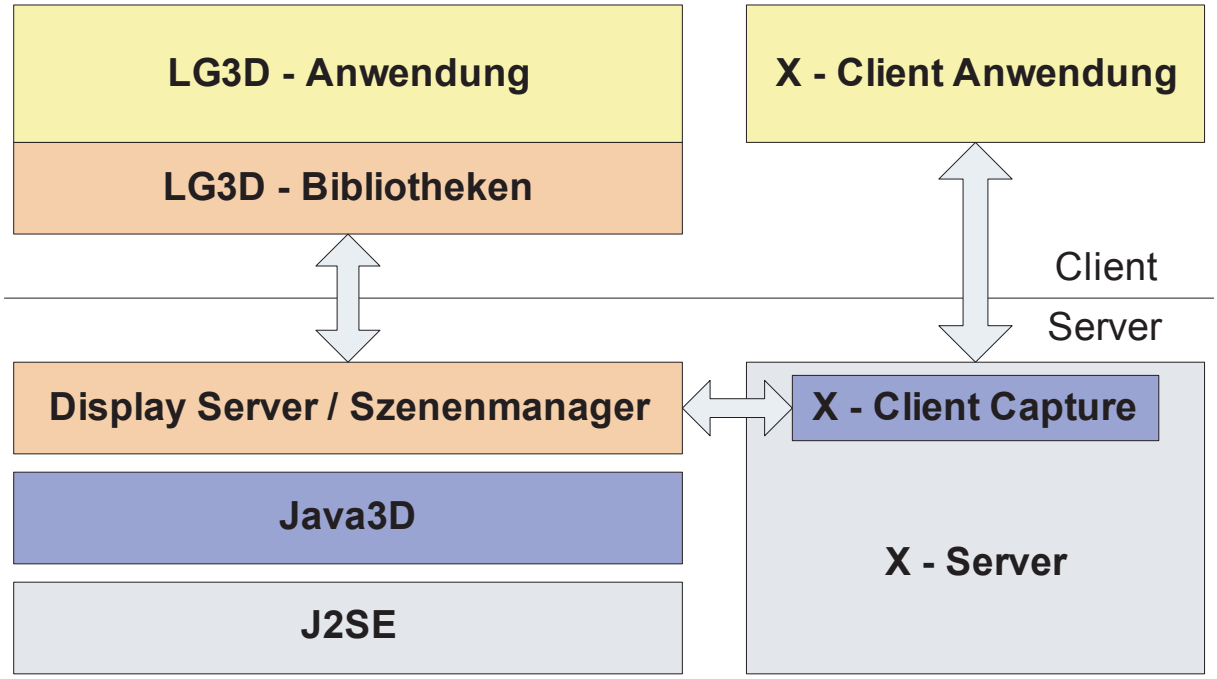

Abbildung 4.14: Looking-Glass-Architektur

darauf achten, welche Klassen man verwendet. Eigentlich sollte nur mit den LG3D-Klassen gearbeitet werden. Ähnlich dem Komponentenmodel von Swing gibt es hier entsprechende 3D-Komponenten-Klassen. Da sich die Api aber noch in der Entwicklung befindet, gibt es auch Anwendungen die Java3D-Klassen benutzen, was leicht zu Verwirrung führt. Es gibt z.B. eine Shape3D Klasse sowohl bei Java3D, als auch bei Looking-Glass. Inhaltlich sind beide dafür da, dreidimensionale geometrische Daten zu halten. In der Benutzung gibt es aber kleine Unterschiede.

4.4.2 API

Für LG3D-Anwendungen gibt es wie unter Java üblich Pakete mit Klassen und Interfaces. Abbildung 4.15 gibt einen Überblick zu den Klassen, die dem Client zur Verfügun stehen. Die auf Java3D basierenden Szenengraphklassen (blau) geben eine Untermenge der Java3D-Funktionalität an die Looking-Glass-Anwendungen weiter. Sie ähneln sehr den Java3D-Klassen, sind aber an manchen Stellen vereinfacht. Eine LG3D-Anwendung arbeitet normalerweise mit den orangen Klassen. Das sind zum einen Komponenten-Klassen, die den AWT-Klassen in Name und Funktionalität sehr ähneln (Paket org.jdesktop.lg3d.wg) und zum anderen vorgefertigte Klassen für geometriche Körper, Erscheinung und Verhalten wie Cylinder, Disc, ColorCube, SimpleAppearence und viele mehr (Paket org.jdesktop.lg3d.utils.shape und ...utils.behavior). Die gelben Klassen sind für den Zusammenhalt aller Anwendungen zuständig.

Component3D ist die Oberklasse für alle Komponenten. Sie kann ein 3D-Objekt in Form eines LG3D-

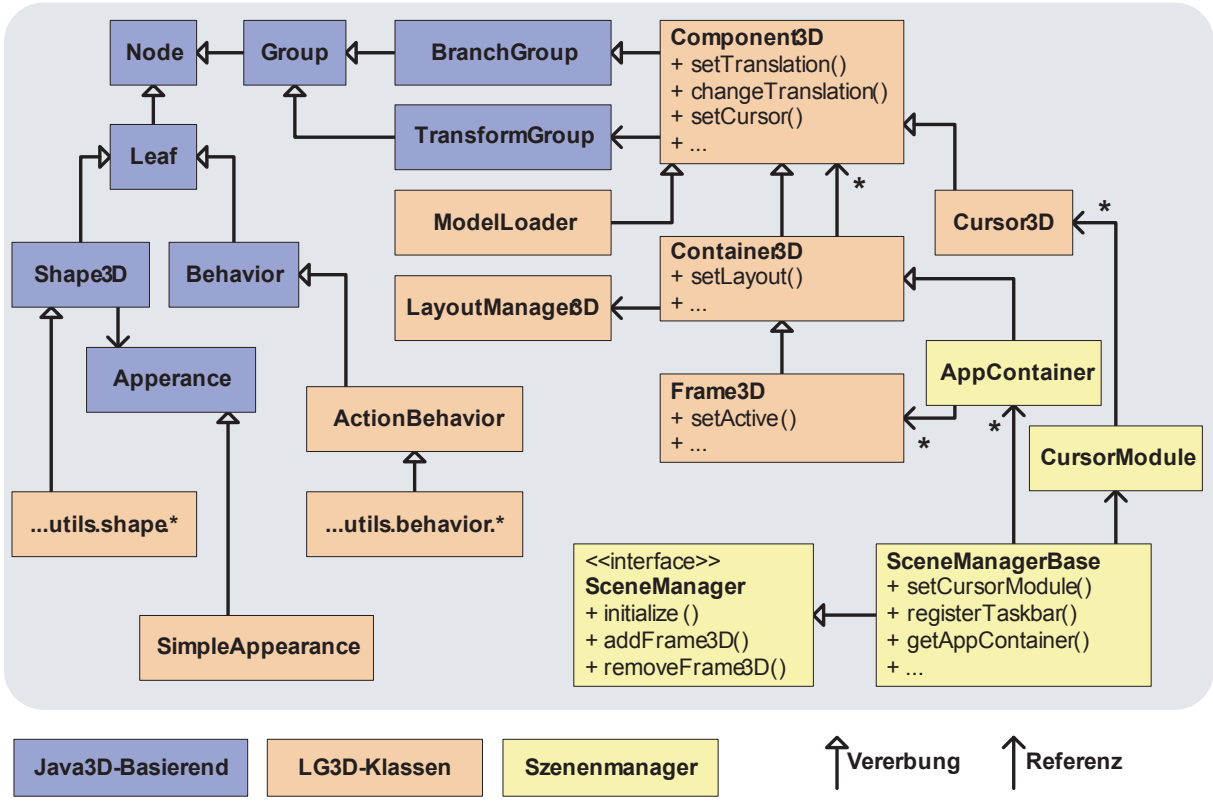

Abbildung 4.15: Aufbau von Looking-Glass auf Clientseite

Szenengraphes enthalten, sie kann aber kein anderes Component3D-Objekt enthalten. Dafür gibt es Container3D. Container3D kann mehrere Component3D-Objekte enthalten, aber keine Szenengraphobjekte. Diese müssen erst einem Component3D-Objekt hinzugefügt werden. Container3D kennt ausserdem einen 3D-LayoutManager, der bei der Anordnung der Komponenten hilft. Von Container3D wird Frame3D abgeleitet. Diese Klasse enthält eine LG3D-Anwendung und wird vom Szenenmanager (Klasse SceneManagerBase) verwaltet. Alle Komponenten implementieren das Interface LgEventSource, können also Ereignisse erzeugen.

Die Animationsunterstützung ist eine Funktionalität, die bei den LG3D-Komponenten im Vergleich zu den AWT-Komponenten neu ist. Es gibt Set-Methoden, die einen Wert (Winkel, Position, Skalierung) sofort ändern und Change-Methoden, die den Wert sichtbar innerhalb einer gegeben Zeit verändern. Dem Entwickler kann dabei egal sein, mit wieviel Bildern pro Sekunde die Animation abläuft, da er nur die Gesamtdauer angibt. Die Art der Wertänderung ist austauschbar, indem der Komponente eine Animationsklasse übergeben wird. Wenn also z.B. die Position einer Komponente geändert wird, kann diese erst beschleunigt und dann wieder gebremst werden, oder einfach mit konstanter Geschwindigkeit bewegt werden. Das hängt von der Implementierung der Animationsklasse ab.

Anhang C beschreibt eine kleine LG3D-Beispielanwendung aus dem Tutorial der Homepage.

4.4.3 Look and Feel

Bei Looking-Glass gibt es kein richtiges Look and Feel. Anders als bei Windows und Mac, wo professionelle Designer an einem einheitlichen, konsistenten Erscheinungsbild arbeiten, gibt es bei Looking-Glass kaum richtige Designer. Die Entwickler und Mitglieder der Community diskutieren, wie es aussehen könnte, und versuchen Beispiele dazu zu machen. Die Qualität dieser Beispiele kann selten mit denen von Windows und Mac mithalten. Das liegt nicht an mangelnden Fähigkeiten von Looking-Glass, sondern einfach an fehlender Manpower.

Für einige Aufgaben wie Beleuchtung, Hintergrund und Taskbar gibt es Implementierungen. Inzwischen gibt es auch ein Startmenu, in dem alle Beispielanwendungen integriert sind. Looking-Glass bietet Schnittstellen, mit denen sich jeder seinen eigenen Desktop gestalten kann. Beleuchtung und Hintergrund, die von der aktuellen Tageszeit abhängen, wären eine Möglichkeit. Abbildung 4.16 zeigt einen LG3D-Desktops. Es sind drei X-Anwendungen gestarted und eine 3D-Anwendung im Vordergrund (CD-Browser). Es gibt eine globale Lichtquelle rechts oben und einen halbdurchsichtigen Taskbar unten, auf dem Thumbnails der laufenen Anwendungen und Icons zum Starten von Anwendungen dargestellt werden. Abbildung 4.17 zeigt zwei 3D-Anwendungen, die als Test für diese Arbeit geschrieben wurden.

Abbildung 4.16: LG3D-Desktop mit einer LG3D-Anwendung und drei X-Anwendungen. Unten der Taskbar mit Thumbnails der Anwendugnen.

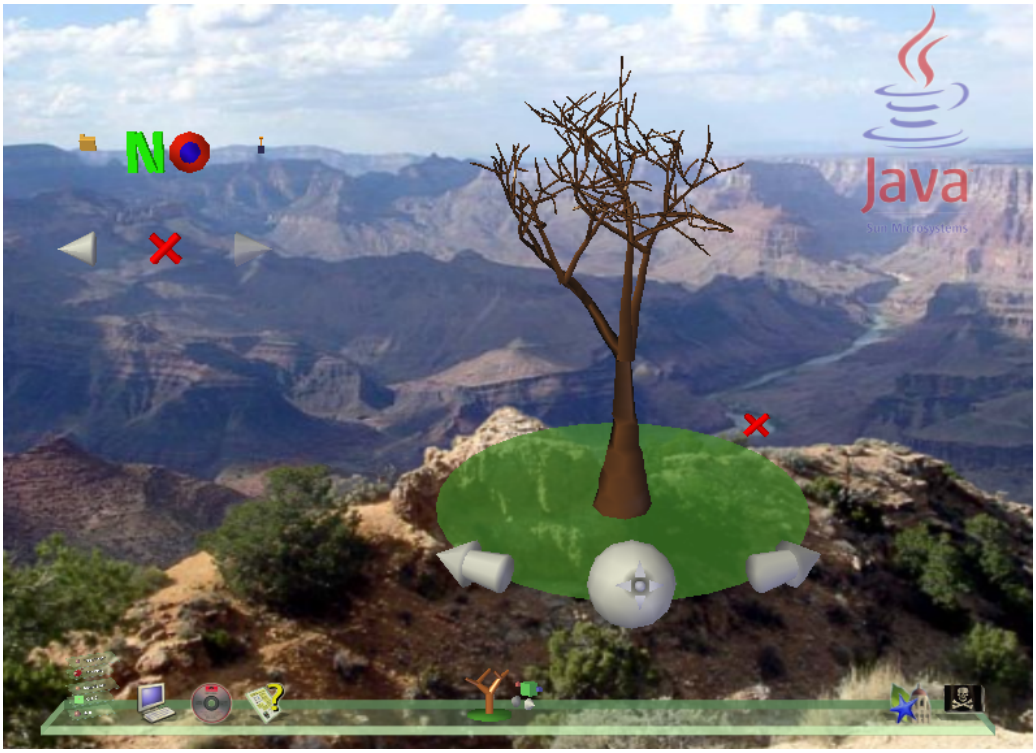

Abbildung 4.17: 2 3D-Anwendungen. Rechts werden 3D-Bäume generiert und angezeigt, links oben ein Icon-Viewer mit dem 3D-Icons angezeigt werden.

Windows-Vista und Mac-OSX benutzen zwar moderne Grafiksysteme, hauptsächlich werden die neuen Möglichkeiten aber eingesetzt, um das bekannte 2D-Fenstersystem schöner darzustellen. Looking-Glass ziehlt auf echte 3D-Anwendungen ab. LG3D ist selber ein Technologietest, es ist also sinnvoll auch auf Anwendungsebene zu experimentieren. Es gilt herrauszufinden, welche Möglichkeiten die dritte Dimension für normale Anwendungen bringt. Aus diesem Grund eignet sich Looking-Glass sehr gut für diese Arbeit.

4.5 Zusammenfassung

Apple hat mit Quartz eine neue Ära im Bereich der Grafiksysteme begonnen. Microsoft und auch die Opensource-Szene, mit Projekten wie Looking-Glass, ziehen nach. Der Grafikprozessor und Grafikspeicher macht eine ähnliche Entwicklung wie der Hauptprozesser und Hauptspeicher durch. Da viele Anwendungen zugleich die Grafikhardware nutzen wollen, muss Rechenleistung und Speicher fair aufgeteilt werden. Die GPU wird von einem Scheduler verwaltet und der Grafikspeicher wird virtualisiert. Die Möglichkeiten moderner Grafikhardware sind dadurch für alle Anwendungen besser nutzbar.

Der Desktop wird in Zukunft als 3D-Szene modeliert. Fenster werden zu Polygonen und Anwendungen

zeichnen in eine Textur für die Fensterpolygone. Anwendungen können in Zukunft also nicht mehr in den Bereich anderer Anwendungen zeichnen. Ein weiterer Vorteil ist, dass jetzt mit Vektorgrafik gearbeitet wird. Zeichenoperationen können so auf verschiedene Ausgabemedien gerendert werden, und sind z.B. unabhängig von der Auflösung des Monitors. Die neuen Grafiksysteme basieren meist auf Technologien wie OpenGL und Direct3D (oder darauf aufbauende wie Java3D).

Im Moment werden die neuen grafischen Möglichkeiten für besseres Aussehen des alten 2D-Fensteransatzes genutzt. Besonders Projekte wie Looking-Glass versuchen aber neue dreidimensionale Wege zu erkunden. Auch Windows-Vista wird es sehr einfach machen, die dritte Dimension in jeder Anwendung zu benutzen. Ob das wirklich ein Vorteil ist und welche Anwendungen wie davon profitieren können, lässt sich schwer abschätzen. Nur die Praxis kann in den nächsten Jahren zeigen, was vom Nutzer angenommen wird und sich durchsetzt.

Tabelle 4.1 zeigt eine Übersicht der vorgestellten Systeme.

	Windows-Presentation-Foundation	**Quartz**	**Looking-Glass**
Plattform	Windows-Vista	Mac-OSX	beliebig
Performance	basiert auf DirectX; passt sich dem Rechner an; 3 Abstufungen: Aero-Glass (3D, PixelShader), Aero (3D, DirectX 9), Classic (2D)	basiert auf OpenGL; da Hardware und Betriebssystem zusammengehören kein Performanceproblem	basiert auf Java3D (kann mit OpenGL oder DirectX laufen); empfohlene Mindestanforderung: CPU 1.4GHz, RAM 512MB, OpenGL 1.2
Features (unvollständig)	Trennung Design und Programmlogik durch XAML; Desktopcomposition; Transparente Fenster; Pixelshader; Gadgets, Methainformation,...	Desktopcomposition; Grafikkontexte (Unabhängig vom Ausgabemedium); Pixelshader; Finder, Dashboard,...	X-Anwendung lauffähig; Zweieinheilb-Dimensionales Nutzererlebnis; Support von Swing; Client-Server-Szenengraph; einfaches Animationssystem
Abwärtskombatibilität	GDI komplett als Softwareimplementierung verfügbar	alte Anwendungen müssen neu Compiliert werden oder in einer speziellen Umgebung laufen	X-Anwendungen laufen wenn spezieller X-Server gestartet wird
Benutzung (aus Entwicklersicht)	XML-Basierte Schnittstelle (XAML)	Bibliotheken zum Einbinden	Java-Pakete; Komponentenmodell, ähnlich AWT
Toolsupport	Visualstudio, Trennung für Programmierer und Designer	Xcode 2.0; Frameworks für Anwendungen (Cocoa); OpenSource-Tools aus Unix-Welt	keine speziellen Tolls
Entwicklungsstand	beta, vorraussichtlich 2007 fertig	im Einsatz, aktuell Version 10.4 mit Nutzung der Grafikhardware	beta, aktuell Version 0.8.0
Weitere Entwicklung	Wird wahrscheinlich erfolgreich Nachfolge von XP antreten und Marktführer bleiben.	neue Hardware bringt neue Features,z.B. GPS in Notebooks; bleibt führend bei professionellen Multimedia-Anwendungen	läuft als OpenSource-Projekt weiter

Tabelle 4.1: Vergleich der vorgestellten Systeme

5 NakedObjects

Der Name NakedObjects erscheint auf den ersten Blick etwas seltsam. Dahinter verbirgt sich aber ein sehr interessantes Vorgehen für Geschäftsanwendungen. Es geht dabei um verhaltensvollständige Geschäftsobjekte, die dem Nutzer direkt zugänglich gemacht werden. Verhaltensvollständig bedeutet, daß alles was zum Objekt gehört, also seine Daten und sein Verhalten, in dem Objekt enthalten ist. Das „Nackt“ in NakedObjects bezieht sich also nicht auf das Objekt selber, sondern darauf, daß keine zusätzlichen Konstrukte um das Objekt herum notwendig sind.

