

**Integration von Geräten und Diensten  
des modernen Alltags in  
Benutzeroberflächen für verschiedene  
Plattformen**

WOLFGANG HAFENSCHER

DIPLOMARBEIT

eingereicht am  
Fachhochschul-Diplomstudiengang  
MEDIEN-TECHNIK UND -DESIGN  
in Hagenberg

im Juni 2004

© Copyright 2004 Wolfgang Hafenscher

Alle Rechte vorbehalten

# Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die aus anderen Quellen entnommenen Stellen als solche gekennzeichnet habe.

Hagenberg, am 11. Juni 2004

Wolfgang Hafenscher

# Inhaltsverzeichnis

<b>Erklärung</b>	<b>iii</b>
<b>Vorwort</b>	<b>vi</b>
<b>Kurzfassung</b>	<b>vii</b>
<b>Abstract</b>	<b>viii</b>
<b>1 Einführung</b>	<b>1</b>
1.1 Einleitung . . . . .	1
1.2 User Interface . . . . .	2
1.3 User Interface Design . . . . .	3
1.4 Verteilte Systeme . . . . .	6
<b>2 Software User Interface Design</b>	<b>8</b>
2.1 Einleitung . . . . .	8
2.1.1 Darstellungs-Ansätze . . . . .	9
2.1.2 Zoombare Benutzeroberflächen . . . . .	10
2.1.3 Mausgesten . . . . .	10
2.2 Menschliche Faktoren . . . . .	11
2.2.1 Faktoren nach Shneiderman . . . . .	12
2.2.2 Faktoren nach Spolsky . . . . .	15
2.3 Entwicklungsprozess . . . . .	16
2.3.1 LUCID . . . . .	16
2.3.2 Partizipatives Design . . . . .	17
2.3.3 Ethnographische Beobachtungen . . . . .	17
2.3.4 Activity-Based Planning . . . . .	17
2.4 Positiv- und Negativbeispiele . . . . .	21
2.4.1 Dynamische/statische Menüleiste . . . . .	21
2.4.2 Unentschlossener Programmierstil . . . . .	23
2.4.3 Alternatives Layout . . . . .	24
2.4.4 Verschachtelte Dialog-/Fensterhierarchien . . . . .	24
2.4.5 Speaking Geek . . . . .	25

<b>3</b>	<b>Konzept</b>	<b>29</b>
3.1	Spezifiziertes Szenario . . . . .	29
3.1.1	Oberflächen . . . . .	30
3.1.2	Geräte und Dienste . . . . .	31
3.2	Grundkonzept . . . . .	32
3.3	Client/Server-Kommunikation . . . . .	35
3.4	Grafische Passworteingabe . . . . .	38
<b>4</b>	<b>Implementierung</b>	<b>41</b>
4.1	Entwicklungs- und Testumgebung . . . . .	41
4.1.1	Entwicklungsumgebung . . . . .	41
4.1.2	Testumgebung . . . . .	42
4.1.3	TabletPC . . . . .	42
4.1.4	PDA . . . . .	42
4.2	Eingesetzte Technologien . . . . .	42
4.2.1	Client für Personal Computer . . . . .	43
4.2.2	Client für mobile Endgeräte . . . . .	47
4.3	Client für Personal Computer . . . . .	48
4.4	Client für mobile Endgeräte . . . . .	52
<b>5</b>	<b>Schlussbemerkungen</b>	<b>56</b>
<b>A</b>	<b>Inhalt der CD-ROM</b>	<b>58</b>
A.1	Diplomarbeit . . . . .	58
A.2	Implementierung . . . . .	58
A.3	Software . . . . .	58
	<b>Literaturverzeichnis</b>	<b>59</b>

# Vorwort

Nicht zum ersten Mal in meiner Laufbahn an der FH Hagenberg brachte mir mein mittlerweile mehrjähriger Coach *FH-Prof. Dipl.-Ing. Robert Kolmhofer* viel Vertrauen, Unterstützung und Hardware entgegen. Vielen Dank, Robert. Vielen Dank auch an den Layoutbetreuer des Projektes, welcher in der Lage war die Kreativität eines in künstlerischen Belangen eher unbeholfenen Diplomanden enorm zu steigern: *Mag Roland Keil*. Meinem Studienkollegen *Markus Thurner* gebührt ebenfalls großer Dank - für eine reibungslose und vor allem freundschaftliche Zusammenarbeit.

„Man soll dem Leib etwas Gutes bieten,  
damit die Seele Lust hat, darin zu wohnen.“

*Winston Churchill*

Es ist gut Menschen um sich zu haben, welche einem die persönlichen und sozialen Werte immer wieder in Erinnerung rufen. Jene Werte, welche beim intensiven Arbeiten leicht in Vergessenheit geraten. Die Personen, die mich während des 8. Semesters sehr beim Erhalt meiner Werte und vor allem beim Erhalt der hohen Lebensqualität unterstützt haben, sollen an dieser Stelle genannt sein: *Tomoe Himukai, Friedrich & Maria Hafenscher, Stefan Hafenscher* und die Sponsoren aus den *Häusern Hafenscher und Neuberger*. *René Löschl, Matthias Bauer* und *Ulli, Thoean & Joe*. In Barcelona: *Dominik & Kerstin* und *Martin & Birgitta*.

# Kurzfassung

In vorausgegangenen Projekten konnte das Interesse der Wirtschaft an Applikationen im mobilen Bereich und im Bereich der Heimautomation nachgewiesen werden. Diese und eine andere Diplomarbeit sollen in Zusammenarbeit diese Bereiche verknüpfen und ein attraktives Produkt entwickeln. Die andere Diplomarbeit setzt sich die Entwicklung einer Middleware, welche die Schnittstelle zu den anzusteuern und zu verwaltenden Diensten und Geräten darstellt, als Ziel.

Die Aufgabe dieser Diplomarbeit ist die Definition einer Schnittstelle zwischen der Middleware und den darauf aufsetzenden Benutzerschnittstellen, welche über die ausgearbeitete Schnittstelle das System steuern. Diese Diplomarbeit stützt sich auf zwei große Pfeiler.

Zum einen beschäftigt sie sich mit dem User Interface Design. Sie gibt einen Einblick in die Regeln und die Theorie des User Interface Designs im Allgemeinen und versucht diese Richtlinien für den Software-Bereich aufzuarbeiten und umzusetzen.

Zum anderen ist das Ziel dieser Diplomarbeit die Entwicklung von Benutzeroberflächen, welche sich an die Richtlinien des Software User Interface Designs halten. Nicht nur die Oberfläche, sondern auch die Schnittstelle und die Kommunikation von und zur Middleware sollen entwickelt werden. Als Interaktionsgeräte soll eine möglichst breite Masse an heute üblichen elektronischen Geräten dienen und bei der Umsetzung auf eine große Verträglichkeit mit unterschiedlichen Plattformen geachtet werden. Der Ansatz soll auch nach Möglichkeit ausschließlich mit dem Einsatz von frei verfügbaren Technologien das Auslangen finden.

# Abstract

In previous projects the economy showed interest in mobile applications and applications in the field of home automation. The target of this and another thesis in cooperation is to produce one attractive product by linking those two areas. The target of the other thesis is the development of a middleware which delivers the interface to the controlled and administrated services and devices.

The challenge of this thesis is the definition of an interface between the middleware and the user interface which is based on the developed interface and will control the system. The thesis has got two major scopes.

The theory of user interface design is within the first scope. It shows the general rules of user interface design and tries to apply those directives to the field of software user interface design.

Within the second scope of this thesis is the target of developing of user interfaces which follow the directives of software user interface design. The target is not only to develop the user interface, also the interface and the communication from and to the middleware should be developed. An as big as possible variety today's electronic devices should be able to be used as interaction devices and there should be a high degree of compatibility with multiple platforms. If possible the attempt should be based on free available technologies only.

# Kapitel 1

## Einführung

### 1.1 Einleitung

In den letzten Semestern durchgeführte Projekte an der Fachhochschule Hagenberg beschäftigen sich vermehrt mit Applikationen im mobilen Bereich. Ein Gebiet, das auch in der Wirtschaft auf großes Interesse stößt, ist die Heimautomation – die computergesteuerte Fernsteuerung und Verwaltung verschiedenster Dienste und Geräte im privaten Heim oder in gewerblichen Gebäuden.