Das NakedObjects-Prinzip wurde von Richard Pawson im Rahmen seiner Doktorarbeit entwickelt [Paw04]. Seit 1999 gibt es ein auf Java basierendes Framework von Robert Matthews. Das NakedObjects-Framework hat sich seitdem ständig weiterentwickelt, und ist inzwischen eine OpenSource-Projekt bei sourceforge.net [Nakb]. 2002 ist ein Buch mit dem Titel „naked objects“ erschienen [PM02]. In diesem Buch werden ausführlich die Hintergründe und Vorteile der NakedObjects Idee erläutert und die Version 1.0 des Frameworke beschrieben. Richard Pawson und Robert Matthews haben eine eigene Firma gegründet, die „Naked Objects Group“, um Software mit Hilfe des NakedObjects-Prinzips zu entwickeln. Seit 2003 haben sie einen großen Auftrag der irischen Regierung, der zeigen wird, ob das NakedObjects-Prinzip wirklich praxistauglich ist.

Das nächsten Kapitel beschreibt die Idee und das Konzept genauer. Danach wird dieser Ansatz mit anderen verglichen. Besonders wird dabei auf Enterpris Java Beans eingegangen, da EJB ein akzeptierter Standard für Geschäftsanwendungen ist. Das vorletzte Kapitel geht auf praktische Anwendungsmöglichkeiten ein. Dort wird z.B. diskutiert, ob wirklich Anwendungen mit dem NakedObjects-Framework möglich sind. In der Zusammenfassung am Ende werden die Vor- und Nachteile noch einmal in einer Tabelle dargestellt. Im Anhang D befindet sich ein kleines Beispiel einer NakedObjects-Anwendung.

5.1 Konzept

Objektorientieredes Programmieren (nachfolgend OOP) ist heutzutage weit verbreitet und anerkannt. Ein Programm aus Objekten aufzubauen, macht es deutlich einfacher, große Anwendungen zu entwickeln. Obwohl fast jede neue Anwendung und Sprache auf Objekten basiert, ist die grundlegende Philosophie

dahinter oftmals schlecht verstanden. Das Benutzen einer objektorientierten Sprache führt nicht zwangsweise zu gutem Code. Selbst unter professionellen Programmierer findet man oft grundlegende Fehler wie z.B. zu lange Methoden, zu große Objekte oder Codeduplikationen. Es hängt nicht von der Technologie ab, ob ein Programmierer gut oder schlecht ist, sondern von seinen Fähigkeiten. Die Technologie beeinflusst nur seine Effizienz. Programmieren ist mehr, als nur Anweisungen in einer Programmiersprache einzutippen [HT03].

Die grundsätzliche Idee beim OOP ist, dass ein Objekt ein Ding vollständig modelliert. Ein Objekt besteht aus Daten und Methoden. Ausserdem kann es mit anderen Objekten kommunizieren. Oft haben die Objekte im Programm einen direkten Bezug zu Objekten aus der Welt des Problemes (Problemdomäne), das man lösen will. Kanditaten für Programmklassen sind z.B. Substantive aus der Problembeschreibung und fachliche Kernbegriffe [OHJ+01]. Die Idee bei NakedObjects ist es, die Programmobjekte dem Anwender direkt zugänglich zu machen. Es wird nur das Geschäftsmodell implementiert, alle anderen Aufgaben wie Persistenz und Visualisierung werden von einem Framework übernommen. Damit man mit diesen Objekten auch arbeiten kann, müssen sie verhaltensvollständig sein (behaviourally complete). Das bedeutet, dass Daten und zugehörige Verarbeitungsmethoden immer in einem Objekt sind, und nicht über mehrere verteilt. Da es zwischen dem Anwender und den Geschäftsobjekten keine Steuerschicht gibt, ist der Entwickler gezwungen, objektorientiert zu arbeiten. Eine Anwendung ohne verhaltensvollständige Geschäftsobjekte wäre nicht intuitiv benutztbar. Es ergeben sich folgende Vorteile (nach [PM02]):

- höhere Produktivität, da keine GUI zu schreiben
- leichter wartbare Systeme, durch verhaltensvollständige Objekte
- verbesserte Benutzbarkeit, durch objektorientierte Oberfläche[1]
- fachliche Anforderungen einfacher erfassbar, da NakedObjects gemeinsame Sprache
- testgetriebendes Entwickeln sehr einfach machbar

Die klassischen Vorteile von OOP wie Agilität und Kapselung bleiben dabei erhalten. Eine Änderungen in der Modellschicht zieht an nur einer Stelle Äenderungen im Programmcode nach sich. Das Programm kann dadurch sehr einfach verändert und angepasst werden (Agilität). Gute Kapselung bedeutet, dass der Programmcode aus vielen Teilen besteht, die über wohldefinierte Schnittstelle kommunizieren. Diese Schnittstellen sollten so platziert sein, dass sie so klein wie möglich sind. Genau das ist bei verhaltensvollständigen Geschäftsobjekten der Fall. Kapselung ist das beste Mittel gegen Entropiezunahmen im Quelltext.

Eine Referenzimplementierung des NakedObjects-Prinzips gibt es in Form eines OpenSource-Frameworks. Es müssen nur die Geschäftsobjekte entwickelt werden. Das Framework generiert automatisch eine grafi-

[1] Der Zustand eines Objektes bestimmt welche Aktionen möglich sind.

sche Oberfläche und kümmert sich um Persistenz. Vorraussetzung ist, dass sich die Klassen an bestimmte Namensregeln halten, damit das Framework durch Reflection die Klassen analysieren kann. Die Visualisierung ist austauschbar und wird in Kapitel 6 genauer untersucht.

NakedObjects ist ein sehr interessanter, radikaler Ansatz. Der Nachteil dabei ist, dass es NakedObjects schwer hat, seinen Weg in die Praxis zu finden. Dort ist Zuverlässigkeit sehr viel wichtiger als gutes Design. Neue Ansätze sind nicht so gut untersucht und meist noch nicht perfekt. Aber schon jetzt kann man NakedObjects für Prototypen, Proof-of-concepts, Tests oder auch in der Lehre einsetzen.

5.2 Vergleich mit anderen Ansätzen

In diesem Abschnitt wird diskutiert, welche Vorgehensweisen es in der Praxis gibt, und wie sich diese von NakedObjects unterscheiden. Es ist schwierig, die gesamte Welt der Softwaretechnik in einem Abschnitt zusammenzufassen. Deshalb sollen in 3 Untersektionen verschiedene Aspekte betrachtet werden. Zuerst wird die NakedObjects-Architektur mit der weitverbreiteten 4-Schichten-Architektur verglichen. Danach geht es um die Probleme bei firmeninternen Frameworks, die mit dem NakedObjects-Framework vergleichbar sind. In dem letzten Teil wird ein Vergleich mit Enterprise Javabeans gezogen.

5.2.1 4-Schichten-Architektur

Die meisten Geschäftsanwendungen werden nach der 4-Schichten-Architektur entwickelt [Paw04]. Diese Architektur wurde 1995 von Kyle Brown vorgeschlagen, wobei sie schon vorher Anwendung fand [Bro95]. Die 4 Schichten werden in diesem Kapitel Präsentations-, Steuer-, Domain- und Persistenzschicht genannt. Die Bezeichnungen variieren aber je nach konkreter Implementierung, wie z.B. beim Model-View-Controller-Pattern. Jede der 4 Schichten hat eine bestimmte Verantwortung (separation of concern). Durch diese Aufteilung soll die Komplexität der Anwendung geteilt und somit verringert werden.

Die Trennung der Verantwortlichkeiten führt zu sehr flexiblen Lösungen. Es ist sehr einfach, für ein Datenobjekt verschiedene Ansichten zu erzeugen. Der Nachteil ist, dass die Geschäftsobjekte in verschiedener Form auf alle 4 Schichten verteilt sind. Die Objekte in der Domainschicht werden zu reinen Datenobjekten, die von den Objekten in der Steuerschicht gesteuert werden. Das führt zu dem funktionellen Programmieren, von dem man durch die Objekte eigentlich weg wollte. Änderungen in der Domainschicht ziehen auch Änderungen in der Präsentationsschicht und der Steuerschicht nach sich. Obwohl man objektorientierte Technologie in den Schichten verwendet, gehen Vorteile wie Agilität verloren.

Der NakedObjects-Ansatz ist angetreten, die Nachteile zu beseitigen und die Vorteile zu behalten. Dazu wird ein Teil der Steuerschicht mit der Domainschicht zusammengeführt. Die Trennung zwischen Verhalten und Daten wird rückgängig gemacht. Das erscheint zunächst wie ein Schritt rückwärts. Der Grund für die Trennung war aber ursprünglich, dass jedes Objekt von seiner Darstellung getrennt werden soll. Das kann man durch eine automatisch generierte Präsentations- und Steuerschicht erreichen. Der Vorteil einer flexiblen Objektdarstellung bleibt erhalten und der Nachteil der Objektaufteilung wird beseitigt.

Der Ansatz von automatisch genererierter Präsentation ist nicht neu. Die meisten Ansätze sind aber mehr daten- als objektorientiert und die Motivation ist meist einfacherere Interface-Entwicklung. Bei NakedObjects stellt die Visualisierung direkt die Domainschicht dar. Die Motivation ist dabei, dass Geschäftsobjekte verhaltensvollständig werden. Nur verhaltensvollständige Geschäftsobjekte lassen sich sinnvoll in einer generischen Oberfläche benutzen.

Heutzutage wird Persistenz automatisch generiert. Es gibt zahlreiche Möglichkeiten wie z.B. JDO oder Hibernate. Da es hier keinen Unterschied zwichen einer normalen 4-Schichten-Architektur und dem NakedObjects-Ansatz gibt, wird darauf nicht weiter eingegangen.

Abbildung 5.1 zeigt schematisch den Unterschied zwischen NakedObjects und der 4-Schichtenarchitektur. Die Steuerschicht entfällt bei NakedObjects, weil das Verhalten in den Geschäftsobjekten enthalten ist[2]. Auf der linken Seite muss sich ein Entwickler normalerweise mindestens mit den oberen 3 Schichten beschäftigen. Es müssen Formulare, Dialoge und Menus für die Präsentationsschicht erstellt werden (die blauen Rechtecke in der Abbildung). Aus den Use-Cases werden die Prozesse der Steuerschicht abgeleitet (die roten Pfeile in der Abbildung). Alle persistenten Daten werden in Objekten der Domainschicht verpackt (gelbe Kreise) und mit einem Persistenzmechanismus verbunden. Auf der rechten Seite muss sich der Entwickler nur mit der Domainschicht beschäftigen. Die verhaltensvollständigen Geschäftsobjekte werden durch gelbe Kreise mit einem Pfeil in der Mitte dargestellt. Präsentation und Persistenz werden vollautomatisch genererierte (die blauen und hellblauen Kreise in der Abbildung).

Ein konkreter Vergleich, ein und derselben Anwendung, einmal mit der 4-Schichtenarchitektur und einmal mit NakedObjects befindet sich in einem TheServerSide-Artikel [PMH04]. Auch wenn die Anwendung nicht für echte Nutzer war, sprechen die Ergebnisse aus technischer Sicht eindeutig für NakedObjects. Usability wurde nicht untersucht. Tabelle 5.1 zeigt einen Teil der Metrik. Eine ausfürhliche Interpretation der Daten ist Teil des Artikels. Man sieht jedoch auf den ersten Blick um wieviel einfacher die NakedObjects-Variante ist.

[2]Genaugenommen existiert immer noch eine Steuerschicht. Sie wird aber automatisch generiert und enthält keine Geschäftslogik.

[3]Lines of Code (dt.: Programmzeilen)

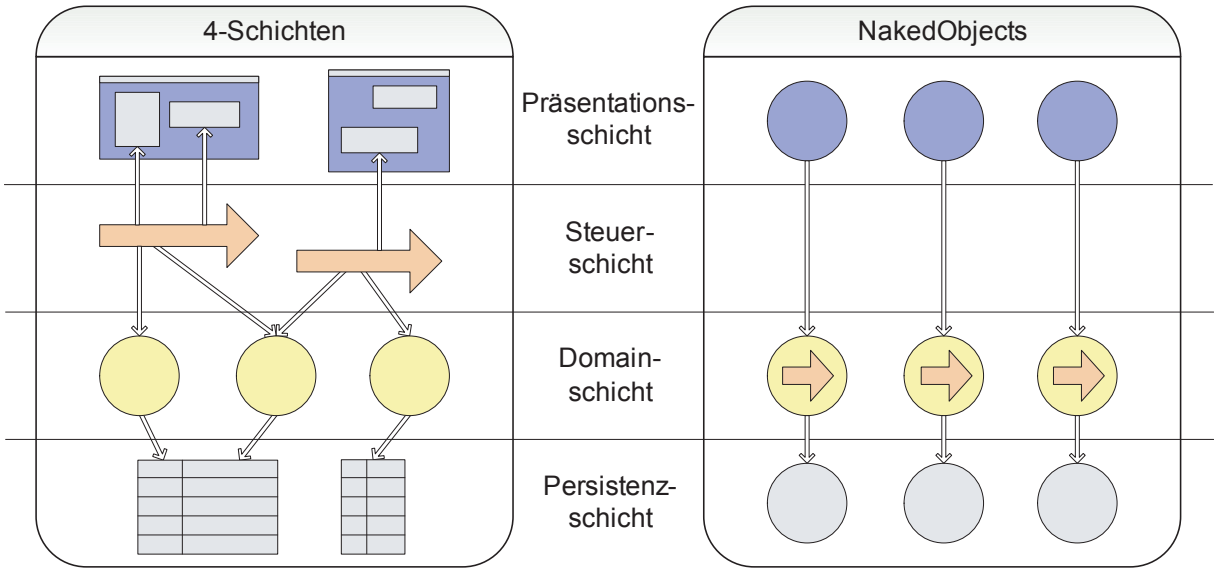

Abbildung 5.1: Architekturvergleich: 4-Schichten-Architektur, NakedObjects

	Klassen	**Methoden**	**Methoden pro Klasse**	**LOC[3]**	**LOC[3]pro Methode**
4-Schichten	190	788	4,1	7304	9,3
NakedObjects	27	230	8,5	1726	7,5

Tabelle 5.1: Vergleich einer NakedObjects-Anwendung mit einer 4-Schichten-Anwendung

5.2.2 Firmeninterne Frameworks

Viele Firmen nutzen veröffentlichte Frameworks oder entwickeln sie für ihre Geschäftsanwendungen selber. Oft kann man diese Frameworks nur nach einer Schulung benutzen, da es viele schlecht dokumentierte Teile gibt, oder Dinge nicht intuitiv funktionieren. Hinzu kommen Namenskonventionen, Styleguides, vorgeschriebene Entwicklungsabläufe und vorgeschriebene eigene und fremde Tools. Ein gut eingearbeiteter Programmierer kann effizient mit so einem Framework arbeiten, für einen Neueinsteiger ist der Aufwand enorm. Manchmal gibt es sogar undokumentierte Probleme oder Bugs, die zu Datenverlust führen, wenn man sie nicht kennt. Meist weiß jeder um die Schwächen des Frameworks, wenn es aber einmal im Einsatz ist, ist das Risiko und der Aufwand zu gross, es zu ersetzen oder grundlegend zu ändern.

Oft wird Information auseinander gerissen oder mehrfach gehalten. Das liegt unter anderem an der Benutzung externer Tools, oder an der trendbedingten übermässigen Benutzung von XML. Wenn eine Klasse zum Beispiel in einem Modellierungstool entwickelt wird, dann als XML exportiert und dann durch einen Generator in Javacode umgewandelt wird, existiert ein und dieselbe Klasse 3 mal in verschiedenen Beschreibungsformen. Duplikationen und XML-Konfigurationsdateien führen meist dazu, dass man nicht mehr genau weiss, an welcher Stelle man ändern muss, um sein Ziel zu erreichen. Im schlimmsten

Fall muss man sogar an mehreren Stellen ändern, um nicht in undefinierte Zustände zu gelangen.

Meist arbeiten Frameworks prozessorientiert. Es werden Vorgänge beschrieben, die der Nutzer abarbeitet. Problematisch ist das bei Sonderfällen, die nicht in die vorgesehenen Prozesse passen. Um auch alle Sonderfälle abzudecken, werden dem Entwickler und Nutzer Möglichkeiten angeboten, von den Prozessen abzuweichen, man baut also Hintertüren ein. Diese Hintertüren sind notwendig und auch nicht unbedingt schlecht. Es gibt genau dann Probleme, wenn Nutzer oder Entwickler nicht genau wissen, was sie tun, und nur ihr Ziel erreichen wollen, egal wie. Die Aufgabe des Frameworks ist es, den Normalfall so einfach zu machen, dass man den Sonderfall nur benutzt, wenn es sein muss. Beim Anwender erreicht man das, indem die grafische Oberfläche intuitiv, leicht verständlich und konsistent ist. Beim Entwickler bedeutet es, dass der einfache Fall soweit wie möglich von alleine funktionieren muss. Es darf nichts unnötig konfigurierbar sein, es muss immer sinnvolle Vorbelegungen geben und mächtige Tools sollten die Fleissarbeit abnehmen.

Tabelle 5.2 stellt in der linken Spalte die oft auftretenden Probleme von Frameworks für Geschäftsanwendungen dar. In der rechten Spalte wird bewertet, inwieweit dieses Problem bei NakedObjects auch besteht.

firmeninterne Frameworks	**NakedObjects**
meist Eigenentwicklungen, nicht öffentlich verfügbar	ist ein OpenSource-Framework
schlecht dokumentiert (Dokumentation nötig weil nicht intuitiv), Schulungen nötig für neue Programmierer	auch schlecht dokumentiert, aber durch intuitiven Aufbau des Frameworks trotzdem leichter Einstieg möglich
externe Tools notwendig, aber meist mangelhafte Leistung der Tools	externe Tools werden nicht gebraucht (könnten aber trotzdem nützlich sein)
Duplikation von Information	keine Duplikation, Code ist gleichzeitig Modell
viele Konfigurationsdateien	Konfiguration einfach (durch Properties-Dateien) und nur an wenigen Stellen nötig
Prozessorientiert, unflexible bei Sonderfällen	keine festen Prozesse, sehr flexible

Tabelle 5.2: Probleme von firmentinternen Frameworks im Vergleich zu NakedObjects.

Der Hauptgrund für die Probleme bei den Frameworks, ist nicht mangelhafte Fähigkeit der Programmierer, sondern die Schwierigkeit, große Softwareprojekte zu organisieren. [FPB03] beschreibt, unter anderem, sehr ausführlich, warum es schwer ist Projekte zu managen. Eines der größten Problem ist,

dass jede Firma heutzugtage im Konkurenzkampf mit anderen steht. Das führt zu Zeitdruck und Geldmangel[4]. Zeitdruck führt oft dazu, dass zuwenig getestet wird, und die erstbesten Lösungen verwendet werden. Wenn der Code einmal funktioniert, wird Refactoring meist aus Kostengründen nicht gemacht. Ausserdem wird die Wichtigkeit von guten Tools meist unterschätzt. Die Benutzbarkeit eines schlechten Frameworks kann durch gute Tools verbessert werden.

NakedObjects löst die meisten dieser Probleme gut. Allerdings steht es also OpenSource-Projekt auch nicht unter Konkurenzdruck. In der Praxis müssen oft Kompromisse aufgrund von Zeitdruck und Geldmangel gemacht werden. Deshalb ist der Vergleich mit firmeninternen Frameworks aus der Praxis nicht Fair. Ausserdem gibt es im Moment noch keine praktisch eingesetzten Anwendungen des NakedObjects-Framework. Der Beweis für Tabelle 5.2 bleibt also offen, bis es mehr praktische Erfahrungen gibt. Die nächsten Absätze sind eine ausführliche Version der rechten Spalte der Tabelle.

NakedObjects ist ein Opensource-Projekt und keine firmeninterne Entwicklung. Es gibt also auch frei verfügbare Dokumentationen. Man ist nicht verpflichtet, teure Schulungen zu besuchen, um an Information zu gelangen. Die Menge und Qualität der Dokumentationen ist leider nicht sehr gut. Der Vorteil besteht aber darin, das NakedObjects so einfach ist, das keine grossen Mengen an Dokumentationen nötig sind. Es gibt ein online verfügbares Buch, das man kostenlos lesen kann ([PM02].

Einer der Gründe, daß NakedObjects so wenig Dokumentation benötigt, ist der geringe Konfigurationsaufwand. Am Anfang muss man sogar überhaupt nichts konfigurieren. Erst wenn man tiefer in das Framework einsteigt, muss man sich mit Konfiguration beschäftigen. Das sind dann keine XML-Dateien sondern einfache Properties-Datein, wie man sie von Java kennt.

Die Java-Klassen bei NakedObjects sind eine fast 1:1 Abbildung des UML-Klassendiagrammes aus der Entwurfsphase. Man braucht also nicht unbedingt ein Designtool, um den Überblick bei einer Anwendung zu behalten. Bei NakedObjects ist der Code selber das Modell, es gibt keine vorgeschriebene Duplikation. Man kann sogar soweit gehen, auf viele der Diagramme ganz zu verzichten. Oft werden UML-Diagramme nicht mit den Änderungen im Programm aktualisiert, wodurch sie zu einer Last werden. [PMH04] beschreibt zuerst einen Ansatz der auf UML verzichtet. Der Artikel zeigt aber auch, dass man mit geeigneten Tools eine wirkliche 1:1 Abbildung zwischen dem UML-Diagramm und einem lauffähigem NakedObjects-Programm erreichen kann, sozusagen ein ausführbares UML-Diagram.

Normalerweise werden Use-Cases in der Steuerschicht einer Anwendung umgesetzt. Bei NakedObjects gibt es dafür die Aktionsmethoden. Beim Entwurf ist also darauf zu achten, dass es für jeden Use-Case eine Aktionsmethode gibt. Auch hier zeigt sich dass UML-Diagram und Code fast eine 1:1 Abbildung

[4] Geld und Zeit stehen dabei in einem funktionellen Zusammenhang, mehr Geld bedeutet weniger Zeit und andersrum. Allerdings lassen sich die beiden Größen nicht beliebig verkleinern (siehe Kapitel 2 in [FPB03]).

sind.

Da die Oberfläche generisch ist, und man nur die Geschäftsobjekte definiert, gibt es keine Prozesse bei NakedObjects. Der Nutzer bekommt Objekte mit denen er alles machen kann, was der aktuelle Zustand des Objektes erlaubt. Die Entscheidung darüber trifft nicht ein Steuerobjekt, sondern das Objekt selber. Wenn ein Kunde vom üblichen Vorgang abweichen will, ist das kein Problem. Das fordert mehr Mitdenken beim Anwender, er muss ein Verständniss für die Geschäftsabläufe haben. Auf diese Art wird Mitdenken und Erforschen aller Möglichkeiten gefördert. Ob das ein Nachteil oder Vorteil ist, hängt von der Persöhnlichkeit des Anwenders ab.

5.2.3 Eenterprise JavaBeans vs NakedObjects

Der Vergleich zwischen Enterprise Java Beans und NakedObjects liegt nahe. Allerdings stellt man schnell fest, dass es eher so ist, als ob man ein Modellflugzeug (NakedObjects) mit dem neuen Airbus (EJB) vergleicht. Ein Modellflugzeug kann sehr gut neue Technologien demonstrieren, muss sich aber nicht mit den vielen Details wie z.B. Sichereheitsstandards und Service für Fluggäste beschäftigen. Trotzdem soll hier versucht werden, einen Vergleich auf der Ebene der Technologie zu ziehen. Da es viele ausführliche Einleitungen zu EJB gibt, wird hier nur kurz die Motivation und Idee erläutert. Einen guten Einstieg bietet das Buch [Rom02].

Mit der Entwicklung von Computernetzwerken haben sich auch die Geschäftsanwendungen weiterentwickelt, von monolithischen zu verteilten Systemen. Damit gab es sehr viele neue Aspekte wie Transaktionen, Sicherheit, entfernte Methodenaufrufe, Multithreading und viele mehr, zu berücksichtigen. Firmen mussten zusätzlich zu ihrem eigentlich Geschäfstproblem noch eine Menge anderer lösen. Diese wiederverwendbaren Softwareteile, die man für ein verteiltes System braucht, werden Middleware genannt. Inzwischen gibt es von vielen verschiedenen Anbietern Middleware in Form von Application-Servern zu kaufen. Der Kunde löst nur noch sein Geschäftsproblem, und fügt diese Lösung in Form von Komponenten in den Application-Server ein. Komponenten sind wiederverwendbare Softwareteile, die einen Container brauchen um zu funktionieren. In der Praxis findet diese Wiederverwendung aber seltener als erhofft Projektübergreifend statt ([Rom02] Seite 11).

Wenn man Komponenten wirklich wiederverwenden will, ist es wichtig, dass es einen offenen Standard dafür gibt. Nur so kann man garantieren, dass die Komponenten auch in einer anderen Umgebung funktioniert. EJB ist genau so ein Standard, genaugenommen ist es eine Spezifikation in Textform und eine Menge von Java-Interfaces. Java eignet sich sehr gut, da die Sprache vergleichsweise sicher ist, auf fast jeder Plattform läuft und sehr umfangreiche Bibliotheken und Dokumentationen besitzt.

Der Vergleich von NakedObjects und EJB ist deshalb schwierig, weil sich EJB über Jahre hinweg an der Praxis entwickelt hat, wärend NakedObjects eher aus dem Labor kommt. Die meisten Probleme mit denen sich EJB beschäftigen muss (z.B. integration alter Hard- und Software, unsichere Netzwerke mit Firewalls, riesige Datenbanken, usw.) werden bei NakedObjects nicht beachtet. Tabelle 5.3 zeigt in einer Spalte die Vorteile von EJB nach [Rom02]. Daneben wird beurteilt, inwieweit NakedObjects mithalten kann.

Enterprise JavaBeans	**NakedObjects**
offener Standard, viele verschiedene Implementierungen	ist ein OpenSource-Framework aber kein Standard
kein spezielles Training für Programmier nötig („Train once, code everywhere“)	eher unbekannt, aber schnell erlernbar
Portable, EJB ist freier Standard, keine Bindung an eine bestimmte Firma	ist kein Standard, implementierung als OpenSource-Projekt
effizientes Entwickeln, da Middleware vorhanden	Aufwand ist sogar noch geringer, es müssen nur noch Geschäftsobjekte entwickelt werden
skalierbar	theoretisch skalierbar, aber kein praktischer Beweis vorhanden
riesige Community im Internet, sehr viele gute Seiten mit Resourcen	eher unbekannt, wenig Quellen und Dokumentationen im Netz vorhanden
sehr gute Toolunterstützung	wenig Tools, Qualität nicht vergleichbar mit denen von EJB
Anwendung automatisch in Schichten getrennt, gut gegen Entropie, gut für Sicherheit	man muss sich nur mit einer Schicht beschäftigen, einfacher als EJB, gleiche Vorteile

Tabelle 5.3: Allgemeine Vorteile von EJB mit einer Beurteilung inwieweit NakedObjects ähnliches bieten kann.

Eine Enterprise Bean ist eine serverseitige Komponente. Sie besteht aus mehreren Klassen, wobei der Client (ein Servlet, Applet, andere Bean,...) immer über ein Interface mit der Bean kommuniziert. Dieses Interface muss die EJB-Spezifikation erfüllen, genauso wie ein NakedObjects seine Namenskonventionen erfüllen muss. Einer der größten Unterschiede ist also, dass eine Enterprise Bean aus mehreren Klassen besteht. Hier wird also eindeutig Information mehrfach gehalten. Wenn z.B. am Interface etwas geändert wird, müssen die anderen Klassen angepasst werden. Es gibt Tools, die automatisch diese

Klassen anlegen und verwalten[5]

Es gibt 3 verschieden Arten von Beans. Session-Beans und Message-driven-Beans um Verhalten zu beschreiben und Entity Beans um Geschäftsdaten zu modellieren. Hier wird Verhalten von Daten getrennt, also das Gegenteil von verhaltensvollständigen Objekten. Der Grund für diese Aufteilung in verschiedene Beans ist, laut [Rom02], dass Sun nicht alleine an der EJB-Spezifikation arbeitet. Es muss auf einer Vielzahl verschiedener verteilter Systeme laufen. Um diese Flexibilität zu erreichen, gibt es verschiedene Beans. Hier zeigt sich, je allgemeiner ein System sein will, desto mehr Kompromisse muss es eingehen.

Die entscheidenden Unterschiede sind also die verschiedenen Beanarten und die Aufteilung auf mehrere Klassen. Diese Komplexität von EJB erhöht den Lernaufwand für Neueinsteiger. Bei mangelhaftem Verständnis kann es schnell zu falscher Benutzung der Beans kommen. Es gibt aber sehr mächtige Tools, die es einfacher und effizienter machen, mit EJB zu arbeiten.

5.3 Praktische Anwendungsmöglichkeiten

Zur Frage ob NakedObjects skalieren wird, sagen die Autoren, sie wissen es nicht. NakedObjects ist so entworfen, dass es Skalieren sollte, aber nur die Praxis kann beweisen, dass es auch funktioniert. Grosse Systeme mit vielen Nutzern kann man also schon als Einsatzgebiet ausschließen.

Der grösste Nutzen liegt im Moment in der Entwicklung von Prototypen. Durch das Konzept der verhaltensvollständigen Geschäftsobjekte kann ein Entwurf sehr schnell und effizient getestet werden. Es können sogar Codegeneratoren geschrieben werden, die direkt aus einem UML-Model NakedObjects erstellen. In der dann schon laufenden Anwendung ist es unter Umständen leichter zu erkennen, ob die Anwendung wirklich so funktionieren kann.

Ein zukünftiges Anwendungsgebiet sehe ich in der sehr schnellen Entwicklung sehr kleiner Anwendungen. Der im Anhang besprochene ToDo-Planer ist sicherlich noch nicht sehr hilfreich. Aber wenn man bedenkt, dass man etwa 10 Minuten braucht um ihn zu schreiben, könnte man innerhalb einiger Tage eine hilfreiche Anwendung erstellen. Im Anhang stelle ich einige kleine Anwendungen vor, die nützlich sein können. Zur Zeit ist die Visualisierung noch nicht gut genug, dass man wirklich produktiv mit NakedObjects arbeiten kann. Genau diese Schwäche soll mit dieser Arbeit verbessert werden.

Da NakedObjects ein Musterbeispiel für objektorientiertes Arbeiten ist, kann es auch in der Lehre eingesetzt werden. Eine NakedObjects-Anwendung funktioniert nur, wenn man das Problem sinnvoll in Objekte einteilt und die richtigen Aktionsmethoden (Usescases) und Relationen anlegt. Man kann in-

[5] Das ist ein Beispiel dafür, dass gute Tools Schwächen im Framework verstecken können, wie es in Kapitel 5.2.2 angesprochen wurde.

nerhalb weniger Minuten anhand einer lauffähigen NakedObjects-Anwendung testen, ob ein Entwurf funtkionieren kann.

5.4 Zusammenfassung

Tabelle 5.4 zeigt die Vor- und Nachteile des Frameworks in Kurzform.

Vorteile	**Nachteile**
wenig code	keine individuelle Visualisierung
zwangsweise objektorientiertes Arbeiten	der nutzer muss wissen was er tut, er kann nicht eingeschränkt werden fehleranfällig gegen unwissende Nutzer
verhaltensvollständige Geschäftsobjekte	keine professionelle Implementierung vorhanden, opensource Projekt -> kein richtiger Suport, keine Garantie, dass es weiterentwickelt wird
automatische GUI, Persistenz	

Tabelle 5.4: Vor- und Nachteile des NakedObjects-Frameworks

6 Konzept für eine 3D-NakedObjects Oberfläche

Mit dem Wissen der bisherigen Kapitel soll nun die NakedObjects-Oberfläche untersucht werden und nach Verbesserungen unter Verwendung dreidimensionaler Visualisierung gesucht werden. Zuerst werden Schwächen in der bestehenden Visualisierung dargestellt. Danach werden bestehende Anwendungen mit 3D-Visualisierungen vorgestellt. Das dritte Unterkapitel diskutiert Ideen und Konzepte für eine neue NakedObjects-Oberfläche.

6.1 Schwächen der bisherigen NakedObjects-Bedienung

Zu Beginn der Arbeit lag nur die Version 1.2 des NakedObjects-Frameworks vor. Die Visualisierung war besonders für neue Nutzer schwierig. Bei den meisten Desktopsystemen kann man z.B. ein Objekt oder Fenster mit der linken Maustaste bewegen. Bei Version 1.2 endet das meist in einem Bild, ähnlich Abbildung 6.1. Die vielen kleinen Icons lassen sich nicht wieder entfernen. Am Ende bleibt der Eindruck, dass man mit jedem Mausklick etwas „kaputmachen“ kann. Wie im Kapitel 3.2.2 beschrieben, ist es besonders wichtig, dass jede Aktion rückgängig gemacht werden kann. Bei dieser Oberfläche muss man jeden Mausklick mit Bedacht ausführen, da man ihn vielleicht nicht rückgängig machen kann. Das führt zu einem schlechten Ersteindruck. Inzwischen ist Version 2.0 erschienen. Darin sind einige Verbesserungen bei der Visualisierung enthalten. Man kann z.B. Objekte mit der linken Maustaste bewegen, wie man es von den Desktopsystemen her kennt. Im folgenden soll deshalb die Version 2.0 genauer untersucht werden. Die NakedObjects-Beispielanwendung ist beim Framework dabei, und stellt eine Autovermietung dar.

NakedObjects wird nur mit der Maus bedient. In Kapitel 3.2 steht, dass eine Kombination aus Maus- und Tastatureingaben für Vielnutzer sinnvoll ist. Da NakedObjects aber noch nicht in der praktischen Nutzung ist, kann auf Tastatureingaben verzichtet werden. Tabelle 6.1 zeigt, welche Mausklicks welche Wirkung haben. Die Zeilen geben die Art der Mauseingabe an. Die Spalten geben an, worauf die Maus zeigt. Einige Verbesserungsmöglichkeiten werden bereits hier deutlich. Beim Ziehen der Maus über den Arbeitsplatz erscheint auf einmal das Icon des ApplicationContext (Vorher ist es nicht zu sehen). Bei einem Doppelklick in ein Fenster, öffnet sich das gleiche Fenster noch einmal. Bei einem Doppelklick auf

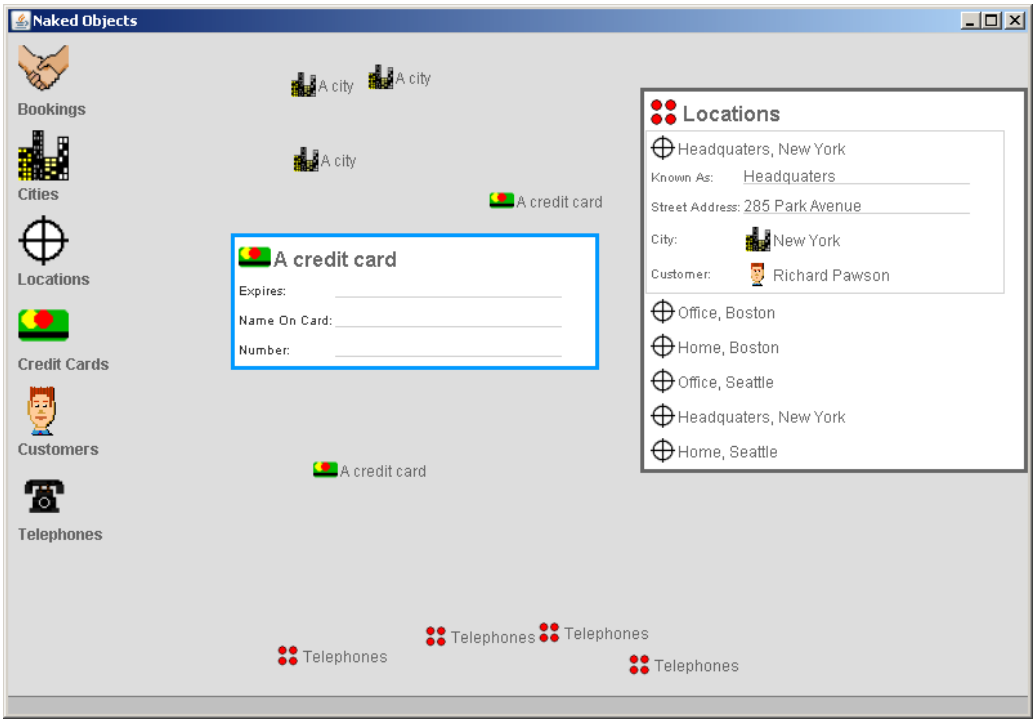

Abbildung 6.1: NakedObjects Version 1.2: Unbedachtes Klicken mit der Maus, führt schnell zu so einem Bild. Die vielen kleinen Icons lassen sich nicht wieder beseitigen.

eine Klasse erwartet man eine Reaktion, es passiert aber nichts. Sinnvoll wäre es z.B. hier die Instanzen der Klasse anzuzeigen. Einige weitere Probleme treten beim Benutzen auf, und werden in den folgenden Absätzen erläutert.