Das Projekt *eibMobile* aus dem 5. Semester verwirklichte eine Gebäudesteuerung mittels mobilem Endgerät durch die Ansteuerung des *European Instabus* Systems von Siemens mittels Java-fähigem Mobiltelefon. Das Projekt *MobileWorkspaceFramework* aus dem 6. Semester setzte erfolgreich ein verteiltes System um, bei welchem ein zentraler Server zahlreiche mobile Einheiten verwaltete. Es war möglich Nachrichten und Arbeitsanweisungen über verschiedene Kanäle zu schicken und die Positionen der mobilen Einheiten abzufragen. Ein drittes (außerschulisches) Projekt in Kooperation mit Siemens realisierte ein ähnliches System wie das Projekt im 5. Semester. Allerdings war dies eine bidirektionale Verbindung, bei welcher auch der zentrale Server den Kontakt zum mobilen Endgerät initiieren konnte und außerdem wurde ein anderes Bus-System angesteuert. Dies teils preisgekrönten Projekte und eine große Resonanz von kommerziellen Interessenten belegen das Interesse der Wirtschaft und die Notwendigkeit solcher Systeme.

Im Rahmen dieser Diplomarbeit soll in Kooperation mit einer anderen ein neues System erschaffen werden, welches vielseitiger und intuitiver als die Ergebnisse der bisherigen Vorgängerprojekte an der FH Hagenberg ist. Im Rahmen einer zweiten Diplomarbeit entsteht eine Middleware, welche es erlaubt Dienste und Geräte des modernen Haushaltes zu steuern und zu verwalten. Im Gegensatz dazu beschäftigt sich diese Diplomarbeit ausschließlich mit der Entwicklung von User Interfaces, welche die Kommunikation zwischen der Middleware und unterschiedlichen Endgeräten abbilden. Die

Ausarbeitung der Kommunikation zwischen Middleware und Interface ist ebenfalls zentraler Bestandteil dieser Arbeit.

Die Betriebssysteme und Programmiersprachen von heute stellen den Softwareentwicklern bequeme APIs und Toolkits zur Erstellung von grafischen Standardkomponenten zur Verfügung. Allerdings lassen seit langem etablierte Standards im Software User Interface Design kaum Platz für Kreativität. Durch die rasche Entwicklung der Technologien - sowohl im Software- als auch im Hardwarebereich - ist es nun an der Zeit alternative Zugänge und Methoden zur Visualisierung von Interfaces zu finden.

Außerdem soll die Kommunikation zwischen Mensch und Maschine nicht nur für eingefleischte Computerspezialisten möglich sein, sondern auch für eine möglichst breite Masse von potentiellen Usern. Dies ist vor allem deswegen wichtig, da die Implementierung als unterstützendes Element im Haushalt für Alt und Jung gedacht ist.

Als Technologie zur Umsetzung soll vorwiegend - oder am besten ausschließlich - auf Open Source Produkte zurückgegriffen werden. Es ist ebenfalls wichtig, dass die Interfaces eine Plattformunabhängigkeit aufweisen. Der Einsatz der fertigen Lösung soll nicht nur auf *State-of-the-Art*-Komponenten, sondern auch auf handelsüblichen Hardwarekomponenten problemlos möglich sein.

## 1.2 User Interface

Oft wird der Begriff „User Interface“ unmittelbar mit der Oberfläche eines Computerprogrammes assoziiert. Natürlich ist auch die Oberfläche eines Computerprogrammes ein User Interface, jedoch sind User Interfaces nicht nur auf den Computer beschränkt.

Jedes Gerät und jedes Ding, mit dem wir interagieren, stellt uns eine Schnittstelle - das User Interface - zur Interaktion zur Verfügung. Abstrakt gesehen ist das User Interface also eine Schnittstelle zur Interaktion zwischen zwei Systemen.

Ist eines dieser Systeme ein Mensch (der „User“) und das andere nicht, so stellt das User Interface jene Schnittstelle dar, die es dem Benutzer ermöglicht Eingaben am anderen System zu tätigen und Ausgaben/Feedback des anderen Systems zu rezipieren (siehe auch [14], [12, S. 14–15]).

Demnach finden sich User Interfaces nicht nur in Computerprogrammen, sondern auch in allen erdenklichen Dingen des Alltags. Jedes Telefon bietet uns Eingabemöglichkeiten mittels Tasten und Feedback durch das Klicken der Tasten und akustische Signale im Telefonlautsprecher. Sogar Türen besitzen nach [14] ein User Interface. Es sind die Türklinken und Drückplatten zum Öffnen der Tür, die uns die Eingabe ermöglichen und es ist das Schloss, welches uns akustische Ausgaben liefert.

Weitere Definition von User Interfaces sind zB:

„Ein Interface bezeichnet [...] die Art und Weise, wie ein Produkt eine bestimmte Aufgabe ausführt - also was der Benutzer tun kann und wie das System darauf reagiert [15, S. 18].“

„The user interface is the part of a computer and its software that people can see, hear, touch, talk to, or otherwise understand or direct. The user interface has essentially two components: input and output. Input is how a person communicates his or her needs or desires to the computer. [...] Output is how the computer conveys the results of its computations and requirements to the user. [...] [7, S. 4]“

Es gibt nicht nur bidirektionale Interfaces. Auch Interfaces, welche nur Eingaben ermöglichen oder nur Ausgaben eines Systems liefern, sind Interfaces.

Im Folgenden werden für den Begriff „Interface“ gleichbedeutend die deutschen Synonyme „Oberfläche“ und „Schnittstelle“ verwendet. Der „User“ wird im Folgenden auch das eine oder andere Mal als „Benutzer“ bezeichnet.

### 1.3 User Interface Design

„User interface design is a subset of a field of study called human-computer interaction (HCI). Human-computer interaction is the study, planning, and design of how people and computers work together so that a person's needs are satisfied in the most effective way. HCI designers must consider a variety of factors: what people want and expect, what physical limitations and abilities people possess, how their perceptual and information processing system work, and what people find enjoyable and attractive. Technical characteristics and limitations of the computer hardware and software must also be considered [7, S. 4].“

User Interface Design beschäftigt sich - im Gegensatz zur häufigen Annahme - nicht ausschließlich mit den ästhetischen Aspekten von Benutzeroberflächen. Die Gestaltung von Benutzerschnittstellen beinhaltet wesentlich mehr Disziplinen als nur die optische Verpackung von User Interfaces. Wichtige Teilbereiche beschäftigen sich beispielsweise eingehend mit der menschlichen Psyche, oder zB mit Richtlinien zur Entwicklung und Konzeption von guten User Interfaces.

Unzählige Bücher beschäftigen sich mit dieser Materie. Jeder Autor hat seine eigene Definition von gutem Design. Leider gibt es nicht so wie in der Mathematik allgemeingültige Formeln, die zum Erfolg führen. Aber es gibt viele Übereinstimmungen bei den Meinungen der Experten.

Laut Cooper [3, S. 16-17] gibt es kein „gutes User Interface“. Die Qualität eines User Interfaces kann nur im Kontext mit der Benutzergruppe und ihren Gewohnheiten und ihrem Nutzungsverhalten bestimmt werden.

„[...] good design makes the user more effective [3, S. 17]“.