Objekte können in der Icon-Ansicht oder als Fenster angezeigt werden. Das gleiche gilt auch für die Klassen. Allerdings hat die Anzeige einer Klasse als Fenster keinen Nutzen. Abbildung 6.2 zeigt eine geöffnete Klasse. Hier ist auch zu sehen, dass man die Icon-Ansicht einer Klasse auch schließen kann. Einmal geschlossen, gibt es keine Möglichkeit, die Klasse wieder anzuzeigen. Das Popup-Menu einer Instanz der Klasse enthält zwar einen Eintrag `Class`, allerdings passiert nichts, wenn man ihn anklickt. Das Popup-Menu der Arbeitsfläche enthält einen Eintrag `Close all`. Dabei werden zwar alle Fenster geschlossen, die Icon-Ansichten von Instanzen auf der Arbeitsfläche bleiben aber erhalten. Um eine Icon-Ansicht zu schließen, muss man auf einen kleinen grauen Bereich am Rand des Icons rechts-klicken und `Close` auswählen. Ein Rechtsklick auf den Text des Icons zeigt ein anderes Popup-Menu ohne Close-Eintrag. Wenn man hier `Destroy Object` auswählt, verschwindet die Icon-Ansicht auch, die Instanz ist aber auch unwiederruflich gelöscht. Keine Abfrage schützt den Nutzer vor dieser Fehlbedienung.

Eine Neuerung bei Version 2.0 ist, dass es mehrere Möglichkeiten gibt, ein Objekt zu öffnen. Abbildung 6.3 zeigt die 3 verschiedenen Ansichten. Verwirrend ist, dass die Ansicht `Data Form` ein nicht

	Klasse	**Arbeitsfläche**	**Objekt als Icon**	**Object geöffnet**
links	nichts	nichts	nichts	selektieren von Feldern; minimieren, maximieren und schließen des Fensters
doppelt links	nichts	nichts	öffnen	minimieren, maximieren; bei Doppelklick in leeren Teil des Fensters, öffnet ein zweites Fensters des gleichen Objektes
ziehen links oder rechts	bewegen	Icon des ApplicationContext bewegt sich, hat keinen weiteren Effekt	bewegen, wenn auf auf geeignetem Feld in geöffnetem Fenster losgelassen, wird Verknüpfung erstellt	bewegen wenn auf Rand des Fensters
rechts	Popup-Menu: Instances, New Instance, New Transient Instance	Popup-Menu: Quit, About, Close all, Tidy up	Popup-Menu: Actions, Destroy Object, Class, Clone	Popup-Menu: Open as ..., View as ..., Print, Close, Iconize
mitte	zeigt eine Readonly-Variante des Objektes, führt teilweise zu NullPointerExceptons, scheinbar eher für Debugzwecke gedacht			nichts

Tabelle 6.1: Mögliche Mausklicks und ihre Wirkungen. Die Zeilen geben die Art des Mausklicks an, die Spalten, worauf er ausgeführt wurde.

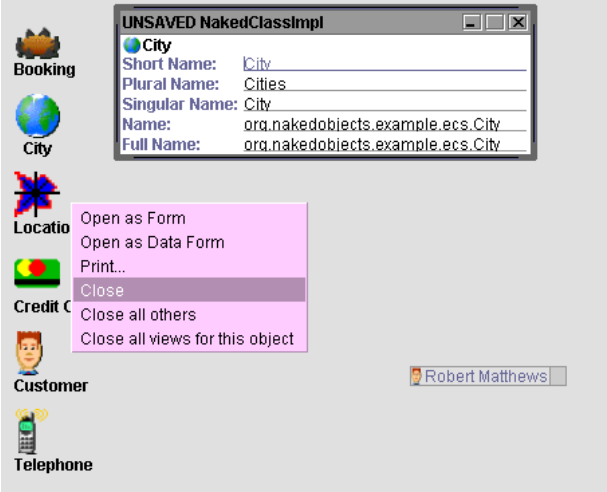

Abbildung 6.2: Klassen lassen sich genauso wie Instanzen öffnen und schließen. Rechts unten sieht man die Icon-Ansicht einer Instanz.

bewegbares Fenster ohne Close-Button ist. Dieses Fenster lässt sich nur über den Close-Eintrag im Popup-Menu schließen. Im `Data Form` Fenster kann man die Felder nicht bearbeiten, der Editor für ein Farbfeld funktioniert aber trotzdem, was wahrscheinlich ein Fehler ist. Es stellt sich die Frage, ob mehrere Ansichten gleichzeitig sinnvoll sind. Auf jeden Fall ist es unnötig, ein und die selbe Ansicht eines Objektes mehrfach zu Zeigen, was z.B. durch einen Doppelklick in einen leeren Teil eines Fensters geschieht.

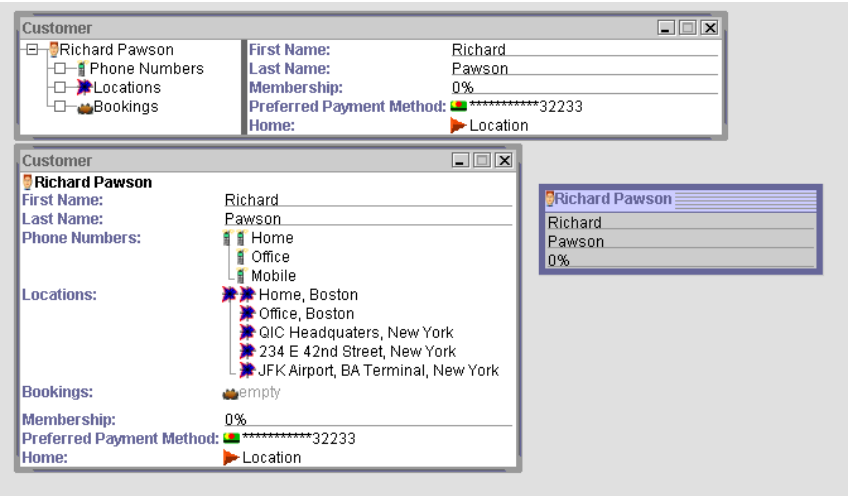

Abbildung 6.3: Verschiedene Ansichten eines Objektes. Oben also `Tree Browser`, unten links als `Form` und unten rechts als `Data Form`.

Bis hierhin hat sich gezeigt, dass neue Nutzer schlecht vor Fehlbedienungen geschützt werden. Es gibt aber auch einige technische Fehler. Öffnen von leeren Verknüpfungen, Clonen mancher Objekte oder

Ziehen von leeren Objekten führen zu Exceptions oder sogar zum Absturz der Anwendung. Wer die Anwendung gut kennt, kann durchaus mit ihr arbeiten. Leider passt das nicht zu den praktischen Einsatzmöglichkeiten des Frameworks. Die grösste Stärke des NakedObjects-Frameworks ist die schnelle Entwicklung von Prototypen, um im direkten Kundendialog ein tragfähiges Objektmodell zu entwickeln. Die Oberfläche hat leider nicht die Robustheit und Eleganz die nötig ist, um Kunde auf Anhieb zu beeindrucken.

Folgende Schwächen wurden bei der NakedObjects-Visualisierung gefunden:

- Fehlbedienungen möglich (Klassen schließen, Icon-Ansichten löschen,...)
- Popup-Menueinträge die nicht oder nicht wie erwartet funktionieren (Class, Close all,...)
- Exceptions, Abstürze
- Verwirrung durch mehrere Ansichten, viele Popup-Menueinträge

Die Oberfläche macht den Eindruck, als ob sie von Entwicklern für Entwickler gemacht ist. Das ist für die praktische Verwendung des Frameworks zur Präsentation von Prototypen ungünstig. Eine robuste, einfache und fehlerresistente Oberfläche ist notwendig. Sie sollte optisch ansprechend sein, um auch Nicht-Techniker anzusprechen.

6.2 Bestehende 3D-Oberflächen

Das Thema dieser Arbeit lautet „3D-Interface für das Naked-Objects-Framework“. Es wird Zeit, etwas genauer über den Begriff 3D-Interface nachzudenken, da er leicht zum Schlagwort wird. Wenn man die verbreiteten Fenster-Desktopsysteme (WIMP-Interface, siehe Kapitel 3.2.3) betrachtet, enthalten sie bereits einige 3D-Elemente. Fenster können sich überlappen und Buttons haben ein 3D-Aussehen. Ursprünglich war die Computerhardware der Grund für viele Kompromisse und Beschränkungen bei den Desktopsystemen. Inzwischen hätte sich das WIMP-Konzept deutlich weiterentwickeln können. Aber es ist weit verbreitet und akzeptiert, wodurch Neuerungen schwer durchsetzbar sind. [wim] gibt einen Überblick zu möglichen Verbesserungen des WIMP-Konzeptes. Auf der gleichen Internetseite gibt es auch einen Artikel, der die Nützlichkeit von 3D-Oberflächen außerhalb von Computerspielen stark in Frage stellt [Thr]. Dort werden Argumente gegen 3D-Oberflächen genannt. Zum einen sind unsere verbreiteten Ein- und Ausgabegeräte zweidimensional, zum anderen funktioniert unser Sehen auch nur über eine Projektion auf die Netzhaut. Diese 2D-Projektion wird erst in unserer Vorstellung wieder zu einem 3D-Bild. Die Frage ob, 3D-Oberflächen wirklich nützlich und erstrebenswert sind, ist also durchaus berechtigt.

Viele 3D-Oberflächen nutzen 3D-Grafik nur, um alte 2D-Funktionalität darzustellen. Ein 3D-Button z.B. hat keinen Vorteil gegenüber einem 2D-Button. Die 3D-Grafik wird also oft nur für optische Zwecke

genutzt. Es gibt aber durchaus auch echte 3D-Anwendungen. Da Ein- und Ausgabegeräte aber nur 2D sind, darf die Anwendung nicht zuviele Freiheitsgrade bieten, sondern muss bei der Positionierung und Orientierung helfen. Das führt zu dem Begriff des zweieinhalb-dimensionalen Nutzererlebnis oder auch „2D+1D“. Gemeint ist eine Anwendung mit 3D-Grafik, die dem Betrachter ein räumliches Bild wahrnehmen lässt, ihm aber nur die nötigen Freiheitsgrade bei Bewegung und Positionierung lässt, damit er die Übersicht nicht verliert. Das bedeutet z.B. dass man den Nutzer seine Entfernung und Position zum Objekt frei wählen lässt, nicht aber seine Blickrichtung. Ein typisches Beispiel sind Karten-Anwendungen wie Google-Earth oder Map24. Eine freie Blickrichtung würde schnell dazu führen, dass der Anwender die Karte aus dem Blick verliert.

Wenn man nach Standarts für 3D-Oberflächen sucht, findet man z.B. VRML [vrm] und viele akademische Arbeiten, die wissenschaftlich die Möglichkeiten von 3D-Grafik untersuchen (z.B. hier [ver]). Leider finden die Ergebnisse dieser Arbeiten wenig Anwendung in der Praxis. Dabei wird 3D-Grafik schon seit Jahren erfolgreich in Computerspielen eingesetzt. [quaa] vergleicht die 3D-Engine von Quake mit VRML und stellt fest, dass die Quake-Engine wenige Dinge gut anstatt viele Dinge schlecht (VRML) löst. Eventuell wird Windows-Vista mit XAML (siehe 4.2.2) zu einem vielgenutzten Standard für 3D-Anwendungen führen.

Aus den bisher genannten Quellen und [SP05] lassen sich einige Richtlinien für 3D-Oberflächen zusammenfassen. Die Nutzer- und Objektbewegung muss einfach sein, damit sich der Anwender zurechtfindet. Das bedeutet z.B., dass der Blick des Nutzer fest auf ein Objekt gerichtet ist, oder Bewegungen nur entlang von Flächen oder Linien zugelassen werden. Weiterhin dürfen optische 3D-Effekte wie Licht, Schatten und Reflektion nicht von der eigentlich Anwendung ablenken. Objekte sollten möglichst räumlich geordnet angezeigt werden, um visuelles Suchen zu erleichtern. Die Anzahl der Objekte kann durch Filter beschränkt werden. Objekte müssen einfach und schnell in den Vordergrund gebracht werden können (semantisches Zoomen) um z.B. mehr Details anzuzeigen oder sie zu bearbeiten. Damit der Nutzer die Orientierung in der Anwendung nicht verliert, muss bei jedem Objekt erkennbar sein, wo es herkommt und wo es hingeht. Dort sollte es dann auch wieder zu finden sein. Übersichtsansichten sollten möglich sein und Transparenz bei großen oder sehr nahen Objekten, damit sie nicht alles verdecken. Die folgende Liste gibt eine Übersicht in Kurzform zu den genannten Richtlinien:

- Bewegung: einfache Nutzer-, Objektbewegung; keine unnötigen Freiheitsgrade; schnellen Positionwechsel ermöglichen
- Optik: 3D-Effekte vorsichtig benutzen, Text lesbar halten
- Organisation: Objekte geordnet anzeigen für visuelles Suchen; visuelle Gruppierungen (Ortsgedächtnis nutzen); semantisches Zoomen und Filtern

- Orientierung: erkennbar halten, wo Objekte herkommen, wo sie hingehen; Überblicksansicht anbieten; Röntgenblick / transparente Objekte

Die folgenden Absätze stellen einige bestehende 3D-Oberflächen genauer vor. Eine umfangreiche Übersicht mit Anwendungen, die dreidimensionale Visualisierung nutzen, gibt [lg3b].

6.2.1 Computerspiele

Computerspielen ist es zu verdanken, dass jeder Heimrechner heutzutage eine 3D-Grafikkarte hat. Mussten die ersten 3D-Spiele noch komplizierte Softwarealgorithmen nutzen, können sie heute auf umfangreiche Bibliotheken wie OpenGL oder Direct3D zurückgreifen. Obwohl moderne Spiele komplexe verteilte Systeme sind und sehr viel Geld umsetzen, finden sie in der Wirtschaftswelt wenig Beachtung und gelten dort als etwas unseriös. Wenn man bedenkt, dass sie seit Jahren erfolgreich 3D-Grafik einsetzen und meist ohne Anleitung erlernbar und benutzbar sind, ist klar, dass man viel von ihnen lernen kann.

Neue Nutzer werden in modernen Spielen über Tutorials eingeführt. Sie erhalten am Anfang einfache Aufgaben, einen verminderten Aktionsumfang und werden in kleinen Schritten durch die Spielwelt geführt. Es werden schrittweise Aktionen hinzugefügt und die Schwierigkeit der Aufgaben erhöht. Vergleichbar in Geschäftsanwendungen sind die Wizards (siehe Kapitel 3.1.2). Sie bieten eine aufgabenbasierte lineare Oberfläche mit vermindertem Funktionsumfang an. Sie tragen aber nicht dazu bei, dass der Nutzer lernt, mit der Anwendung komplexere Aufgaben zu lössen. Weitere Beispiele sind die sprechende Büroklammer bei Microsoft-Word, Tips die beim Starten der Anwendung angezeigt werden, und selten findet man auch Tutorials mit kleinen Beispielaufgaben. Geschäftsanwendungen haben um ein vielfaches mehr mögliche Aktionen als Spiele. Deshalb ist es schwer, schrittweise Aktionen einzuführen, aber kleine Tutorial-Aufgaben, durch die man geführt wird, sollten mehr Verbreitung finden.

Computerspiele sind sehr gute Beispiele für Direct-Manipulation (siehe Kapitel 3.2.2). Es werden kaum Fehlermeldungen benötigt und die meisten Aktionen sind intuitiv umkehrbar. Wenn man z.B. ein Auto zu weit nach links steuert, muss man einfach in die Gegenrichtung lenken um wieder zurückzugelangen. Die einfache Bediengung und der Spaß am Spielen sind Dinge, die man gerne auf Geschäftsanwendungen übetragen möchte. Allerdings gibt es auch entscheidente Unterschiede. Spiele sollen den Anwender herausfordern und enthalten z.B. Zufallsereignisse, auf die schnell reagiert werden muss. Geschäftsanwendungen sollten sich immer vorhersehbar verhalten und dem Anwender die volle Kontrolle geben. Ein Spiel mit diesen Eigenschaften hätte wenig Erfolg. Eine Geschäftsanwendung kann aber auch durchaus „Spaß machen“, indem der Anwender auf kreative Art seine Probleme löst und sie optisch ansprechend ist [Sch04]. Abschließend bleibt zu sagen, dass noch viele Studien nötig sind (wie z.B. [Lew]), um zu

untersuchen, wie man den Erfolg von Computerspielen in Geschäftsanwendungen nutzen kann.

6.2.2 Das Croquet-Projekt

Croquet ist ein OpenSource-Projekt mit sehr hoch gesteckten Zielen [cro]. Es soll eine Plattform entstehen, bei der viele Nutzer gemeinsam in einer virtuellen Welt arbeiten können. Nutzer sollen zusammen auf den gleichen Daten arbeiten können und die gleiche 3D-Welt sehen. Verschiedene 3D-Welten sind durch Fenster verbunden. Diese Fenster sind vergleichbar mit Hyperlinks im Internet. Croquet ist also eine Art 3D-Internet. Ob das Projekt seine Ziele erreichen kann, ist fraglich. [dis] ist eine sehr harte Kritik an dem Projekt, mit der Kernausage, dass es eigentlich nichts leistet. Auf der Croquet-Internetseite kann man eine Beta-Version herunterladen. Es bleibt abzuwarten, was zukünftige Versionen des Projektes leisten werden.