Es gilt unzählige Dinge zu beachten um dieses Ziel zu erreichen. Macht man sich Gedanken über folgende Punkte, sollte man ein akzeptables Ergebnis erreichen:

- *Mögliche Aktionen sichtbar machen.* Sehr wichtig ist es dem Benutzer seine aktuellen Möglichkeiten aufzuzeigen. Ein gutes Beispiel zur Demonstration dieser Notwendigkeit sind die aktuell erhältlichen Telefone. Die Telefone der heutigen Generation besitzen sehr viele Funktionen, aber trotzdem nur um ein bis zwei Tasten mehr als die erste Generation von Tastentelefonen. Um Aktionen wie zB das Einleiten einer Konferenzschaltung oder das Halten des Anrufes während eines Telefonates auszuführen, muss der Benutzer zahlreiche telefonspezifische Befehlssequenzen auswendig beherrschen. Diese Situation kann mit einfachen Mitteln verbessert werden. Zum Beispiel durch die Integration eines zusätzlichen Digitaldisplays. Das Display kann, auch wenn es nur in der Lage ist wenige Zeichen darzustellen, die möglichen Zusatzfunktionen während des Telefonates anzeigen. Zusätzliche Tasten werden nicht benötigt, denn es genügt ein Knopf um das Menü weiterzuschalten und eine Taste um eine Auswahl zu treffen [14, S. 17ff].
- *Sofortiges Feedback geben.* Es ist wichtig, dass der Benutzer unmittelbar auf seine ausgelöste Aktion Feedback erhält. Das Feedback kann sich verschiedener Medien bedienen und muss somit nicht unbedingt visuell sein. Auch akustische Signale können als Feedback dienen. Es ist auch wichtig als Feedback nicht nur Signale zu werten, die uns einen Erfolg oder einen Misserfolg der Aktion erkennen lassen. Auch eine Information darüber, dass eine Aktion initiiert wurde, ist notwendig. Es gibt Aktionen, welche ein Ergebnis erst nach einem gewissen Zeitraum liefern. Deshalb ist es ebenfalls notwendig den Benutzer auch darüber zu informieren, dass eine Aktion ausgelöst wurde und gerade ausgeführt wird. Ohne unmittelbares Feedback würde der User wahrscheinlich die gleiche Aktion mehrmals kurz hintereinander ausführen [14, S. 17-18], [12, S. 57-58].
- *Die Erfahrung des Benutzers nutzen.* Gelingt es dem User Interface Designer die Erfahrung des Benutzers in der bisherigen Welt für die Bedienung des Interfaces zu nutzen hat dies enorm positive Auswirkungen. Der User kann den Umgang mit der Schnittstelle schnell erlernen und benötigt keine Bedienungsanleitungen oder sonstige Merkhilfen [14]

- *Konsistenz aufrechterhalten.* Das User Interface sollte konstant aufgebaut sein. Dadurch ist die Benutzung des Interfaces und letztendlich auch der User effizienter. Je effizienter der User ist, desto positiver wird er das Interface wahrnehmen [17, S. 43ff], [12, S. 72ff].
- *Keep it simple.* Viele User Interfaces leiden unter der Last an Features und Gimmicks, die im vermeintlichen Interesse des Users integriert wurden. Nicht immer ist es vorteilhaft alle möglichen Raffinessen zu verwirklichen. Zu komplexe Aktionsmöglichkeiten können zu Verwirrung, vermehrten Fehlern und letztendlich zur Unzufriedenheit des Benutzers führen [14, S. 172-174].
- *Beschränkungen bewusst einsetzen.* Beschränkungen von Aktionen erleichtern den Umgang mit dem User Interface. Die möglichen Fehler werden auf ein Minimum reduziert und der Benutzer kann den Zusammenhang zwischen Aktion und Ergebnis besser begreifen [14, S. 60-62, 81ff].
- *Alle möglichen Fehler behandeln.* Man sollte prinzipiell davon ausgehen, dass von allen möglichen Fehlern alle gemacht werden. Daher sollte der Fehlerbehandlung erhöhte Aufmerksamkeit geschenkt werden und alle Fehlerquellen sollten entsprechend behandelt werden [14, S. 105ff].

Diese Liste erhebt bei weitem keinen Anspruch auf Vollständigkeit und es liegt im Ermessen des User Interface Designers, wie viel Gewicht er welchem Punkt beimisst. Weitere Punkte wären zB:

- *Intelligente Aktionslisten.* Unter diesem Begriff fasst man alle Möglichkeiten für den Benutzer, Menüs dynamisch zu gestalten, zusammen. Sei es ob sich das System die zuletzt oder am öftesten getätigten Aktionen merkt, oder ob sich der User ein personalisiertes Menü aus gewünschten Aktionen selbst zusammenstellen kann [8, S. 62ff].
- *Multilingualität.* Die Möglichkeit ein User Interface in mehreren Sprachen erleben zu können ist nicht alles, was mit diesem Punkt gemeint ist. Denkbar ist es auch das Vokabular der gewählten Phrasen an die Zielgruppe anzupassen. Beispielsweise durch die Zurverfügungstellung von unterschiedlichen Versionen von ein und demselben Interface für erfahrene und unerfahrene Benutzer, oder Kinder und Erwachsene [17, S. 118ff], [16, S. 38-39].
- *Personalisiertes Layout.* Jeder Benutzer sollte die Möglichkeit haben aus verschiedenen Layouts zu wählen oder gar sein eigenes Wunschlayout erstellen zu können [12, S. 19ff], [12, S. 60-61]. Der Wunsch nach einem alternativen Layout kann auch durch eine Veränderung der

Bedingungen am Arbeitsplatz ausgelöst werden. Beispiele hierfür sind zB die Veränderung der Helligkeit am Arbeitsplatz, die Verwendung des Produktes auf unterschiedlich großen Ausgabegeräten, etc.

- *Respekt.* Es ist wichtig dem User Respekt zu zollen und ihn niemals als unwissend oder unfähig hinzustellen. Das User Interface sollte in jeder Situation höflich auftreten [3, S. 11ff].

Die einzelnen Schritte, wie man von nichts - außer dem Wissen um die wünschenswerten Eigenschaften eines User Interfaces - bis zum fertigen optimalen User Interface kommt, werden in einem späteren Kapitel detailliert behandelt.

## 1.4 Verteilte Systeme

Nach dieser kurzen Erläuterung der Begriffe „User Interface“ und „User Interface Design“ wird nun die Position des User Interfaces im Konzept des kompletten Systems des Diplomarbeitsszenarios bestimmt.

Beim Diplomarbeitsszenario (wie noch später in der Arbeit genauer beschrieben) handelt es sich um ein verteiltes System. Die Hauptmotivation verteilte Systeme zu realisieren liegt darin, gemeinsame Ressourcen zwischen verschiedensten Plattformen und Benutzern zu teilen. Eine Definition nach [18, S. 2] ist folgende:

„A distributed system is a collection of independent computers that appears to its users as a single coherent system.“

Folgende Herausforderungen werden nach [4, S. 16–25] an verteilte Systeme gestellt:

- *Heterogenität* Verteilte Systeme werden aus verschiedenartigen Plattformen, Netzwerken und Komponenten konstruiert und wollen gemeinsame Ressourcen teilen.
- *Offenheit* Die Erweiterbarkeit des Systems um neue Elemente und der problemlose Austausch bestehender Elemente durch neue Technologien soll gewährt sein.
- *Sicherheit* Der Schutz vor nicht berechtigten Zugriffen, der Schutz vor Integritätsverletzungen und der Schutz vor der Beeinflussung der Absicht des Ressourcenzugriffs soll gewährleistet sein.
- *Skalierbarkeit* Der Aufwand zur Erweiterung des Systems um weitere User soll konstant sein.

- *Fehlerbehandlung* Jede Komponente des Systems soll die Fähigkeit haben auf den Ausfall aller in Kontakt stehenden Komponenten reagieren zu können.
- *Gleichzeitiger Zugriff* Bei der Präsenz mehrerer User in einem verteilten System soll das System fähig sein auf den gleichzeitigen Zugriff auf ein und die selbe Ressource reagieren zu können.
- *Transparenz* Verteilte Systeme sollen jene Aspekte des Systems, welche für Entwickler von Applikationen für das System nicht relevant sind, verstecken.

Die aus dieser Diplomarbeit hervorgehende Implementierung entwickelt nur einen Teil des komplexen verteilten Systems: Das Interface, welches der User nutzt, um mit dem verteilten System zu kommunizieren. Die Anbindung des Interfaces - was im Allgemeinen nichts anderes ist als die Kommunikation zwischen Komponenten im verteilten System generell - an das restliche System kann auf verschiedene Arten erfolgen.