6.2.3 Data-Mountain

Das Data-Mountain-Projekt versucht das Ortsgedächtnis für die Organisation von Dokumenten zu benutzen [dat]. Dazu wird eine schräge Ebene angezeigt, auf der die Objekte bewegt werden können (Abbildung 6.4 rechts). Die Ebene ist mit einer Textur versehen, die das Gruppieren der Objekte einfacher macht. Per Mausklick kann man Objekte in den Vordergrund bringen und wieder zurück. Um die Objekte anzuordnen, hält man die Maustaste gedrückt und zieht es an die gewünschte Stelle. Andere Objekte werden von dem Gezogenen abgestoßen. So wird vermieden, dass sie sich zu nahe kommen. Unterstützt wird das Nutzererlebnis durch 3D-Sound. Das System eignet sich für bis zu 100 Objekte, die sich ohne Untergruppen in mehrere Gruppen aufteilen lassen sollten. Auf der Internetseite des Projektes wird auch eine Studie vorgestellt, bei der das System für die Favoritenverwaltung des Internetbrowsers verwendet. Die Studie bescheinigt der Data-Mountain-Favoritenverwaltung einen Vorteil gegenüber der normalen Favoritenverwaltung des Internet-Explorers.

6.2.4 Die Task-Gallery

Das Task-Gallery-Projekt [tas] stellt sich dem Problem, dass viele Fenster, die zu verschiedenen Aufgaben gehören, nur schwer verwaltet werden können. Es nutzt die Metapher einer Galerie. Der Nutzer befindet sich in einem langen Gang, in dem er vor- und zurücklaufen kann. Die Bilder an den Wänden der Galerie sind jeweils eine Arbeitsfläche mit beliebig vielen Anwendungen, die zu einer Aufgabe gehören. Die aktive Arbeitsfläche befindet sich an der Wand, die dem Anwender frontal gegenüber steht (siehe Abbildung 6.4 links). Die Fenster lassen sich als Thumbnail-Ansicht in einer Reihe anordnen und

auswählen. Diese Funkionalität ist der Flip3D-Funktion im neuen Windows-Vista sehr ähnlich (siehe Kapitel 4.2.3). In der linken Hand hält der Nutzer ein Tablett, dass mit der eben beschrieben Data-Mountain-Technik arbeitet. Auf dem Tablett befinden sich oft genutzte Anwendungen und Dokumente. Gesteuert wird die Bewegung in der Galerie über ein einfaches Menu, das erscheint, wenn man mit der Maus auf den grünen Mann rechts unten zeigt oder über die Cursor-Tasten.

Abbildung 6.4: Links: Die Task-Gallery. Rechts: Data-Mountain

6.3 Diskussion von Ideen und Konzepten

Für eine NakedObjects-Oberfläche stellen sich sehr konkrete Fragen:

- wie werden Klassen und Objekte dargestellt (6.3.1 und 6.3.3)
- wie werden Verknüpfungen zwischen Objekten erstellt und dargestellt (6.3.4)
- wie werden Objekte erzeugt (6.3.3)
- wie werden Objekte gefunden (6.3.3)
- wie soll die Arbeitsfläche aussehen (6.3.4)

Die folgenden Unterkapitel behandeln einzelne Themen der Oberfläche. Es werden verschiedene Möglichkeiten angesprochen und jeweils eine für den Protoypen ausgewählt. Dazu werden zunächst Entwurfsskizzen verwendet. Im Kapitel 6.4 werden Screenshots der ersten Vorabversion gezeigt. Auf der CD, die zur Arbeit gehört, befindet sich eine lauffähige Version des Protypen.

6.3.1 Icons

Ein Icon ist meist eine kleine Repräsentation einer Aktion oder eines Objektes. Bei den klassichen WIMP-Interfaces sind das kleine Bilder. Es liegt nahe, bei einer 3D-Anwendung kleine 3D-Objekte

zu verwenden. Oft wird angenommen, dass Icons als universelle visuelle Sprache dienen können. Ob ein Anwender die Bedeutung eines Icons aber sofort erkennt, hängt von seiner Vorbildung ab. Ein Beispiel sind Verkehrschilder. Sie sind leicht zu verstehen und zu erkennen. Ihre Bedeutung muss aber trotzdem erst gelernt werden. Das Ziel der sofort erkennbaren Icons ist also nicht erreichbar. Wichtiger ist die Wiedererkennbarkeit. Durch das gute Bildgedächtnis des Menschen werden einmal gelernte Icons nicht so leicht vergessen.

Genaugenommen sollte ein Icon eine kleine Visualisierung des Objektes sein. Also z.B. ein 3D-Model des Kunden für ein Kundenobjekt. Technisch ist das noch nicht realisierbar. Deshalb kombiniert man für gewöhnlich Text und Icon. Bei einer Datei wird z.B. der Typ über ein Icon und der Name über einen Text angezeigt. Bei einem NakedObject zeigt man den Titel und das Icon der Klasse an.

Wie bereits erwähnt, muss ein Icon vor allem erkennbar sein. Es sollte sich von anderen Icons deutlich unterscheiden und trotzdem gut als Icon zu erkennen sein. Es muss sich vom Hintergrund abheben und Selektion muss deutlich erkennbar sein. Icons müssen also innerhalb einer Anwendung alle dem gleichen Stil folgen.

Für diese Arbeit werden alle Icons aus geometrichen Primitiven aufgebaut. Abbildung 6.6 zeigt einige Beispiele. Die Icons bestehen nur aus Kisten, Zylindern und Kugeln. Sie werden nicht aus Dateien geladen, sondern mit wenigen Zeilen Java-Code erzeugt. Das passt sehr gut zum NakedObjects-Framework. NakedObjects versucht, soweit es geht, ohne externe Resourcen auszukommen. Die bei Java-Anwendungen häufig bestehenden Pfadprobleme beim Laden von Resourcen fallen damit weg. Das einfache Aussehen der Icons passt zum minimalistischen Vorgehen von NakedObjects. Abbildung 6.5 zeigt den Java-Code für eine Lupe. Die Lupe ist in Abbildung 6.6 an achter Stelle von links zu sehen. Die folgende Liste zeigt noch einmal die Vor- und Nachteile von Icons aus geometrischen Primitiven.

Vorteile:

- einheitlicher Stil erzwungen
- deutlich als Icon erkennbar
- Wiedererkennbarkeit gut
- kein Laden aus Datei notwendig

Nachteile:

- manchmal sehr abstrakt, schlechte Ersterkennbarkeit
- Grafikfehler bei Darstellung durch zu geringe Auflösung des Z-Buffers

```
public class IconMagnifyingGlass extends Component3D {
    public IconMagnifyingGlass() {
        IconBuilder builder;
        builder = new IconBuilder( this );
        builder.setColor( 250, 156, 20 );
        builder.addCylinder( 5, 5, 50, 39, 39, 50, 8, 4 );
        builder.addCylinder( 60, 60, 48, 60, 60, 52, 30, 30 );
        builder.setColor( 150, 150, 200 );
        builder.addSphere( 60, 60, 50, 25, 25, 10 );
    }
}
```

Abbildung 6.5: Eine Icon-Klasse. Dargestellt wird eine Lupe (8. Icon v. links in Abbildung 6.6).

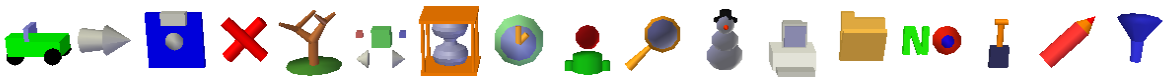

Abbildung 6.6: 3D-Icons aus geometrichen Primitiven

6.3.2 Selektion

Das Ziel von Selektion ist es Objekte sichtbar zu kenzeichnen. Normalerweise kann man auf diesen selektierten Objekten Aktionen ausführen. Kapitel 3.1.1 hat erklärt, dass es viele Wege gibt, die Aufmerksamkeit auf ein Objekt zu lenken. Am verbreitesten bei den WIMP-Systemen ist es, den Hintergrund des Objektes einzufärben oder es zu umranden. Bei 3D-Anwendungen liegt es nahe, Selektion durch Skalierung anzuzeigen. Effekte wie Beleuchtung oder Animation sind möglich aber technisch aufwändig und können ablenken.

6.3.3 Toolbars und Menus

Fast jede Anwendung benutzt Toolbars und Menus um Aktionen auszuwählen. Menus treten meist in Form von Listen von Kommandos auf. Komplexe hierarchische Menus sind für neue Nutzer und Gelegenheitsnutzer schwer zu durchschauen. Besser ist es, die Menus aufzuteilen und sie dort anzuzeigen, wo die Kommandos wirken sollen. Das kann z.B. durch Pulldown-Menus erreicht werden. Idealerweise kommt eine Anwendung fast ohne globale Menus aus. Alle Funktionalität lässt sich auch auf Pulldown-Menus und Toolbars verteilen.

Ein Toolbar ist eine Ansammlung von anklickbaren Buttons und Icons. Meist sind Toolbars rechteckige Bereiche, die am Rand eines Fensters angezeigt werden. In der Realität entspricht eine Steuerkonsole mit Schaltern am ehesten dem Konzept des Toolbars. Die Metapher der Steuerkonsole, vor der der Benutzer

steht, soll für den Prototypen des NakedObjects-Oberfläche verwendet werden. Abbildung 6.7-Links zeigt eine Skizze von diesem Konzept. Auf dem Bild ist auch die Arbeitsfläche und der Aufgabentisch abgebildet, die im nächsten Unterkapitel besprochen werden.

Abbildung 6.7-Mitte zeigt eine Entwurfsskizze für die Steuerkonsole. Die NakedObject-Klassen werden in einer Reihe als 3D-Icon angezeigt. Falls es zu viele sind, kann man mit den Pfeilen rechts und links durchwechseln. Bei sehr vielen Klassen könnte es z.B. einen extra Auswahlmodus geben, bei dem die Klassen-Icons stark verkleinert angezeigt werden, damit alle sichtbar sind. Um die Objekte einer Klasse anzuzeigen, reicht es, das Klassen-Icon anzuklicken. Die Liste mit den Objekten kommt von unten aus der Konsole. Aus dieser Liste können Instanzen auf die Arbeitsfläche gezogen werden. Die Liste enthält einen Eintrag mit dem Titel `Neu` am Anfang. Um ein neues Objekt zu erzeugen, zieht man dieses auf die Arbeitsfläche. Es fehlen noch Möglichkeiten zum Suchen und zum Löschen. Ausserdem sollte es einen Button zum Schließen der Anwendung geben. Eine zweite Reihe mit Icons könnte diese Aufgabe übernehmen. Ein Filter zum Suchen, ein Papierkorb zum Löschen und ein X zum beenden. Das X könnte der Metapher nach auch eine Art Stromstecker sein, den man zum Beenden herauszieht. Ob dieser Zusammenhang vom Anwender verstanden wird, muss getestet werden. Um eine Liste mit Objekten zu filtern, zieht man den Trichter auf die Klasse. Um die gefiltertete Liste zu löschen zieht man den Papierkorb auf die Klasse, oder um ein einzelnes Objekte zu löschen, das Objekt auf den Papierkorb.

6.3.4 Die Arbeitsfläche

Die Desktop-Metapher orientiert sich an einem Schreibtisch. Mit 3D-Grafik ist es möglich, eine schräge Ebene darzustellen, wie man sie bei einen Schreibtisch vor sich hat. Allerdings ist der Monitor selber eine 2D-Arbeitsfläche. Wieso soll man diese nicht direkt nutzen und erst etwas anderes darstellen? Die Data-Mountain-Technik, welche im vorhergehenden Kapitel (6.2) vorgestellt wurde, zeigt, dass eine schräge Ebene gut ist, um Objekte abzulegen und wiederzufinden, indem das Ortsgedächtnis des Anwenders genutzt wird. Die eigentliche Arbeitsfläche zum Anzeigen und Bearbeiten von Daten sollte nicht räumlich verzerrt sein.

Die Data-Mountain-Technik eignet sich gut, um mehrere Arbeitsflächen abzulegen. Zu jeder Aufgabe wird eine neue Arbeitsfläche angelegt, in der die entsprechenden Objekte geöffnet und bearbeitet werden. Zu jeder Zeit kann man zwischen den einzelnen Aufgaben-Arbeitsflächen wechseln. Die schräge Ebene mit den Aufgaben-Arbeitsflächen wird im folgenden Aufgabentisch genannt. [dat] hat Tests mit bis zu hundert Objekten auf dem Tisch gemacht. Selten wird man diese Menge an Aufgaben gleichzeitig mit einer Anwendung lösen wollen. Die Übersicht sollte also auch bei viele Aufgaben nicht verloren gehen.

Die Arbeitsfläche sollte einen Titel und einen Knopf zum Schließ haben. Beides sollte am oberen Rand

angezeigt werden, da fast jedes Fenstersystem den Titel und den Schließen-Knopf dort platziert. Auf der Arbeitsfläche gibt es Objekte in der Icon-Ansicht und geöffnete Objekte in der Fenster-Ansicht. Interessant ist die Frage, ob man wirklich beliebig viele Öffnungen von Objekten zulassen muss, oder ob man die Anzahl begrenzen sollte. Ein Schreibtisch, auf dem zuviele Dokumente liegen, ist schlecht benutzbar. Das gleiche gilt für eine Arbeitsfläche, auf der zuviele Objekte geöffnet sind. Auf jeden Fall sollte es einen Stapel für die Icon-Ansichten und einen für die Fenster-Ansichten geben. Die Icon-Ansichten können als Liste untereinander angezeigt werden und die Fenster-Ansichten als Kartenstapel wie bei der Task-Gallery (Kapitel 6.2.4) oder der Flip-3D-Funktionalität von Windows-Vista (Kapitel 4.2.3, Abbildung 4.9). Wenn die Arbeitsfläche geschlossen wird, darf sie nicht einfach verschwinden, sondern wird, genauso wie gelöschte Objekte, in den Papierkorb verschoben. Das kann durch eine Animation verdeutlicht werden.

Die Objekte auf der Arbeitsfläche können sich an drei verschiedenen Stellen befinden. Sie können im Kartenstapel, bei den geöffneten Objekten, in der Liste der Icon-Ansichten und als Fenster, geöffnet zum Bearbeiten und Anzeigen, angeordnet werden. Um Verwirrung zu vermeiden, sollten sie nicht mehrfach existieren. Um zwischen den drei Zuständen zu wechseln, bietet sich ein kleiner Toolbar am oberen Rand des Objektes oder Drag-and-Drop zwischen den Bereichen an. Die Aktionen, die ein Objekt anbietet, können durch ein Pulldown-Menu angezeigt werden. Üblicherweise wird das Menu durch die rechte Maustaste angezeigt.

Eine letzte Frage stellt sich noch. Wie sollen die Relationen zwischen den Objekten angezeigt werden? Bei einer 1:1-Relation, kann die Icon-Ansicht des anderen Objektes in der Fenster-Ansicht angezeigt werden. Enthält das geöffnete Objekt eine 1:N-Relation, könnte man, in einer kompakten Ansicht, mehrere Icon-Ansichten in die dritte Dimension übereinander stapeln. Bei Bedarf wird dieser Stapel gedreht, so dass die Icon-Ansichten als Liste angezeigt werden.

Abbildung 6.7 zeigt rechts die Konzeptzeichung einer Arbeitsfläche.

6.4 Der Prototyp

Der Protoyp versucht die Ideen und Konzepte aus Kapitel 6.3 ein einer Looking-Glass-Anwendung umzusetzen. Besonders wurde darauf geachtet, das Animationssystem von Looking-Glass zu nutzen. Bei jedem Objekt soll erkennbar sein, wo es herkommt und wo es hingeht. Wenn sich ein Objekt z.B. zu seinem Klassen-Icon bewegt, muss es dort auch wieder zu finden sein. Die Transparenz hat in der verwendeten Looking-Glass-Version (0.8.0) noch einige Fehler, soll aber trotzdem verwendet werden. Abbildung 6.8 zeig einen Screenshot des Prototypen mit der Steuerkonsole und dem Aufgabentisch. Aktuellere Screens-

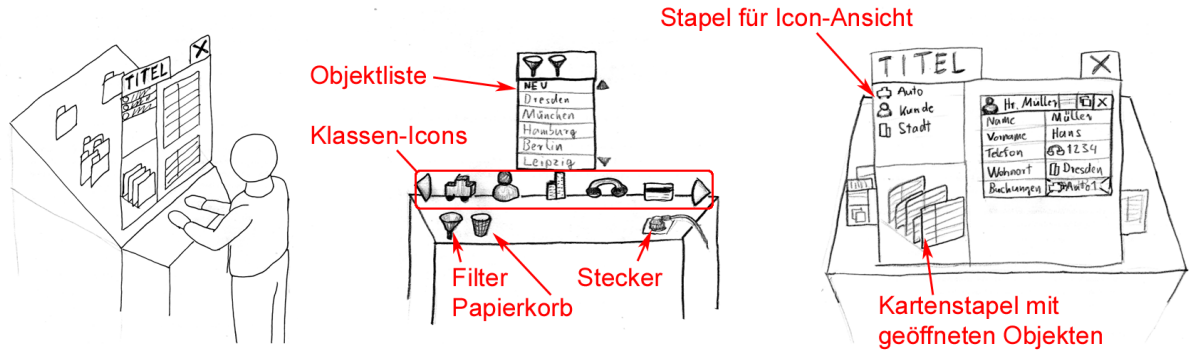

Abbildung 6.7: Konzeptskizzen. Links: Metapher der Steuerkonsole mit Arbeitsfläche und Aufgabentisch. Mitte: Steuerkonsole. Rechts: Arbeitsfläche.

hots, Dokumentation, ein Demonstrationsvideo und die lauffähige Version des Prototypen befinden sich auf der zur Arbeit gehörenden CD.

Tabelle 6.2 zeigt die möglichen Mausklicks und ihre Wirkung im Prototypen. Da der Prototyp noch weiterentwickelt wird, sind Änderungen in der Belegung möglich.