Eine häufig genutzte Methode ist die Verwendung von bestehenden Protokollen (zB das Hypertext Transfer Protocol - HTTP). Es kann aber auch ein eigenes proprietäres Protokoll - beispielsweise auf der Basis einer Socket-Verbindung - implementiert werden. Der große Unterschied ist, dass bestehende und etablierte Protokolle bereits lange Zeit erfolgreich im Einsatz sind und mögliche Schwachstellen bekannt und weitgehendst eliminiert sind. Zu bestehenden Protokollen gibt es außerdem in den meisten Fällen bereits zahlreiche APIs in verschiedensten Programmiersprachen und der Aufwand zur Realisierung der Kommunikation hält sich so in Grenzen. Unter Umständen ist aber eine eigene Routine zur Kommunikation schneller, flexibler oder sicherer als bestehende Standards.

Eine interessante Alternative stellen APIs dar, welche die Verteilung des Systems für Softwarekomponenten unsichtbar machen. Jede Komponente kann auf Objekte und Methoden anderer Komponenten zugreifen und der Entwickler muss sich nicht um die darunterliegenden Schichten bemühen. Beispiele für solche APIs sind Java RMI<sup>1</sup>, .NET Remoting<sup>2</sup> oder CORBA<sup>3</sup> [4, 18].

---

<sup>1</sup><http://java.sun.com/products/jdk/rmi/>

<sup>2</sup><http://msdn.microsoft.com/>

<sup>3</sup><http://www.corba.org/>

## Kapitel 2

# Software User Interface Design

### 2.1 Einleitung

Für das Software User Interface Design gelten ebenfalls die in Kapitel 1.3 vorgestellten Prinzipien. Der große Unterschied zum Interface Design von Alltagsgegenständen liegt wohl darin, dass sich die Entwicklung von Software Interfaces im Wesentlichen auf zwei Rahmenbedingungen beschränkt: auf das Betriebssystem und auf die Programmiersprache. Natürlich spielen auch andere Bedingungen wie zB Bildschirmauflösung, Art des Endgerätes, etc. eine wichtige Rolle, jedoch muss auf diese nur bedingt Rücksicht genommen werden.

Eine Technik, welche unabhängig von der Bildschirmauflösung funktioniert, ist die Skalierung der darzustellenden Inhalte auf die richtige Bildschirmgröße. Besonders vorteilhaft ist hierbei die Verwendung von *Vektorformaten* zur Grafikdarstellung, welche keinen Qualitätsverlust bei Skalierung aufweisen. Der Trend zu Benutzeroberflächen, die Vektorgrafiken einsetzen, ist deutlich an aktuellen Softwareprodukten und neuen XML-Formaten zu erkennen. Die nächste Generation von XML-Formaten, welche gezielt zur Entwicklung von User Interfaces gedacht sind, wurde bereits eingeläutet. Der Entwickler wird weitgehend von aufwändigen Grafikoperationen befreit, da die Formate - ähnlich High-Level-APIs - die primitiven Funktionen verbergen und vordefinierte Grafikobjekte, Funktionen zur Manipulation und zahlreiche andere Features zur Verfügung stellen.

Zum aktuellen Zeitpunkt wirken eine Vielzahl an Programmen eines Betriebssystems mit grafischer Oberfläche auf den ersten Blick nahezu ident. Der Grund dafür liegt darin, dass zahlreiche Programmiersprachen (wie zB C++, C# [11], Java) Bibliotheken und Frameworks zur Erstellung von grafischen Benutzeroberflächen zur Verfügung stellen. Sie bieten im Aussehen standardisierte Fenster, Anzeigeflächen, Menüleisten, Titelleisten und alle

möglichen anderen standardisierten Elemente (zB Buttons, Schaltflächen, Tabellen) und lassen aber dennoch einen gewissen Grad an Flexibilität zu. Solche Standardkomponenten können sehr einfach erzeugt und gewisse Eigenschaften, wie zB die Farbe, die Schrift und die Größe, verändert werden. Das grundsätzliche Look-And-Feel der Anwendung bleibt aber dennoch erhalten und der User kann Parallelen zwischen den Programmen erkennen. Das Ziel der Implementierung dieser Diplomarbeit ist unter anderem das Ausbrechen aus diesem Schema und die Erzeugung eines Look-And-Feels, welches keinem Betriebssystem und keiner Programmiersprache zugeordnet werden kann.

### 2.1.1 Darstellungs-Ansätze

Mit dem Nachfolger des WindowsXP Betriebssystems von Microsoft<sup>1</sup> - Microsoft Longhorn<sup>2</sup> - führt Microsoft eine neues Beschreibungsformat ein. Das von Microsoft entwickelte Extensible Applikation Markup Language<sup>3</sup>-Format (XAML) stellt dem Programmierer unter anderem zahlreiche Vektorgrafikfunktionen zur Verfügung. XAML-Entwickler können außerdem bequem über XAML auf einen vordefinierten Pool an Standardkomponenten der heute üblichen Microsoft-Programme zurückgreifen. Im Bereich Web-Applikationen setzt Macromedia<sup>4</sup> auf das neue Server-Produkt Flex in Verbindung mit Macromedia Flex Markup Language (MXML) [2]. MXML ist eine Beschreibungssprache zur Entwicklung von Benutzeroberflächen mittels Integration von bereits vorhandenen Standards (wie zB XML, CSS, SVG). Die vom W3C<sup>5</sup> seit Oktober 2001 entwickelte und im Jänner 2003 veröffentlichte Beschreibungssprache Scalable Vector Graphics<sup>6</sup> (SVG) für 2D Vektorgrafiken kann wie schon erwähnt bei MXML-Applikationen verwendet werden und findet bereits jetzt in zahlreichen Applikationen und Programmiersprachen Unterstützung. Die Sprache beschränkt sich jedoch nicht nur auf die Beschreibung von 2D-Objekten, sondern stellt auch zahlreiche Zusatzfunktionen wie beispielsweise Transformationen, Filter und Animation zur Verfügung.

Abgesehen von der Darstellung von User Interfaces gibt es zwei noch nicht etablierte Entwicklungen im Bereich Software User Interface Design, welche sich vor allem auf die Usability von User Interfaces auswirken. Diese zwei Entwicklungen sind zoombare Benutzeroberflächen und Mausgesten, wobei vor allem zoombare Oberflächen zum aktuellen Zeitpunkt wenig Verbreitung finden. Leider sind aber beide Entwicklungen generell selten im

---

<sup>1</sup><http://www.microsoft.com/>

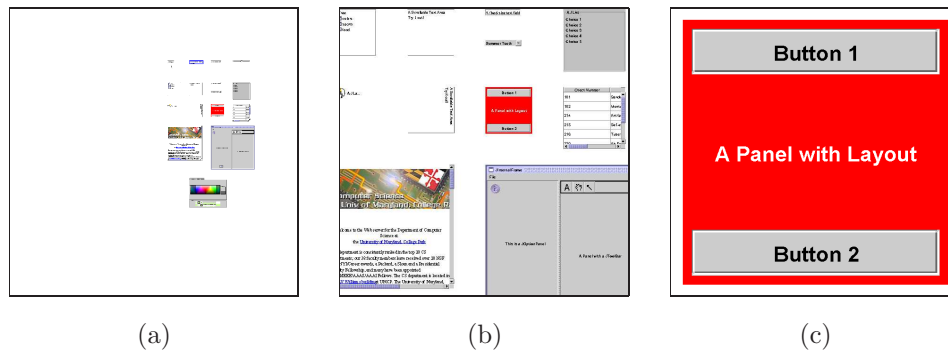
<sup>2</sup><http://www.microsoft.com/windows/longhorn/>

<sup>3</sup><http://www.xamlon.com/>

<sup>4</sup><http://www.macromedia.com/>

<sup>5</sup><http://www.w3.org/>

<sup>6</sup><http://www.w3.org/Graphics/SVG/>



**Abbildung 2.1:** Beispiel einer zoombaren Oberfläche anhand einer Demo-Implementierung des Projektes *Jazz*. Diese Demo-Implementierung (*Swing-Jazz Example*) ist im Lieferumfang des Projektes enthalten. (a) Unendlich große Oberfläche aus großer Distanz, (b) Zoom zu der Elementgruppe, (c) Konzentration auf ein einzelnes Element der Elementgruppe.

alltäglichen Umgang mit dem Computer integriert.

### 2.1.2 Zoombare Benutzeroberflächen

*Zoombare Benutzeroberflächen* ermöglichen einen schnellen und einfachen Zugang zu beliebig großen Datenmengen. Man kann sich zoombare Interfaces so vorstellen, dass der Benutzer eine unendlich große Fläche vor sich hat, welche er beliebig schnell zu sich ziehen oder von sich weg stoßen kann. Auf der Oberfläche kann der Interface Designer nun alle Daten beliebig anordnen und der Benutzer kann dann auf der Oberfläche navigieren. Aus weiteren Entfernungen bekommt der Benutzer einen guten Gesamtüberblick und behält so immer die Orientierung. Bewegt er sich nahe an eine beliebige Stelle, kann er so zu den einzelnen Elementen navigieren und mit den Elementen interagieren. Abb. 2.1 zeigt dies anhand eines Beispiels des Projektes *Piccolo*<sup>7</sup>. Das Projekt *Piccolo*<sup>7</sup> geht aus dem Projektes *Jazz* (beides Projekte am Human Interaction-Lab an der University of Maryland) hervor. Es ist ein auf der Java2D-API von Sun Microsystems basierendes Toolkit, welches es ermöglicht zoomable Interfaces in Java zu entwickeln. In [15, S. 180 ff.] werden unter dem Begriff *Zoom World* ebenfalls zoomable Interfaces vorgestellt.

### 2.1.3 Mausgesten

*Mausgesten* sind Tastatur-Kurzbefehlen sehr ähnlich. Bei Keyboard Shortcuts werden Tasten(-kombinationen) mit Aktionen belegt, bei Mausgesten

<sup>7</sup><http://www.cs.umd.edu/hcil/piccolo/>



### 2.2.1 Faktoren nach Shneiderman

1. *Physische Fähigkeiten und physische Arbeitsstätten.* Zahlreiche äußere Faktoren können sich direkt, sowohl positiv als auch negativ, auf die Leistung und die Fehlerquote des Benutzers, welcher in Interaktion mit einem User Interface steht, auswirken. Einige Faktoren sind zum Beispiel die Lautstärke der Umgebung, die Beleuchtung oder die Temperatur. Die richtige Gestaltung der Arbeitsstätte wirkt sich also direkt auf den Arbeitserfolg des Benutzers aus.

Um den Arbeitsplatz richtig zu gestalten müssen wiederum viele Faktoren berücksichtigt werden. Zum Beispiel liefert die Anthropometrie („*Ein Verfahren der biologischen Anthropologie zur metrischen und numerischen Erfassung von Körper- und Skelettmerkmalen.*“ [20]) zahlreiche Untersuchungsergebnisse über die physischen Merkmale der Menschen. Im Laufe der Geschichte der Anthropometrie wurden Tausende Messreihen über Hunderte menschlicher Eigenschaften durchgeführt. All diese Daten liefern Durchschnittswerte über verschiedene Merkmale und können zur Verwendung herangezogen werden. Doch die Anthropologie alleine liefert nur einen Teil der zu berücksichtigenden Daten.

„Das physische Design von Arbeitsräumen wird oft unter der Bezeichnung *Ergonomie* diskutiert. Anthropometrie, Soziologie, Arbeitspsychologie, Verhaltensweisen in Organisationen und Anthropologie können nützliche Einsichten in diesen Bereich bieten. [16, S. 34]“

Der Grund, warum man die Berichte und Ergebnisse dieser Forschungsgebiete als Grundlage zur Gestaltung des Arbeitsplatzes heranzieht liegt darin, dass der Computer hierbei nur eine untergeordnete Rolle spielt und die menschlichen Bedürfnisse und Fähigkeiten im Vordergrund stehen.

2. *Kognitive und perzeptorische Fähigkeiten.* Es ist sehr wichtig, dass der Designer die kognitiven und perzeptorischen Fähigkeiten der Benutzer kennt. Die nachfolgende Tabelle zeigt eine Klassifikation kognitiver Prozesse und Beeinflussungsfaktoren der perzeptorischen und motorischen Leistung der Zeitschrift *Ergonomics Abstracts*:

<i>Klassifikation menschlicher kognitiver Prozesse</i>	<i>Faktoren, die die perzeptorische und motorische Leistung beeinflussen</i>
Kurzzeitgedächtnis, Langzeitgedächtnis und Lernen, Problemlösung, Entscheidungen treffen, Aufmerksamkeit und Bereitschaft (Bereich des Interesses), Absuchen und Ermitteln, Zeitwahrnehmung	Erregung und Wachsamkeit, Ermüdung, Perzeptorische (mentale) Belastung, Kenntnis der Ergebnisse, Eintönigkeit und Langeweile, Sensorische Deprivation, Angst und Schrecken, Isolation, Altern, Drogen und Alkohol, Tagesrhythmen

„Die menschliche Fähigkeit, sensorische Eingaben rasch zu interpretieren und daraus komplexe Handlungen zu initiieren, macht moderne Computersysteme möglich. Anwender erkennen in Millisekunden minimale Änderungen auf ihrer Anzeige und beginnen, eine Folge von Befehlen zu initiieren. [16, S. 34]“

3. *Unterschiede in der Persönlichkeit.* Zu den zuvor genannten Gebieten gibt es auch auf dem Gebiet der Persönlichkeitsforschung viele Untersuchungen. Auch kann das Wissen um die Persönlichkeit im Falle einer spezifischen Zielgruppe beim Entwerfen eines Interfaces sehr positive Aspekte einbringen. Bemerkenswert ist, dass durch die Untersuchung der Vorlieben von Männern und Frauen bei beispielsweise Computerspielen keine messbaren Kriterien festgehalten werden konnten, welche Spiele einem der beiden Geschlechter attraktiver erscheinen. Trotzdem sollte der Designer in der Lage sein das Interface auch mit den Augen des anderen Geschlechts zu sehen. Beispiele für eine undurchdachte Wortwahl bei Befehlen in der Computerwelt, welche eventuell bei weiblichen Benutzern negative Gefühle hervorrufen, sind zB die Befehle *abort* und *kill*. Wie schon erwähnt gibt es keine geschlechterspezifischen Kriterien, aber man kann Persönlichkeitstypen allgemein einteilen. Eine Klassifizierungsmethode ist die Verwendung des Myers-Briggs Type Indicator (MBTI, siehe Tabelle 2.1).

„Die Theorie hinter dem MBTI stellt Porträts der Beziehungen zwischen Berufen und Persönlichkeitstypen und zwischen Menschen mit unterschiedlichen Persönlichkeitstypen vor. Sie wurde auf Testreihen für Nutzergemeinschaften angewandt und bot Richtlinien für Designer [16, S. 37].“

4. *Kulturelle und internationale Vielfalt.* Es gibt zahlreiche von der Kultur und der Gesellschaft, in der ein Mensch aufwächst, beeinflusste

Extraversion - Introversion
Empfindung - Intuition
Perzeptiv - urteilend
Fühlend - denkend

**Tabelle 2.1:** Der MBTI beruht auf der Theorie der Persönlichkeitstypen nach C.G. Jung. Nach Jung gibt es vier Wertepaare [16, S. 37ff].

Faktoren, die sein Verhalten und seine Gewohnheiten beeinflussen. Auch diese Faktoren gehören beim Design einer Benutzerschnittstelle berücksichtigt. Beispielsweise haben Menschen, welche westliche Sprachen als Muttersprachen haben, eine andere Art ihre Augen über den Bildschirm zu bewegen als Menschen, die primär Lesen von Chinesisch oder Japanisch gelernt haben. Im Design muss also bei Produkten mit einer weit gestreuten Zielgruppe nicht nur die einfache Übersetzung in eine andere Sprache berücksichtigt werden, sondern es müssen auch andere Faktoren miteinbezogen werden. Weitere Beispiele wären zB die Etikette, die Politik, der Tonfall, Förmlichkeiten, Metaphern, Farben, numerische und Währungsformate, usw.. In früheren Zeiten war es leicht über diese Art der Designfehler hinwegzusehen, aber heutzutage kann eine effektivere Lokalisierung einen enormen Wettbewerbsvorteil für ein Produkt bedeuten.