6.5 Zusammenfassung

Die bisherige NakedObjects-Oberfläche weist einige Schwächen auf. Sie schützt den Anwender schlecht vor Fehlbediengungen und kann am Anfang etwas verwirren. Die Popup-Menus sind unübersichtlich und verhalten sich teilweise unerwartet. Hinzu kommen einige Bugs, die Exceptions erzeugen oder die Anwendung abstürzen lassen. Das Konzept für die NakedObjects-Oberfläche aus dieser Arbeit versucht diese Schwächen zu beseitigen. Da sie aber dreidimensionale Visualisierung verwendet, ist sie selber sehr experimentell. Weitere Studien sind nötig, um einen Vorteil der 3D-Oberfläche zu belegen.

Kapitel 6.2 hat gezeigt, dass man gute und schlechte Beispiele für 3D-Anwendungen finden kann. Eine Studie zeigt, dass die Data-Mountain-Technik bei der Organisation von Objekten einen Vorteil gegenüber herkömmlichen Techniken hat. Das liegt daran, dass das Ortsgedächtnis des Anwenders gut genutzt wird. 3D-Visualisierung kann also Vorteile haben, ist aber immer vorsichtig einzusetzen. Oft wird 3D-Grafik nur für optische Effekte verwendet und bringt keinen funktionellen Vorteil. Die Gefahr besteht, dass zuviele grafische Spielereien den Nutzer abgelenken und verwirren.

In Kapitel 6.3 wird ein Konzept für die 3D-Oberfläche entwickelt. Die Oberfläche soll aus drei Teilen bestehen: aus einer Arbeitsfläche, die sich genau vor dem Anwender befindet, einer Steuerkonsole vor der Arbeitsfläche und einem Aufgabentisch hinter der Arbeitsfläche. Zu jeder Aufgabe, die der Nutzer lösen

	Klasse	**Arbeitsfläche**	**Objekt als Icon**	**Object geöffnet**
links	Objekte anzeigen	nichts	selektieren	selektieren von Feldern; Ansicht ändern
doppelt links	nichts	ablegen auf Aufgabentisch	öffnen	schließen
ziehen links oder rechts	nichts	verkleinerte Arbeitsflächen-Ansicht auf Aufgabentisch bewegen	bewegen, kann an geigneten Stellen losgelassen werden (anderes Objekt zum Verknüpfen, Papierkorb zum Löschen, Klasse zum Schließen)	nichts
rechts	nichts	nichts	Popup-Menu mit Aktionen	
mitte	nichts			

Tabelle 6.2: Mögliche Mausklicks und ihre Wirkungen beim Prototypen. Die Zeilen geben die Art des Mausklicks an, die Spalten, worauf er ausgeführt wird.

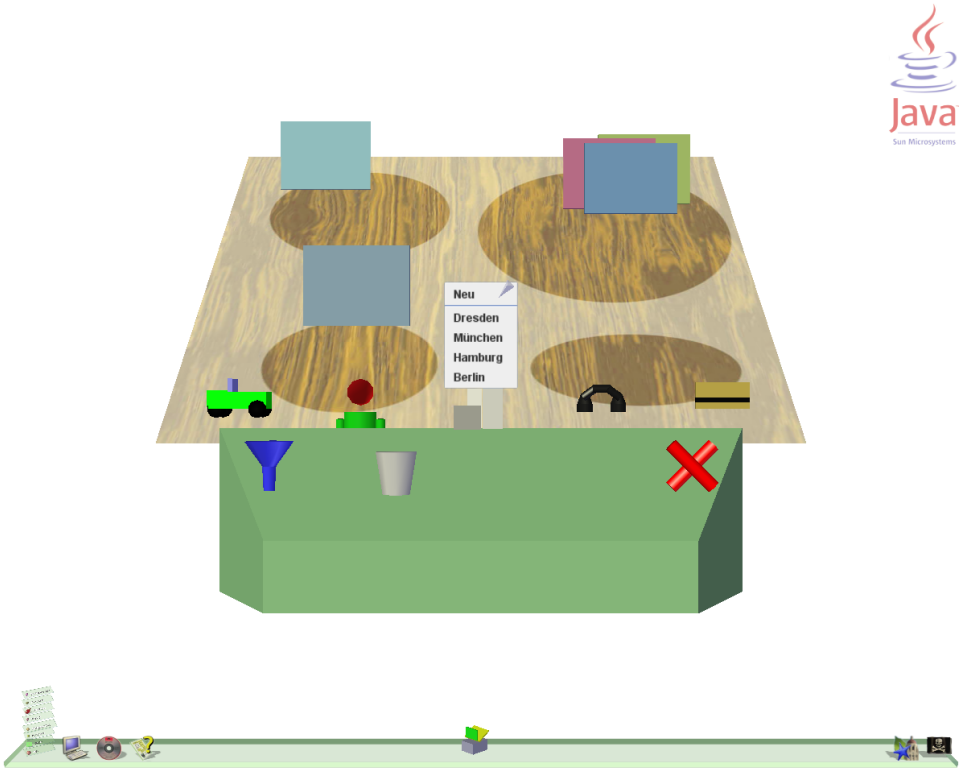

Abbildung 6.8: Ein Screenshot von einer frühen Versin des Prototypen.

will, legt er eine Arbeitsfläche an. Auf diese legt er alle Objekte ab, die er zum Arbeiten benötigt. Die einzelnen Arbeitsflächen werden mit der Data-Mountain-Technik auf dem Aufgabentisch abgelegt. Um Klassen zu erzeugen und zu finden, gibt es eine Steuerkonsole vor der Arbeitsfläche. Die Steuerkonsole zeigt Icon-Ansichten aller Klassen, einen Papierkorb zum Löschen und einen Filter zum Suchen von Objekten. Der Prototyp versucht die Ideen des Konzeptes in einer Looking-Glass-Anwendung umzusetzen.

7 Zusammenfassung

Die Arbeit hat sich zunächst in drei großen Kapiteln mit den bestehenden Grundlagen des Themas beschäftigt. Es wird ein umfrangreicher Einblick in Mensch-Maschine-Schnittstellen, 3D-Desktopsysteme und NakedObjects gegeben. Wissen aus den Bereichen Psychologie, Softwaretechnik und Hardware wird zusammengetragen und bewertet. Dieses Wissen wird mit neuen Ideen zu einem Konzept für eine 3D-Oberfläche des NakedObjects-Frameworks kombiniert.

Kapitel 3 zeigt, dass alte Konzepte immer wieder neu überdacht werden müssen. Neues Wissen aus der Psychologie und neue Hardware führen zu neuen Möglichkeiten. Jedoch werden alte Konzepte weiter verwendet, obwohl Verbesserungen greifbar sind. Wenn ein System einmal verbreitet und akzeptiert ist, setzen sich Neuerungen nur langsam durch. Für Entwickler ist es deshalb wichtig, immer wieder die Grundlagen, auf denen sie ihre Konzepte aufbauen, zu prüfen.

Nachdem es seit Jahren 3D-Grafikhardware in fast jedem Rechner gibt, wird diese jetzt auch außerhalb von Computerspielen und Spezialanwendungen intensiv genutzt. Kapitel 4 stellt drei Desktopsysteme mit neuen Ideen und Technologien vor. Der Grafikprozessor wird von vielen Anwendungen zugleich genutzt und der Grafikspeicher wird virtualisiert. Die nächsten Jahren werden zeigen, ob 3D-Grafik breite Verwendung in der Praxis findet. Die technischen Grundlagen sind bereits vorhanden.

In der Softwaretechnologie setzen sich immer wieder neue Methoden und Technologien durch, um die wachsende Komplexität von Anwendungen zu beherrschen. NakedObjects versucht einen minimalistischen Ansatz, der sich streng an die Grundlagen und Vorteile von objektorientierter Prgrammierung hält. Obwohl seit Jahren fast überall mit Objekten programmiert wird, sind die Grundlagen oft schlecht verstanden. Das NakedObjects-Framework gibt ein Beispiel, wie einfach ein objektorientierter Ansatz für Geschäftsanwendungen sein kann.

Des letzte Kapitel der Arbeit (6) erstellt mit dem erarbeiteten Wissen ein Konzept für eine NakedObjects-Oberfläche. Es zeigt sich, dass 3D-Grafik auch für Geschäftsanwendungen Vorteile haben kann. Das Ortsgedächtnis kann besser genutzt werden und Metaphern und Informationen lassen sich intuitiver darstellen.

Weiterführende Arbeiten sollten sich vor allem mit Nutzertests beschäftigen. Folgende Fragen sind zu

klären:

- Wird die Metapher des Konzeptes verstanden?
- Finden sich Nutzer in der Anwendung zurecht?
- Empfinden sie die dritte Dimension als angenehm oder störend?
- Ist der Ersteindruck gut oder schlecht?
- Werden Vielnutzer gebremst?

Im Laufe dieser Arbeit gab es mehrfach gravierende Änderungen in den Basis-Systemen. Da diese schnelle Entwicklung weitergehen wird, sollte jede folgende Arbeit zunächst prüfen, welche neuen Studien und Projekte es zu dem Thema 3D-Visualisierung gibt.

Literaturverzeichnis

[app87] *Apple human interface guidelines: the Apple desktop interface*. Addison-Wesley, 1987

[Bro95] BROWN, Kyle: Remembrance of Things Past: Layered Architectures for Smalltalk Applications. In: *The Smalltalk Report* 4(9) (1995), April, S. 4–7

[Cha] CHANNEL9. *Interviews mit Microsoftmitarbeitern: Kam Vedbrat - Looking at Windows Vista's user interface (AERO), Daniel Lehenbauer - Demo of Avalon 3D, Pablo Fernicola (and others) - An hour with the Avalon Team*.
http://channel9.msdn.com/

[CLB+] CEAPARU, Irina ; LAZAR, Jonathan ; BESSIERE, Katie ; ROBINSON3, John ; SHNEIDERMAN, Ben. *Determining Causes and Severity of End-User Frustration*.
http://hcil.cs.umd.edu/trs/2002-11/2002-11.html

[cro] *Croquet*.
http://www.opencroquet.org/

[dat] *Data Mountain*.
http://research.microsoft.com/adapt/datamountain/

[dis] *Open Croquet*.
http://c2.com/cgi/wiki?OpenCroquet

[Dmi] DMITRIEV, Sergey. *Language Oriented Programming*.
http://www.onboard.jetbrains.com/is1/articles/04/10/lop/

[Dun] DUNLAP, Nathan. *Designers Love .NET*.
http://www.designerslove.net/

[DW68] DOUGLAS, C. E. ; WILLIAM, K. E.: A research center for augmenting human intellect. In: *AFIPS Conference Proceedings, Fall Joint Computer Conference* (1968), December, S. 395–410

[ein] *Electronic paper display*.
http://www.eink.com/

[fit] *Fittsbits - A game investigating Fitts' Law*.
http://www.rodo.nl/fittsbits/

[FPB03] FREDERICK P. BROOKS, Jr.: *The Mythical Man-Month*. Anniversary Edition. Addison Wesley, 2003

[Gal02] GALITZ, Wilbert O.: *The Essential Guide to User Interface Design*. Wiley, 2002

[geh] *Hirn-Implantat*.
http://www.spiegel.de/wissenschaft/mensch/0,1518,426451,00.html

[gom] *GoMonkey*.
http://www.gomonkey.at/index2.htm

[HT03] HUNT, Andrew ; THOMAS, David: *Der Pragmatische Programmierer*. Hanser, 2003

[kom] *KommDesign.de Informationsarchitektur*.
http://www.kommdesign.de/

[Lew] LEWIS, David. *Games Interfaces: Report on StarCraft*.
http://www.rpi.edu/~gricer/hci_lab/starcraft.html

[lg3a] *Looking Glass Homepage*.
https://lg3d.dev.java.net/

[lg3b] *Project Looking Glass Related Technologies*.
https://lg3d.dev.java.net/lg3d-related-technologies.html

[Mic] MICROSOFT. *Präsentationsfolien WinHEC 2004:Avalon Graphics - 2D, 3D, Imaging and Composition, Avalon Graphics Stack Overview, Graphics on the Windows Desktop, Windows Graphics Foundation, Windows Longhorn Display Driver Model - Details and Requirements*.
http://www.microsoft.com/whdc/winhec/pres04-tech.mspx/

[MS87] MARGONO, Sepeedeh ; SCHNEIDERMAN, Ben: A study of file mainpulation by novices using commands versus direct manipulation. In: *Twenty-sizth Annual Technical Symposium* (1987), Juni, S. 154–159

[Naka] *The naked objects approach.*
http://www.nakedobjects.org/static.php?content=no-approach.html

[Nakb] *The Naked Objects Framework.*
http://sourceforge.net/projects/nakedobjects/

[Nakc] *A short User Guide for Naked Objects applications.*
http://www.nakedobjects.org/static.php?content=user-guide.html#user-guide

[nin] *Nintendo 3D-Controller.*
http://ms.nintendo-europe.com/wii/?site=controller.html&l=deDE

[OHJ+01] OESTEREICH, Bernd ; HRUSCHKA, Dr. P. ; JOSUTTIS, Nicolai ; KOCHER, Dr. H. ; KRASEMANN, Dr. H. ; REINHOLD, Markus: *Erfolgreich mit Objektorientierung.* Oldenbourg, 2001

[Paw04] PAWSON, Richard: *Naked objects.*
http://www.nakedobjects.org/downloads/thesis.pdf, Diss., Juni 2004

[PM02] PAWSON, Richard ; MATHEWS, Robert: *Naked Objects.* Wiley, 2002

[PMH04] PAWSON, Richard ; MATTHEWS, Robert ; HAYWOOD, Dan. *The Naked Objects Architecture Series.*
http://www.theserverside.com/articles/article.tss?l=NakedObjectSeries_1. Januar 2004

[pro] *Projezierte Tastatur.*
http://www.canesta.com/html/celluon.htm

[quaa] *Playfulness in 3-D Spaces.*
http://www.shirky.com/writings/quake.html

[quab] *Quartz.*
http://developer.apple.com/graphicsimaging/quartz/

[quac] *Quartz 2D Programming Guide.*
http://developer.apple.com/documentation/GraphicsImaging/

Conceptual/drawingwithquartz2d/index.html#//apple_ref/doc/uid/TP30001066

[Ras] RASKIN, Jef. *Intuitive equals familiar*. http://www.asktog.com/papers/raskinintuit.html

[rea] *reacTable*. http://www.iua.upf.es/mtg/reacTable/

[Rom02] ROMAN, Ed: *Mastering Enterprise JavaBeans*. Wiley, 2002

[Sch83] SCHNEIDERMAN, Ben: Direct manipulation: a step beyond programming languages. In: *IEEE Computer* 16 (1983), August, S. 57–69

[Sch04] SCHNEIDERMAN, Ben: Designing for Fun: How can we design user interaces to be more fun? In: *interactions* (2004), September, October, S. 48–50

[SP05] SCHNEIDERMAN, Ben ; PLAISANT, Catherine: *Designing the user interface*. Addison Wesley, 2005

[SS97] SCHÖNPFLUG, Wolfgang ; SCHÖNPFLUG, Ute: *Psychologie*. Beltz, Psychologie-Verl.-Union, 1997

[tas] *The TaskGallery*. http://research.microsoft.com/adapt/TaskGallery/

[Thr] *Three Dee Interface*. http://c2.com/cgi/wiki?ThreeDeeInterface

[Thu] THURROTT, Paul. *SuperSite for Windows*. http://www.winsupersite.com/

[ver] *Veröffentlichungen im Bereich Multimediatechnik*. http://www-mmt.inf.tu-dresden.de/Projekte/CONTIGRA/Dokumentation/Publikationen/

[vrm] *VRML Virtual Reality Modeling Language*. http://www.w3.org/MarkUp/VRML/

[Wei05] WEINHOLD, Carsten: Display-Architekturen. 2005. – Forschungsbericht

[wika] *3D-Brillen*. http://de.wikipedia.org/wiki/3D-Brille

[wikb] *Actions per minute*.
http://en.wikipedia.org/wiki/Actions_per_minute

[wikc] *Chord-Tastatur*.
http://en.wikipedia.org/wiki/Chorded_keyboard

[wikd] *Cocktail party effect*.
http://en.wikipedia.org/wiki/Cocktail_party_effect

[wike] *DVORAK Tastaturlayout*.
http://en.wikipedia.org/wiki/Dvorak_Simplified_Keyboard

[wikf] *Fitts Law*.
http://en.wikipedia.org/wiki/Fitts'_law

[wikg] *Gestaltpsychologie*.
http://de.wikipedia.org/wiki/Gestaltpsychologie

[wikh] *Kreis-Menus*.
http://en.wikipedia.org/wiki/Pie_menu

[wiki] *Letterwise*.
http://en.wikipedia.org/wiki/LetterWise

[wikj] *Multitap*.
http://en.wikipedia.org/wiki/Multi-tap

[wikk] *QWERTY Tastaturlayout*.
http://en.wikipedia.org/wiki/QWERTY

[wikl] *Stereogramme*.
http://de.wikipedia.org/wiki/Stereogramm

[wikm] *Wikipedia - Eye tracking*.
http://en.wikipedia.org/wiki/Eye_tracking

[wikn] *Wikipedia - Intuition*.
http://de.wikipedia.org/wiki/Intuition

[wiko] *Wikipedia - Metapher*.
http://de.wikipedia.org/wiki/Metapher

[wim] *WIMP is broken*.
http://c2.com/cgi/wiki?WimpIsBroken

[Wir04] WIRTH, Thomas: *Missing Links*. Hanser, 2004

A Tabellen zu Motivation und Handeln

Die Folgenden beiden Kapitel bieten Zusatzinformation zu Kapitel 3.1. Es handelt sich um Tabellen zu emotionalen Begleiterscheinungen und Basismotivationen. Emotionen und Motivationen sind können nicht eindeutig beschrieben werden. Die Tabellen sind also eher als Richtwerte zu betrachten.

A.1 Emotionale Begleiterscheinungen

Emotionen sind schwer berechenbar, beeinflussen aber sehr die Leistungsfähigkeit des Menschen. Tabelle A.1 stammt aus [kom] und versucht, den Ereignissen beim Handeln und Problemlösen Emotionen zuzuordnen.

Beispiele aus Geschäftsanwendungen die bestimmte Emotionen fördern:

- lange Ladezeiten durch aufwendige Grafik (5. und 8.)
- nicht erkennbare Icons ohne Tooltips (8. und 9.)
- Suchanfragen ergeben brauchbare Ergebnisse (1., 2. und 4.)
- Suchanfragen ergeben hunderte Ergebnisse, zu 90% unbrauchbar (5., 7., 8., 9. und 10.)

A.2 Basismotivationen

Tabelle A.2 und der Text danach stammen aus [kom]. Die meisten Menschen verfügen über diese 16 Basismotivationen, wobei sie sehr unterschiedlich stark ausgeprägt sein können.

Beispiele aus der Computerwelt, die bestimmte Motivationen ansprechen:

- Service-Hotline (7. Hilfe)
- Diskussionsforum (3. Kontakt)
- Virenscanner (5. Sicherheit)
- Computerspiel (1. Neugier, 2. Leistung, 4. Macht, 10.Spiel,...)

Bedingung	**emotionale Konsequenz**
1. Handeln/Problemlösen schreitet fort	Freude, Interesse
2. ein Ziel oder Zwischenziel, wird erreicht, das Problem gelöst	Freude, Stolz, Entspannung, Erleichterung
3. neue Zielzustände werden definiert	Neugier, Interesse
4. wirksame Operatoren werden gefunden	Interesse, Freude
5. trotz Anstrengung rückt die Lösung nicht näher oder entfernt sich	Ärger, Frustration, Angst, Hilflosigkeit
6. völliges Fehlen von Zielen	Langeweile, Orientierungslosigkeit, Depression
7. zu viele oder unpräzise (Unter-) Ziele werden verfolgt	Verwirrung, Frustration, Ärger
8. das Handeln oder der Problemlöseprozeß wird behindert oder stark verzögert	Ärger, Irritation, Ungeduld
9. es gibt keine sinnvollen Operatoren	Ärger, Angst, Resignation
10. ein Operator stellt sich wider Erwarten als wirkungslos heraus	Ärger, Frustration

Tabelle A.1: Emotionale Begleiterscheinungen bei Handlungen und Problemlösen.

Motivation	**Ziele / Bedeutungen**
1. Neugier	Abwechslung / Neuheit / Wißbegierde / Horizonterweiterung
2. Leistung	Ehrgeiz / Erfolg / Perfektionismus / Effizienz / Wettbewerb
3. Kontakt	Ausleben bestehender o. Aufbau neuer Beziehungen
4. Macht	Dominanz / Führung / Kontrolle über andere
5. Sicherheit	Risikovorsorge / Vermeiden von Mißerfolgen, Schmerz, Krankheit
6. Hilfe (anderen)	Hilfe o. Unterstützung leisten / Schützen / Fürsorge
7. Hilfe (selbst)	unterstützt / angeleitet / beschützt werden
8. Bequemlichkeit	Vermeiden von Anstrengung, Zeitersparnis
9. Ordnung	Einfachheit, Verständlichkeit, Vorhersagbarkeit der Umwelt
10. Spiel	Zerstreuung / Unterhaltung / Ablenkung
11. Gewinn	Geld verdienen o. gewinnbringend anlegen / Sparen / günstige Geschäfte o. Käufe / Besitz mehren
12. Prestige	Bewunderung und Anerkennung durch sich selbst, reale oder nur vorgestellte Dritte
13. Sex	reale oder phantasierte sexuelle Aktivität
14. Emotion	Gefühlsbetonung / Aufregung, Risiko („sensation seeking“) / Vermeiden bzw. Herbeiführen negativer bzw. positiver Emotionen
15. Rückzug	Ruhe / Regeneration / Schlaf
16. Autonomie	Selbstbestimmung / Freiheit / Widerstand gegen Beeinflussung / Verteidigung der eigenen Werte und Meinungen

Tabelle A.2: Warum handeln Menschen? 16 Basismotivationen

Besonders Computerspiele versuchen möglichst viele Bedürfnisse anzusprechen. Neuere Online-Spiele, bei denen man sich mit tausenden anderen Spielern in einer virtuellen Welt bewegt, bieten eine Ersatzrealität, die viele Motivationen anspricht. Besonders Macht (4.), innerhalb der Spielwelt, aber über reale Personen, und Prestige (12.) kommen bei dieser Spielart hinzu.

B Longhorn-Display-Driver-Model (LDDM)

Da bei Windows-Vista jede Anwendung intensiv Gebrauch von der GPU und dem Grafikspeicher macht, ergeben sich neue Anforderungen an die Hardware. Keine Anwendung soll in den Grafikspeicher einer anderen schreiben können und die Leistung der GPU soll fair unter allen Anwendungen aufgeteilt werden.

Jede gängige CPU kann in einem Protected-Mode laufen, in dem der Hauptspeicher virtualisiert wird. Genauso gibt es seit langem Algorithmen für Multitaskingsysteme, die die CPU-Leistung fair unter den Anwendungen aufteilen. Die Grafikhardware ist leider nicht in der Lage, ihren Speicher zu virtualisieren. Die bisherigen Grafiktreiber waren darauf ausgelegt, dass nur wenige Anwendungen die Grafikhardware benutzen. Computerspiele haben meist sogar exklusiven Zugriff. Scheduling der GPU und Virtualisierung des Speichers sind also neue Anforderungen, die durch ein neues Treibermodell gelöst werden müssen.

Dieses neue Treibermodell wird Longhorn-Display-Driver-Modell (LDDM) genannt. Ziel bei diesem Modell ist nicht nur die eben angesprochene Aufteilung der Grafikhardware auf mehrere Anwendungen, sondern auch verbesserte Stabilität und Sicherheit. Die Grafikdarstellung ist einer der Hauptgründe für Systeminstabilität. LDDM soll deshalb sauber neu entwickelt werden, wobei so wenig wie möglich Code im Kernmodus läuft. Weiterhin soll die Toleranz gegenüber Hardwarefehlern verbessert werden und Nutzermodus-Code grundsätzlich nicht vertraut werden.

Bisher wurde Grafikspeicher nach dem „first come, first serve“-Prinzip vergeben. Wenn also nicht mehr genug Speicher da war, lief die Anwendung nicht. Mit LDDM hat jeder Prozess seinen eigenen linearen virtuellen Grafikspeicher, in dem er Texturen ablegen kann. Da dieser Vorgang nicht von der GPU unterstützt wird, muss die Umwandlung von virtuellen zu physichen Adressen per Software erfolgen. Abbildung B.1 zeigt anhand eines Ablaufes, wie Scheduling und Virtualisierung umgesetzt werden.

1. Die Anwendung schickt Zeichenkommandos zur D3D-Laufzeitbibliothek.

2. Die D3D-Bibliothek ruft den im Nutzermodus laufenden Teile des Grafiktreibers auf (UMD). Dieser wird für jeden Prozess geladen und erzeugt Command Buffers, die für den im Kernmodus laufenden Teil des Grafiktreibers (KMD) gedacht sind.

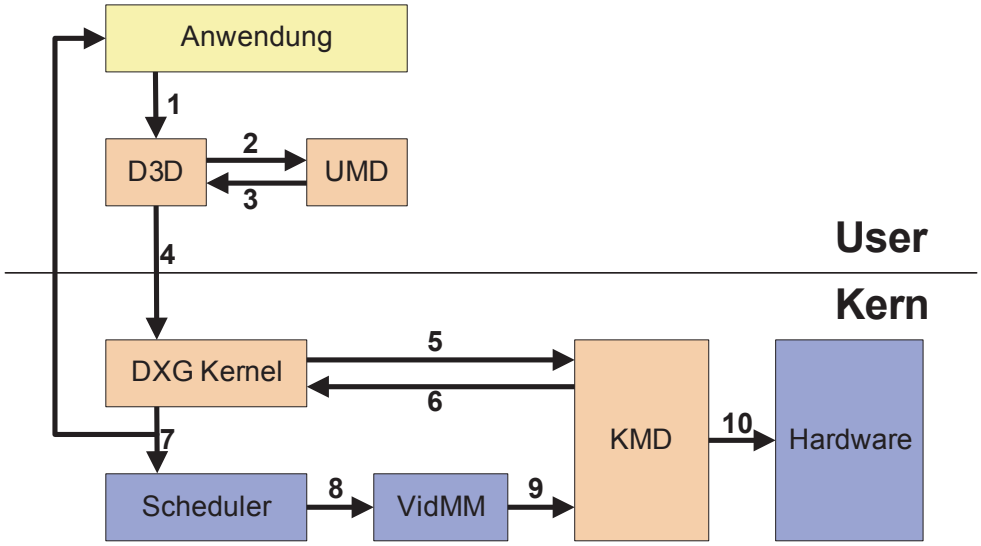

Abbildung B.1: Ablauf einer Grafikoperation beim Longhorn Display Driver Model.

3. Der Command-Buffer wird an die D3D-Bibliothek übergeben. Dieser Buffer enthält nur Handles, keine physischen Adressen.

4. Die D3D-Bibliothek übergibt den Command-Buffer an den Kernteil von DirectXG.

5. Der Command-Buffer wird an den Kernteil des Grafiktreibers (KMD) übergeben. Der KMD überprüft den Command Buffer und kopiert ihn in einen DMA-Buffer. Ausserdem wird ein Liste für benötigte Resourcen angehängt.

6. Der DMA-Buffer und Resourcenliste werden an DXG zurückgegeben.

7. Der DMA-Buffer und Resourcenliste werden an den Scheduler übergeben. Damit ist der synchrone Teil der Verarbeitung beendet und der Systemaufruf kehrt zur Anwendung zurück.

8. Der Scheduler entscheidet welchen DMA-Buffer er als nächstes bearbeitet, und sendet ihn an den Video Memory Manager (VidMM) damit dieser die physichen Adressen der benötigten Ressourcen einfügen kann.

9. Der VidMM übergibt den fertigen Buffer an den KMD.

10. Die Zeichenkommandos werden von der GPU ausgeführt.

Der beschriebene Vorgang erzeugt erhebliche CPU-Last, ist aber im Moment die beste Lösung um GPU-Sharing, Stabilität und Sicherheit zu erreichen. Insbesondere kann den Commandbuffers nicht vertraut werden, da sie aus dem Nutzermodus kommen. Sie müssen validiert werden um sicherzustellen, dass keine Anwendung auf fremden Grafikspeicher zugreift, oder Grafikhardware beschädigt bzw. abstürzen lässt.

C Looking-Glass Beispiel

Abbildung C.1 zeigt eine LG3D-Beispielanwendung. Das Beispiel stammt aus dem Tutorial auf der LG3D-Homepage [lg3a]. Es wird eine Box erzeugt und leicht rotiert. Das Tutorial besteht aus 8 Schritten. Abbildung C.2 zeigt einen Screenshot des Programmes.

1. Es wird root-Container erzeugt. Danach wird eine Box erzeugt. Die Box beschreibt sich über ihre Ausdehung und Erscheinung (Variable `app` vom Typ `SimpleAppearance`). Zuletzt wird die Box einem Component3D-Objekt hinzugefügt. Die Box ist nur ein geometrisches Objekt während die Komponente ein manipulierbares Interfaceobjekt ist.

2. Damit die Box nicht nur als Rechteck von vorne zu sehen ist, wird sie etwas rotiert. Component3D enthält Hilfsmethoden zum rotieren. Zuerst wird die Rotationsachse angegeben, danach der Winkel.

3. Die soll etwas grösser werden, wenn der Mauscursor über ihr ist. Dazu wird ein EventAdapter bei der Komponente registriert, der das Ereignis an eine Skalierungsaktion weitergibt. Damit sich die Skalierung fließend ändert, wird bei der Komponente auch noch eine Animationsklasse gesetzt. Damit das Mouseereignis auch noch vom Szenenmanager verarbeitet werden kann, darf es nicht „verbraucht“ (consumed) werden. Das wird durch die Anweisung `comp.setMouseEventPropagatable(tr` erreicht.

4. Der Mousecursor kann jede beliebige Form annehmen. In diesem Beispiel wird für die Box ein Kugelförmiger roter Cursor erzeugt und gesetzt.

5. Der Szenenmanager erzeugt für jede Anwendung ein Standart-17.07.2006Thumbnail für den Taskbar. In diesem Beispiel wird ein eigenes Thumbnail, eine kleine Box, erzeugt.

6. Zuletzt wird die Komponente zum Container hinzugefügt und sichtbar gemacht. Die Grössenangabe hilft dem Szenenmanager, Konflikte mit anderen Anwendungen zu vermeiden.

```
package org.jdesktop.lg3d.apps.tutorial;
import ...
public class Tutorial2 {
    public static void main(String[] args) {
        new Tutorial2();
    }

    public Tutorial2() {
        // Schritt 1: Initialisieren der 3D-Anwendung
        Frame3D frame3d = new Frame3D();
        SimpleAppearance app = new SimpleAppearance(0.6f, 0.8f, 0.6f, 0.7f);
        Box box = new Box(0.04f, 0.03f, 0.02f, app);
        Component3D comp = new Component3D();
        comp.addChild(box);

        // Schritt 2: Box rotieren
        comp.setRotationAxis(1.0f, 0.5f, 0.0f);
        comp.setRotationAngle((float)Math.toRadians(60));

        // Schritt 3: Skalieren der Box wenn unter Mauscursor
        comp.addListener( new MouseEnteredEventAdapter( new ScaleActionBoolean(comp, 1.2f)));
        comp.setAnimation(new NaturalMotionAnimation(500));
        comp.setMouseEventPropagatable(true);

        // Schritt 4: Erzeugen und setzen des Cursors
        SimpleAppearance cursorApp = new SimpleAppearance(1.0f, 0.0f, 0.0f, 0.5f);
        Sphere cursorBody          = new Sphere(0.002f, Sphere.GENERATE_NORMALS, 12, cursorApp);
        Cursor3D cursor            = new Cursor3D("Red␣Ball␣Cursor", cursorBody);
        cursor.setPreferredSize(new Vector3f(0.004f, 0.004f, 0.004f));
        comp.setCursor(cursor);

        // Schritt 5: Thumbnail erzeugen und setzen
        SimpleAppearance thumbnailApp = new SimpleAppearance(0.6f, 0.8f, 0.6f, 0.7f);
        Box thumbnailBody             = new Box(0.0052f, 0.0039f, 0.0026f, thumbnailApp);
        Thumbnail thumbnail           = new Thumbnail();
        thumbnail.addChild(thumbnailBody);
        thumbnail.setPreferredSize(new Vector3f(0.0104f, 0.0078f, 0.0052f));
        frame3d.setThumbnail(thumbnail);

        // Step 6: Container initialisieren
        frame3d.addChild(comp);
        frame3d.setPreferredSize(new Vector3f(0.08f, 0.08f, 0.08f));
        frame3d.changeEnabled(true);
        frame3d.changeVisible(true);
    }
}
```

Abbildung C.1: Looking-Glass-Beispielanwendung stellt eine rotierte Box dar. Dieses Beispiel ist Teil des Tutorials auf der LG3D-Homepage [lg3a]

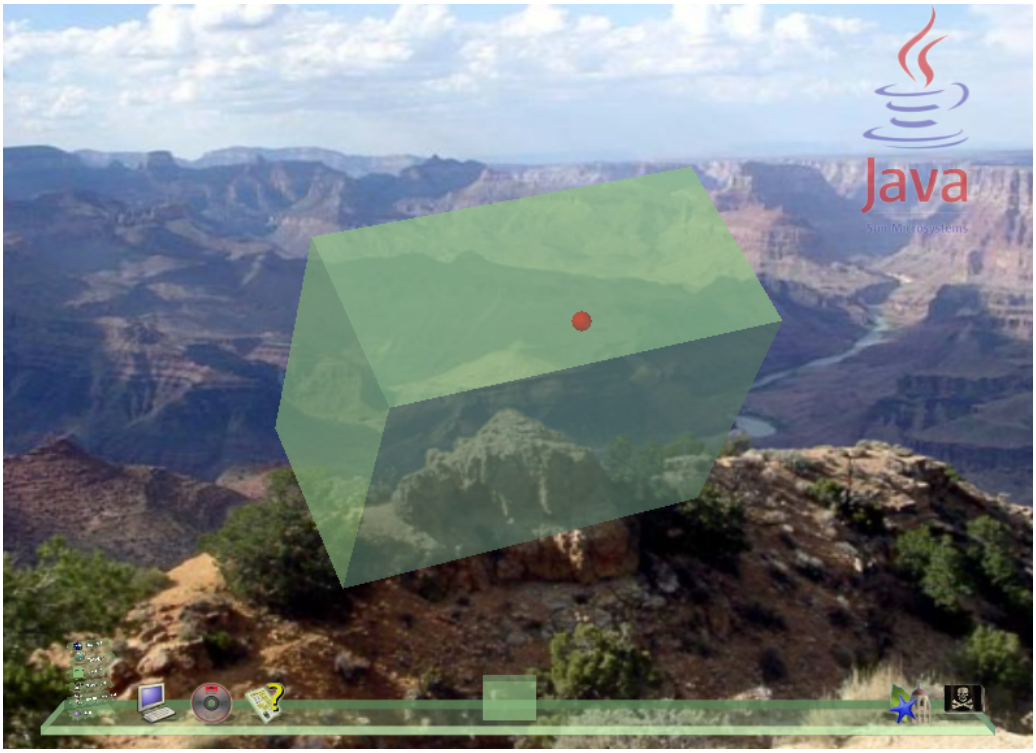

Abbildung C.2: Screenshot des Beispielprogrammes. Zu sehen ist die rotierte Box mit dem roten kugelförmigen Mauscursor und dem boxförmigen Thumbnail auf dem Taskbar.

D NakedObjects Beispiel

Dieses Kapitel beschreibt ein sehr einfaches Beispiel einer Anwendung für das NakedObjects-Framework Version 1.2.1. Es enthält nur eine Klasse und soll nur einen ersten Einblick geben. Umfangreichere Beispiele finden sich in [PM02] und [Paw04].