5. *Behinderte Anwender.* Es gibt zahlreiche Entwicklungen - sowohl im Hardware-, als auch im Softwarebereich - welche es behinderten Menschen möglich machen den Computer nutzen zu können. Für Menschen mit beeinträchtigtem Gehör genügen oft kleine Anpassungen, um das Produkt trotzdem nutzen zu können. Bildschirme mit größerer Anzeigefläche, teilweises Vergrößern von Bildschirminhalten oder Braille Ein- und Ausgabegeräte ermöglichen Menschen mit Sehschwäche die Interaktion mit dem Computer. Auch wenn man Zugänge für Behinderte nicht komplett realisiert, ist es von Vorteil bereits beim Planen Schnittstellen anzudenken und vorzusehen. Eine Im-

plementierung der zusätzlichen Hilfen für Behinderte ist dann um einiges einfacher und verursacht weit weniger Aufwand und Kosten. Manche Dinge, die behinderten Menschen den Umgang mit der Software ermöglichen, erleichtern auch oftmals nicht behinderten Menschen den Umgang mit der Software. Beispielsweise hilft eine durchdachte, präzise, einfache Formulierung von Arbeitsanweisungen, Informationstexten, etc. nicht nur lern- und leseschwachen Menschen, sondern jedem.

6. *Ältere Anwender.* Die letzte Gruppe von Menschen, welche bis jetzt noch nicht genannt wurde und welcher der Designer auch besondere Aufmerksamkeit schenken sollte, ist die Gruppe der alten Menschen. Zwar ist die aktuelle Generation von „alten Menschen“ zumindest teilweise bereits mit dem Computer an sich vertraut, aber trotzdem machen es Veränderungen des alternden Menschen notwendig Software speziellen Bedürfnissen anzupassen. Mit zunehmendem Alter lässt die visuelle und auditive Schärfe nach und auch andere Körperfunktionen sind nicht mehr zu 100 Prozent einsatzfähig, aber auch diese Veränderungen kann man durch intelligentes Design berücksichtigen.

### 2.2.2 Faktoren nach Spolsky

In [17, S. 61-80] werden drei menschliche Faktoren, die es in Bezug auf Softwareentwicklung zu berücksichtigen gibt, genannt: *People can't read*, *People can't control the mouse* und *People can't remember*.

Mit *People can't read* meint der Autor, dass die Benutzer von Softwareprodukten weder das Handbuch noch Fehlermeldungen oder Hinweise lesen. Deshalb sollte bei der Gestaltung einer Oberfläche die Ausgabe von Textinhalten auf das nötige Minimum reduziert werden. Interface Entwickler unterliegen leicht der falschen Annahme, dass lang ausformulierte und großzügig beschreibende Fehlermeldungen, Hinweistexte und alle anderen Arten von Textinformationen dem User helfen den Inhalt besser zu verstehen. Zahlreiche Usability-Test unterstreichen aber die Tatsache, dass User kürzeren Texten mehr Aufmerksamkeit schenken als längeren, die unter Umständen nicht gelesen werden. Es ist also wichtig, alle Arten von textlicher Information so kurz und präzise wie möglich zu halten.

*People can't control the mouse* beschreibt die Eigenheiten des Menschen im Umgang mit dem Eingabegerät „Maus“. Es soll nicht heißen, dass Menschen keine Mäuse bedienen können, aber der User Interface Entwickler soll keine absolute Pixelgenauigkeit von Usern erwarten und Spielraum anbieten. Eine Vielzahl von Gründen kann dazu führen, dass der Einsatz der Maus unpräzise ist. Es gibt eine Vielzahl von mausähnlichen Geräten

(zB Trackball, Touchpads, etc.), welche nicht die Genauigkeit von Standardmäusen zulassen. In manchen Fällen führt die Umgebung, in der die Maus benutzt wird (zB unebene Oberflächen), zu Ungenauigkeit. Es gibt Menschen, die nicht die nötigen motorischen Fähigkeiten haben um eine Maus exakt zu bedienen und auch Anfänger im Umgang mit der Maus beherrschen die Maus meistens nicht absolut treffsicher. Die Steuerung mittels Maus sollte also keine hohe Präzision verlangen.

Der dritte Faktor - *People can't remember* - soll den Entwickler dazu anregen dem User so viel mentalen Aufwand wie möglich abzunehmen. Weiters wird empfohlen selbst Usability-Tests am Interface durchzuführen, indem man die Oberfläche möglichst tölpelhaft bedient. Zum Beispiel durch die Bedienung der Maus mit nur einem Finger oder wildem Herumklicken in den Menüs. Dadurch lässt sich feststellen, ob und wie die Software auf Fehler bei der Eingabe reagiert.

## 2.3 Entwicklungsprozess

Alle Methodiken zur Entwicklung von User Interfaces müssen die Anwender, welche die Schnittstelle nutzen werden, miteinbeziehen. Es gibt verschiedene Methodiken, wobei die eine den User mehr und die andere den User weniger miteinbezieht. Die Anfänge der Entwicklungsmethodiken erlaubten es dem Entwickler zwar sich an finanzielle und zeitliche Pläne zu halten, jedoch präsentierten sie keine Lösung zur Entwicklung benutzergerechter Schnittstellen. Im Laufe der Zeit entwickelten Wissenschaftler Design-Methoden, welche sich auf die Interfaces konzentrierten, und auf diese Methoden stützen sich die Design-Methoden der zweiten Generation. Diese kommerziell orientierten Ansätze bestehen aus einem Mix aus Managementstrategien zur Einhaltung der Finanz- und Zeitpläne und Strategien zur Erstellung der Benutzerschnittstelle. Eine dieser Methodiken ist die Logical User-Centered Interactive Design Methodology (LUCID).

### 2.3.1 Logical User-Centered Interactive Design Methodology

Diese Methodik spezifiziert einen sechsstufigen Entwicklungsprozess (siehe Tabelle 2.2). Bei LUCID stehen Prototypen mit Schlüssel-Screens - Screens, welche die bedeutendsten Navigationspfade des Systems beinhalten, im Vordergrund. Anwenderzentrierte Design-Methodiken greifen ebenfalls - wie LUCID - auf das Rapid Prototyping und die iterativen Usability-Tests zurück. Um das Produkt optimal für die Bedürfnisse der Nutzer entwickeln zu können bedient sich LUCID partizipativer Design-Sessions.

Stufe 1:	Entwicklung des Produkt-Konzepts
Stufe 2:	Durchführung von Forschungs- und Bedarfsanalysen
Stufe 3:	Design-Konzepte und Prototypen mit zentral wichtigen Screens
Stufe 4:	Wiederholtes Design mit Verfeinerungen
Stufe 5:	Implementierung der Software
Stufe 6:	Sicherstellung des Supports beim Rollout

**Tabelle 2.2:** Entwicklungsprozess der Logical User-Centered Interactive Design Methodology aus [16, S. 135].

### 2.3.2 Partizipatives Design

Beim partizipativen Design werden die Anwender zum Designprozess hinzugezogen. Projekte, die auf das partizipative Design zurückgegriffen haben, berichten von sehr positiven Ergebnissen und können oftmals auf zahlreiche Verbesserungen, die ohne den Einbezug der Anwender nie stattgefunden hätten, zurückblicken. Allerdings muss, wie auch bei allen anderen Methodiken, die Anwender direkt miteinbeziehen, auch hier damit gerechnet werden, dass eine Anteilnahme der Anwender unter Umständen höhere Kosten beim Entwicklungsprozess verursacht und der Prozess eventuell länger dauert als mit anderen Methodiken. Ebenfalls muss strengstens auf die Auswahl der miteinzubeziehenden Anwender geachtet werden. Teilnehmer, die nicht zur konstruktiven Mitarbeit bereit sind, können sich negativ auf den ganzen Prozess auswirken [16, S. 141-142].

### 2.3.3 Ethnographische Beobachtungen

Bei ethnographischen Beobachtungen nimmt ein *Ethnograph* inkognito oder angekündigt am Alltag der Zielgruppe teil. Seine Aufgabe ist es den Alltag mitzuerleben, Fragen an die Anwender zu stellen, zu hören was gesagt wird und alle Vorgänge zu beobachten. Die erhobenen Daten nehmen dann entsprechenden Einfluss auf das Re-Design von existierenden Schnittstellen [16, S. 138-141]. Bei geheimen ethnographischen Beobachtungen wirken sich die Anwender im Vergleich zu offenen ethnographischen Beobachtungen nur indirekt auf den Entwicklungsprozess aus.

### 2.3.4 Activity-Based Planning

In [17, S. 81ff] wird das Activity-Based Planning vorgestellt. Da es eine schlechte Methode zur Entwicklung von Interfaces ist alle Funktionalitäten der Software zu planen und die Befehle für jede Funktionalität der Reihe nach in einem Menü anzuführen, sollte man einen anderen Ansatz wählen: das Activity-Based Planning, welches beispielsweise auch bei Microsoft zum

Einsatz kommt. Hierbei überlegt sich der Entwickler die Software aus der Sicht des Benutzers. Der Entwickler überlegt sich, welche Aktionen der User mit der Software durchführen will und gestaltet die Menüführung dementsprechend. Dies fördert die Usability und kann beim Design der Software zur Entwicklung neuer Features anregen. Um sich in den User besser hineinversetzen zu können und um das Interface für die Zielgruppe optimal entwickeln zu können empfiehlt es sich fiktive User zu erfinden. Jede einzelne fiktive Person sollte so detailliert wie möglich ausgearbeitet werden.

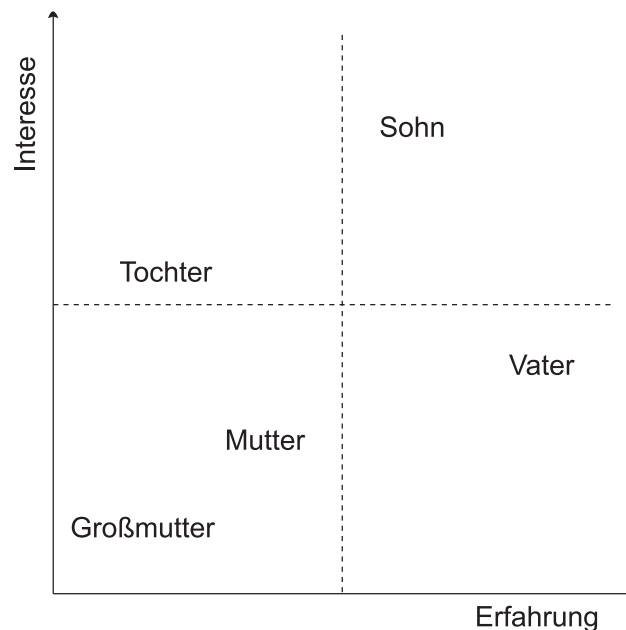
Im Rahmen dieser Diplomarbeit wurde der Entwicklungsprozess nicht schriftlich dokumentiert. Aber einige Ideen der verschiedenen Methodiken konnten erfolgreich angewandt werden. Beispielsweise wurden fiktive Personen erschaffen um die Anwendung aus der Sicht der Zielgruppe wahrnehmen zu können.

Es handelt sich bei den fiktiven Personen um eine moderne Familie, welche in einem Haushalt in einer mittelgroßen österreichischen Stadt lebt. Sie besteht aus dem Vater, der Mutter, zwei Kindern und der Großmutter. Abb. 2.3 zeigt eine Matrix, die für jedes Familienmitglied das Interesse an und die Erfahrung mit Technik grafisch darstellt.

- Der Vater ist 51 und Leiter der Personalabteilung in einem Großunternehmen. Er ist seit den ersten Tagen des Personal Computers an dessen Entwicklung und allen damit verbundenen Trends interessiert. Sein Beruf macht es notwendig einen PDA zur Koordinierung aller Termine zu nutzen und zusätzlich muss er ständig via Mobiltelefon erreichbar sein. Seit einigen Jahren verfolgt er die neuen Entwicklungen nur noch halbherzig und zeigt nur noch wenig Interesse an den neuesten Trends. Er ist sehr versiert im Umgang mit etablierten Standards, hat aber mittlerweile Schwierigkeiten sich an Neuerungen anzupassen.
- Die Mutter ist Angestellte in einer Handelskette und um zehn Jahre jünger als ihr Mann. Sie zeigte noch nie Interesse an technischen Dingen, ist aber gegenüber allen Dingen, die Wellness und Gesundheit betreffen, sehr aufgeschlossen. Sie verwendet so gut wie nie den Computer und das Mobiltelefon nur zum Telefonieren.
- Der Sohn ist 16 und steht kurz vor dem Beginn des Führerscheinkurses. Er besucht das Gymnasium und spielt am PC gerne Spiele. An der Schule lernt er den Umgang mit dem Betriebssystem und Office-Produkten. Am liebsten benutzt er seinen Computer aber zum Spielen von Action-Games. Er besitzt ein Wertkartenhandy und verschickt am liebsten SMS damit.
- Die Tochter besucht seit einem Jahr den örtlichen Kindergarten. Das einzige technische Gerät, das sie bisweilen alleine bedienen kann, ist

der Fernseher. Aber auch den bedient sie nur am Wochenende, während alle anderen ausschlafen.

- Die rüstige Großmutter ist schon seit langem in Pension. Sie hat keinen Bezug zur Technik und ist noch ohne elektronische Medien aufgewachsen. Wäre sie noch einmal jung, würde sie gerne den Umgang mit allen modernen Apparaten erlernen, aber sie fühlt sich jetzt schon zu alt dafür und unternimmt keinen Versuch mehr sich mit der Technik auseinanderzusetzen.



**Abbildung 2.3:** Grafische Darstellung von Interesse an und Erfahrung mit Technik der einzelnen Familienmitglieder.

Ein weiteres interessantes Instrument zur Verbesserung des Produktes, welches am Anfang oder Ende des Entwicklungsprozesses eingesetzt wird, ist das *Expertenreview* [16, S. 160ff]. Expertenreview-Teilnehmer sind ausschließlich Fachleute im Bereich der Applikation oder im Bereich von Benutzerschnittstellen, welche dem Designer einen formalen Bericht präsentieren. Wichtig dabei ist, dass die Experten Kritik geben, aber die Lösung von möglichen Problemen ausschließlich den Designern vorbehalten ist. Es gibt zahlreiche unterschiedliche Methoden für Expertenreviews:

- Heuristische Evaluation
- Richtlinien-Review

- Konsistenzinspektion
- Kognitiver Durchgang
- Formale Anwendbarkeitsinspektion

Der Kern eines Expertenreviews ist immer gleich: die Experten kontrollieren das Interface auf die Einhaltung von vorgegebenen Richtlinien oder anderen bestimmten Vorgaben, oder der Experte spielt die Rolle eines typischen Anwenders in einer realitätsnahen Umgebung.

*Usability-Tests* [16, S. 163ff] erfreuen sich bei Designern ebenfalls großer Beliebtheit. Mittlerweile haben einige Softwarehäuser bereits eigens dafür eingerichtete Usability-Labors. Diese Labors dienen einzig und alleine der Durchführung von Usability-Tests. Eine Testperson muss versuchen unter Aufsicht eines Beobachters an seiner Seite vorgegebene Aufgaben zu bewältigen. Der Beobachter hat keine aktive Rolle – er soll sich lediglich auf das Festhalten seiner Beobachtungen beim Anwender beschränken. Es gibt auch Labors, die über einseitig verspiegelte Wände verfügen, damit ein zusätzliches Beobachtungs-Team gemeinsam die Aktionen und das Verhalten der Testperson mitverfolgen kann. Teilweise wird auch Protokollierungssoftware, welche alle Aktionen des Benutzers protokolliert, eingesetzt. Manche Labors verfügen auch über Audioaufzeichnungsgeräte um alle Aussagen der Testperson und eventuelle laute Gedanken mitzuprotokollieren und in die Auswertung miteinbeziehen zu können. Videoaufzeichnungsgeräte können ebenfalls verwendet werden um alle Gestiken der Testperson aufzuzeichnen. Aber ein Usability-Test muss nicht in einem High-End-Labor durchgeführt werden. Eine Art des Usability-Tests, welche kostengünstig und zu jedem Zeitpunkt innerhalb des Entwicklungsprozesses durchgeführt werden kann, ist ein Test mit einem Papiermodell. Die Testperson muss ebenfalls vordefinierte Aufgaben bewältigen, bedient aber nicht den Computer, sondern ein Test-Administrator blättert im Papiermodell an die richtige Stelle. Usability-Tests sind sehr produktiv und liefern sehr gutes Feedback.