Ähnlich dem Java-Bean Konzept gibt es für ein NakedObject Regeln. Diese Regeln ermöglichen es dem Framework, z.B. die Objekte automatisch in die Oberfläche zu integrieren oder persistent zu machen. Die Beispielklasse enthält einen Namen, eine Collection mit Kindern und genau ein Elternobjekt. So eine Klasse könnte z.B. eine zu erledigende Aufgabe (engl. ToDo) sein, mit untergeordneten Teilaufgaben. Wenn man ein `ToDo` erledigt hat, soll man es als erledigt markieren können. (Abbildung D.1)

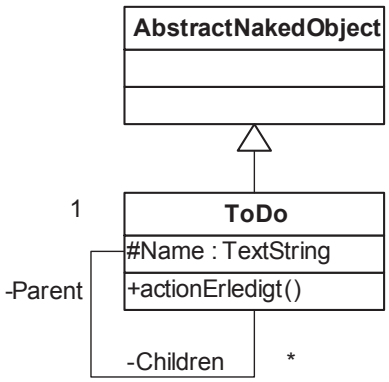

Abbildung D.1: UML-Diagram zum Beispiel

Jedes NakedObject wird von AbstractNakedObject abgeleitet. Für Collections und Primitive wie Texte und Zahlen gibt es eigene Klassen. In diesem Fall wird eine Textvariable für den Namen, eine Collection für untergeordnete Aufgaben und eine Vairable `parent` vom Typ `ToDo` benötigt. (Abbildung D.2)

```
public class ToDo extends AbstractNakedObject {
    private final TextString name;
    private final InternalCollection subToDos;
    private ToDo parent;
}
```

Abbildung D.2: Die Klasse ohne Methoden

Als nächstes kommt der Constructor und die Get- und Set-Methoden hinzu. Bis dahin sieht es fast wie eine normale JavaBean aus. (Abbildung D.3)

```
public TextString getName() {
    return name;
}

public final InternalCollection getSubToDos() {
    return subToDos;
}

public ToDo getParent() {
    resolve( parent );
    return parent;
}

public void setParent( ToDo newParent ) {
    parent = newParent;
    objectChanged();
}
```

Abbildung D.3: Get- und Setmethoden

In der Methode `getParent` wird die statische Methode `resolve` aufgerufen um sicherzustellen, daß das Objekt auch im Speicher geladen ist. Bei `setParent` muss dem Framework mitgeteilt werden, dass sich das Objekt geändert hat. Das wird durch `objectChanged()` erledigt.

Für dieses Beispiel fehlen noch zwei Dinge. Man soll das ToDo als erledigt markieren können und das Framework muss wissen, woher es den Anzeigenamen der Instanzen bekommt. (Abbildung D.4)

```
public void actionErledigt() {
    name.setValue( name.stringValue() + "␣(Eredigt)");
    objectChanged();
}

public Title title() {
    return name.title();
}
```

Abbildung D.4: Action und Title-Methode

Jede Methode die mit einem `action` beginnt, ist eine Aktion die der Benutzer auf diesem Objekt ausführen kann. In diesem Fall wird dem Namen nur ein „Erledigt“ angehängt, eigentlich sollte man hier zumindest noch ein Flag setzen. Aber es geht hier nur darum, dass etwas Sichtbares passiert. Damit die Änderung auch wirklich angezeigt wird, muss wieder `objectChanged()` aufgerufen werden.

Als letztes wird noch die Methode `title()` überschrieben. Sie liefert den Anzeigenamen das Objektes.

Die Klasse ist jetzt ein fertiges NakedObject. Um es in Aktion zu sehen, wird noch eine von `Exploration` abgeleitete Klasse benötigt. Dort wird dem Framework mitgeteilt, welche Objekte zu der Anwendung gehören. (Abbildung D.5)

```
public class ToDoExploration extends Exploration {
    public void classSet(NakedClassList classes) {
        classes.addClass( ToDo.class );
    }

    public static void main(String[] args) {
        new ToDoExploration();
    }
}
```

Abbildung D.5: Jede NakedObjects-Anwendung benötigt eine Klasse, die von `Exploration` erbt. Diese Klasse teilt dem Framework mit, welche Klassen zur Anwendung gehören.

Da es in dieser Arbeit um die Visualisierung der NakedObjects geht, zeige ich mehrere Screenshots zu diesem Beispiel. Im Kapitel 6.1 wird die Visualisierung genauer untersucht und beschrieben.

1. Abbildung D.6: Direkt nach dem Starten sieht man ein fast leeres Fenster. Links oben werden alle Klassen untereinander angezeigt, die in der Exploration eingetragen sind. Da kein Icon erzeugt wurde, wird ein Fragezeichen angezeigt. Icons werden im Working-Directory oder bei den Resourcen in einem Images-Unterverzeichnis gesucht, und müssen den gleichen Namen haben wie die Klasse, für die sie bestimmt sind. Unten wird der Name der Anwendung und der Nutzer angegeben. Das sind auch NakedObjects, die automatisch angelegt werden. Die nächsten Screenshots zeigen nur den relevanten Teil des Fensters.

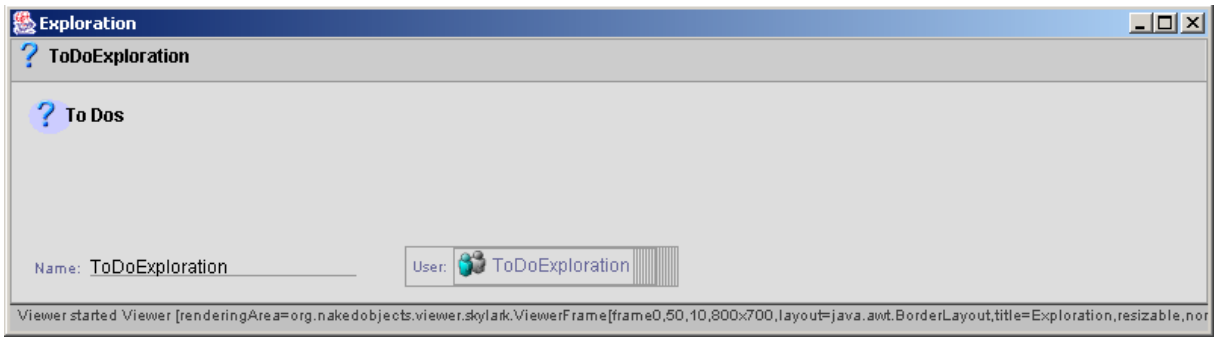

Abbildung D.6: NakedObjects-Anwendung direkt nach dem Start.

2. Abbildung D.7: Per Rechtsklick erscheint ein Popup-Menu. Der oberste Menupunkt soll helfen In-

stanzen der Klasse zu finden. In der für dieses Beispiel benutzten Version 1.2.1 von NakedObjects funktioniert das aber noch nicht. Der 2. Menupunkt zeigt alle Instanzen der Klasse in einer Liste untereinander an. Mit dem dritten Eintrag kann man neue Objekte anlegen.

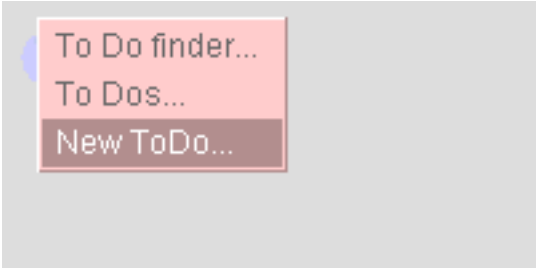

Abbildung D.7: Popup-Menu bei rechtem Mausklick auf eine Klasse.

3. Abbildung D.8: Nachdem ein `ToDo` erzeugt wurde, kann der Name eintragen werden. Da dieser auch als Titel definiert wurde, erscheint er im Kopf des Objektfensters neben dem Icon. Hier sieht man auch die noch leeren Referenzen auf genau ein Elternobjekt, gekennzeichnet durch ein oval und eine `Collection` von Kindobjekten, gekennzeichnet durch 3 kleine Kisten.

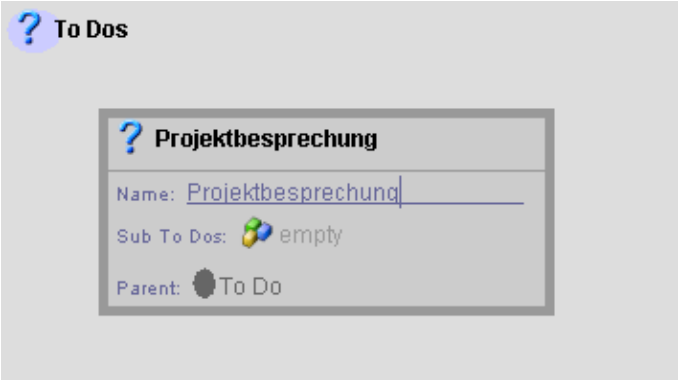

Abbildung D.8: Ein neu erzeugtes NakedObject mit eingetragenem Namen.

4. Abbildung D.9: Hier sind noch 3 weitere `ToDo`'s angelegt, die jetzt hierarchisch verknüpft werden sollen. Das geht ganz einfach per Drag and Drop. Auf dem Bild wird gerade die Aufgabe `Projektbesprechung` als Elternobjekt von `Terminvorschlag verschicken` gesetzt.

5. Abbildung D.10: Jetzt sind alle `ToDo`'s richtig miteinander verknüpft. Diese Art der Visualisierung vermittelt nicht die Baumstruktur, in der sich die `ToDo`'s befinden. Aber es musste auch keine einzige Zeile Code geschrieben werden für diese universelle Form der Visualisierung.

6. Abbildung D.11: Ein Rechtsklick auf eines der Objekte zeigt die möglichen Aktionen an, die auf diesem Objekt ausgeführt werden können. In diesem Fall gibt es nur die eine, die definiert wurde.

Abbildung D.9: Verknüpfungen zwischen NakedObjects werden per Drag and Drop erstellt.

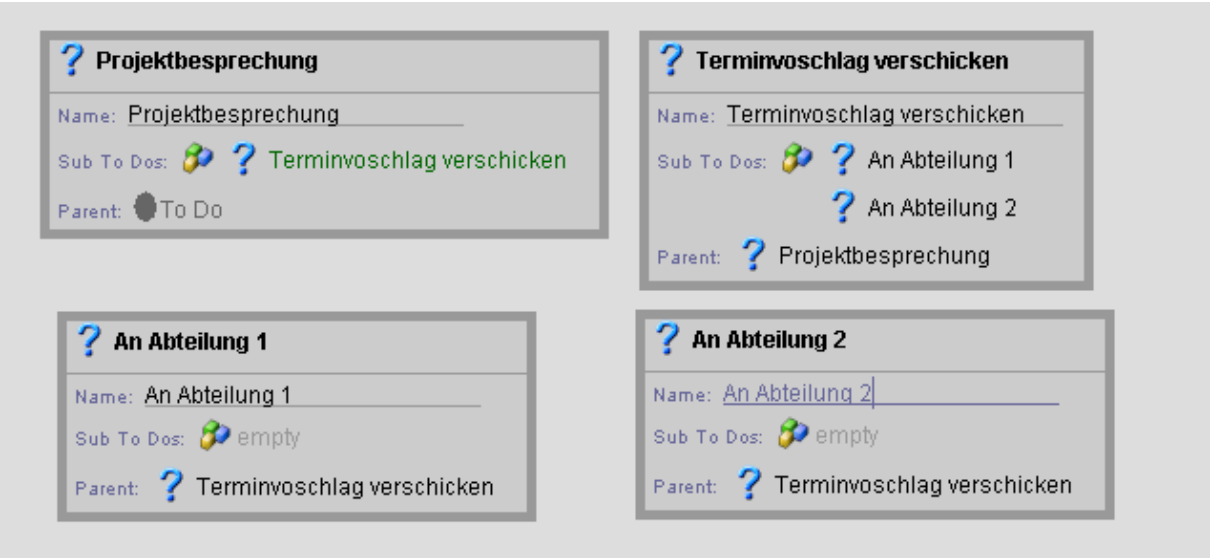

Abbildung D.10: 4 NakedObjects, die untereinander verknüpft sind.

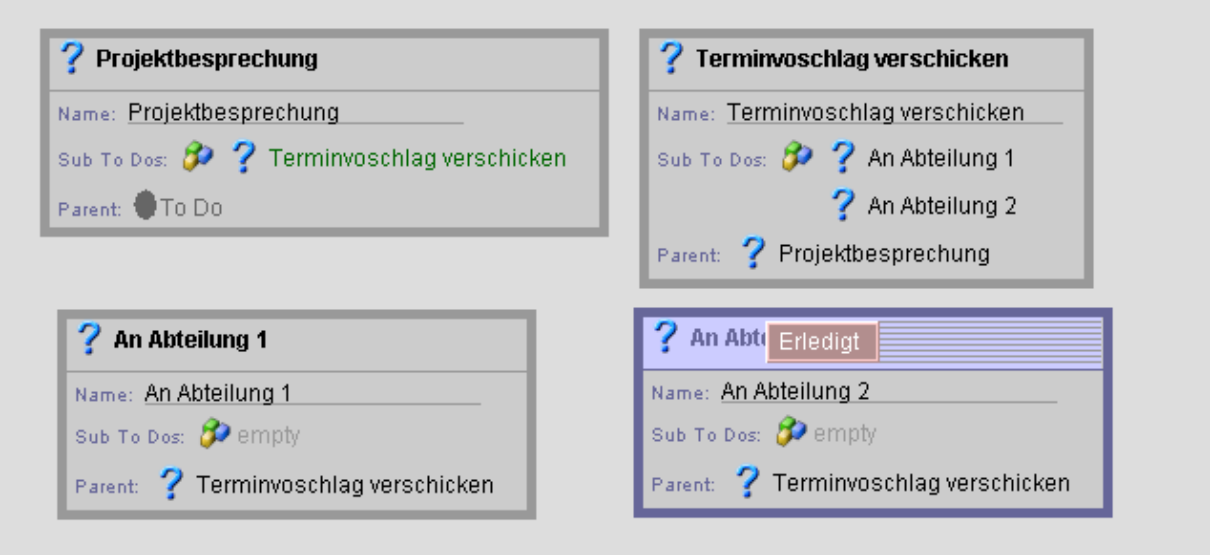

Abbildung D.11: Rechtsklick auf ein Objekt zeigt die Aktionen an die man auf ihm ausführen kann.

7. Abbildung D.12: Die Aktion hängt an den Namen den Text „Erledigt“ an. Das wird sofort auch im Titel sichtbar. Auch bei den `Sub To Dos` von `Terminvorschlag verschicken` sind jetzt 3 kleine Punkte hinter `An Abteilung 2` die andeuten, dass der Name noch länger geworden ist.

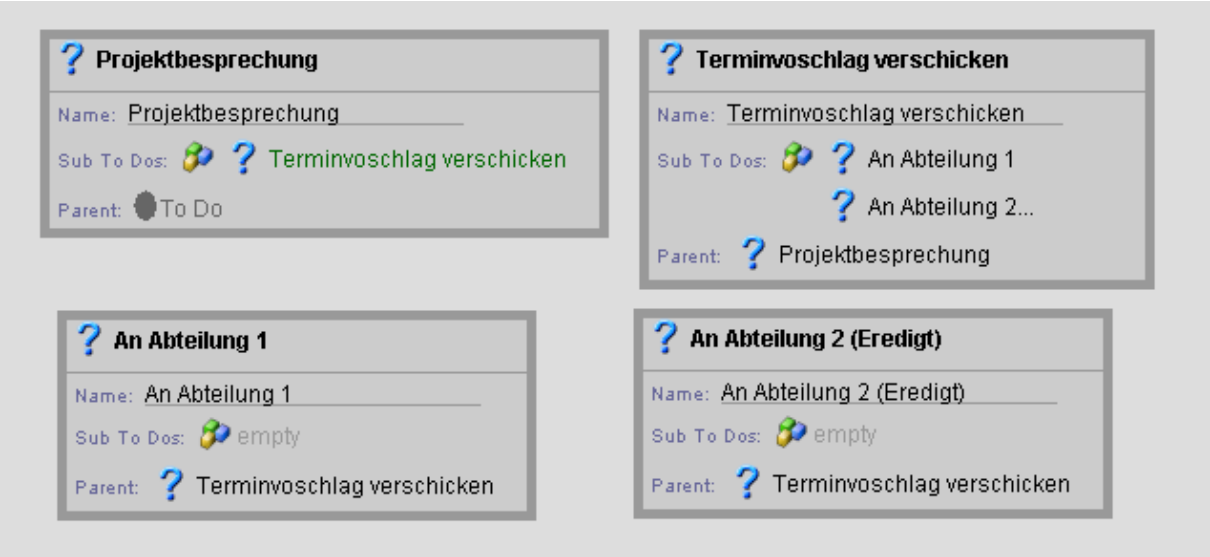

Abbildung D.12: Die Aktion hat den Namen des Objektes geändert, was sofort sichtbar wird.

Inhaltlich ist das ToDo-Beispiel an vielen Stellen noch unvollständig. Die Aktion `Erledigt` könnte z.B. auch seine Kinder auf erledigt setzen. Die Relationen zwischen den Objekten sollten eigentlich Bidirektional sein. Das heißt, wenn einem ToDo ein Kind-ToDo zugewiesen wird, sollte bei diesem Kind-ToDo auch automatisch das Eltern-Objekt gesetzt werden.

Abbildung D.13 zeigt den Code, den man der Klasse hinzufügen muss, um die Relation bidirektional zu machen. Zusätzlich zu den normalen Get- und Set-Methoden werden Associate- und Dissociate-Methoden hinzugefügt. Wenn diese vorhanden sind, werden sie vom Framework anstatt der Set- und Get-Methoden benutzt. Wenn man jetzt per Drag and Drop ein ToDo in das Elternfeld eines anderen zieht, wird die Kindliste des gezogenen ToDo's automatisch aktualisiert.

```
public void associateSubToDos( ToDo aToDo ) {
    getSubToDos().add( aToDo );
    aToDo.setParent( this );
}

public void dissociateSubToDos( ToDo aToDo ) {
    getSubToDos().remove( aToDo );
    aToDo.setParent( null );
}

public void associateParent( ToDo aToDo ) {
    aToDo.associateSubToDos( this );
}

public void dissociateParent( ToDo aToDo ) {
    aToDo.dissociateSubToDos( this );
}
```

Abbildung D.13: Bidrektionale Relationen.

Diese kleine Beispiel zeigt, wie einfach und unmittelbar man mit NakedObjects arbeiten kann. Das Geschäftsobjekt wird direkt sichtbar, ohne dass man extra Code dafür schreiben muss. [Nakc] zeigt weitere Bilder zur NakedObjects-Oberfläche. Eine genauere Einführung in die Programmierung mit NakedObjects bietet das Buch [PM02].

[]

Danksagung

Zuerst danke ich Steffen Gemkow. Er hat die Arbeit ermöglicht und sehr viel Verständnis für mich bewiesen. Er stand mir immer mit Rat und Tat zur Seite und hat den nötigen Druck für ein Vorankommen erzeugt. Ebenso gilt mein Dank Benjamin Neidhold. Er und Steffen Gemkow haben mich über den gesamten Zeitraum der Arbeit betreut.

Weiterhin danke ich meiner Oma für das Korrekturlesen, meiner Familie für die Geduld mit mir, und meinen Freunden für genügend Ausgleich zur Arbeit am Computer.

Erklärung

Hiermit erkläre ich, dass ich diese Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.