„Trotz aller Erfolge haben Usability-Tests zumindest zwei ernstzunehmende Einschränkungen: sie legen den Schwerpunkt auf die Erstbenutzung und decken die Vielfalt der Schnittstelleneigenschaften nur begrenzt ab. Da Usability-Tests in der Regel zwei bis vier Stunden dauern, wird es schwierig festzustellen, wie die Performance nach einer Woche oder einem Monat regelmäßiger Benutzung sein wird. Innerhalb der typischen zwei bis vier Stunden eines Usability-Tests werden die Teilnehmer wahrscheinlich nur einen kleinen Bruchteil der Features, Menüs, Dialogboxen oder Hilfe-Screens kennen lernen. Diese und andere Bedenken haben Design-Teams dazu gebracht, Usability-

Tests mit den verschiedenen Formen der Expertenreviews zu ergänzen [16, S. 168].“

## 2.4 Positiv- und Negativbeispiele

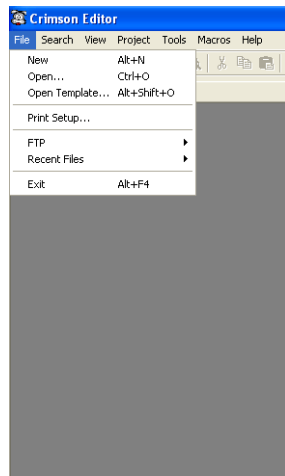
Gut umgesetzte User Interfaces helfen dem User schneller an sein Ziel zu kommen. Sie helfen dem User weniger Fehler beim Ausführen seiner Aufgabe zu machen und erlauben es so dem User sich besser auf die eigentliche Aufgabe zu konzentrieren. Auch Kleinigkeiten im Design können große Auswirkungen haben. Tabelle 2.3 veranschaulicht die Auswirkung von Designfehlern sehr gut:

<i>ADDITIONAL SECONDS REQUIRED PER SCREEN IN SECONDS</i>	<i>ADDITIONAL PERSON-YEARS REQUIRED TO PROCESS 4.8 MILLION SCREENS PER YEAR</i>
1	.7
5	3.6
10	7.1
20	14.2

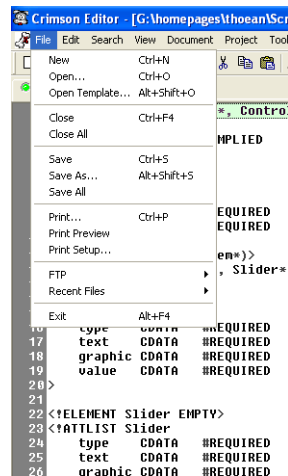
**Tabelle 2.3:** Beispiel aus [7, S. 5] zur Veranschaulichung der Auswirkung von Designfehlern.

### 2.4.1 Dynamische/statische Menüleiste

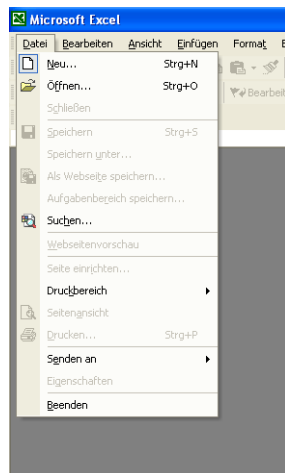
Das erste konkrete Beispiel beschäftigt sich mit der Menüleiste. Zahlreiche Applikationen verfügen über dynamische Menüs [8, S. 62ff]. Dh.: Das Menü verändert sich während der Laufzeit des Programms. Sinnvoll sind dynamische Menüs dann, wenn die Applikation über zusätzliche Features (zB Plugins) verfügt, die in der Basisversion nicht vorhanden sind. Durch Erweiterungen ist es oftmals nicht möglich die Menüstruktur von vornherein festzulegen und die Anzeige aller zukünftigen Gimmicks im Menü fix zu integrieren. Nicht sinnvoll ist die dynamische Veränderung des Menüs dann, wenn das Menüelement nur aus limitierten Befehlen besteht und eine Erweiterung zu einem späteren Zeitpunkt nicht stattfindet. In Abb. 2.4 zeigen dies die Bilder (a) und (b) anhand des Crimson Editors. Eine bessere Lösung ist es die gerade nicht möglichen Menübefehle durch spezielle Kennzeichnung zu deaktivieren (Abb. 2.4 (c) und (d)). Der User kann so Veränderungen in einem Menü schneller wahrnehmen und besser eine Abhängigkeit von einem bestimmten Zustand feststellen.



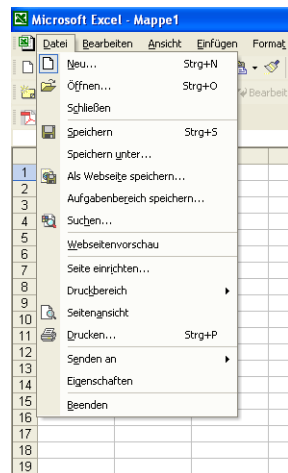
(a)



(b)



(c)



(d)

**Abbildung 2.4:** Menüleisten Crimson Editor und Microsoft Excel. (a) Menü von Crimson Editor ohne geöffnetem Dokument, (b) Menü von Crimson Editor bei geöffnetem Dokument, (c) Menü von Microsoft Excel ohne geöffnetem, (d) Menü von Microsoft Excel bei geöffnetem Dokument.

Jede zusätzliche Option, die ein Programm einem User zur Verfügung stellt, verlangt vom User eine Entscheidung. Es empfiehlt sich als Programmierer eine Vorauswahl an Einstellungen zu treffen ohne dem User bei der Installation oder Nutzung eines Programmes einen regelrechten Optionenmarathon aufzuzwingen. Natürlich soll der User zu einem späteren Zeitpunkt die Möglichkeit haben Änderungen an den Einstellungen vorzu-

nehmen. Erschwerend zur Entscheidungsfindung kann hinzukommen, dass der Benutzer eventuell nicht den benötigten Wissensstand hat um eine entsprechende Entscheidung zu treffen [8, S. 37ff, 260ff].

#### 2.4.2 Unentschlossener Programmierstil

Ein Beispiel für die Auswirkung eines unentschlossenen Programmierstils bietet uns Microsofts „Find Setup Wizard“ (wie auch in [17, S. 15ff] beschrieben). Zahlreiche Programme von Microsoft hatten (und haben zum Teil noch immer) die Angewohnheit einen User, welcher die Hilfefunktion des Programmes aufruft, mit dem Dialog, welcher in Abb. 2.5 zu sehen ist, zu belästigen. Der User versucht eine Aufgabe zu bewältigen und gelangt dabei an die Grenzen seiner Kenntnisse über das Programm. Er entscheidet sich dazu Rat in der Hilfefunktion des Programmes zu suchen und muss mit Erstaunen feststellen, dass sich eben genannter Dialog öffnet. Der User ist ohnehin in der Ausführung seiner Tätigkeit gestört und sucht nach Hilfe. Anstatt mit der Suche nach Hilfe beginnen zu können nötigt ihn die Hilfefunktion dazu sich auf ein neues Problem zu konzentrieren. Er soll sich dazu entschließen eine Entscheidung zu fällen, in welcher Weise die Suchfunktion optimiert werden soll. Nicht nur, dass der unentschlossene Programmierer den User von seiner ursprünglichen Suche ablenkt - nein - er zwingt ihm einen Optionsdialog auf, bei welchem anscheinend ohnehin standardmäßig eine Variante den anderen vorzuziehen ist. Eine gute Möglichkeit dies zu vermeiden wäre gewesen, eine Standardeinstellung zu wählen, und dem User aber die Möglichkeit zu lassen die Einstellung wann immer er möchte in den Systemeinstellungen zu ändern.

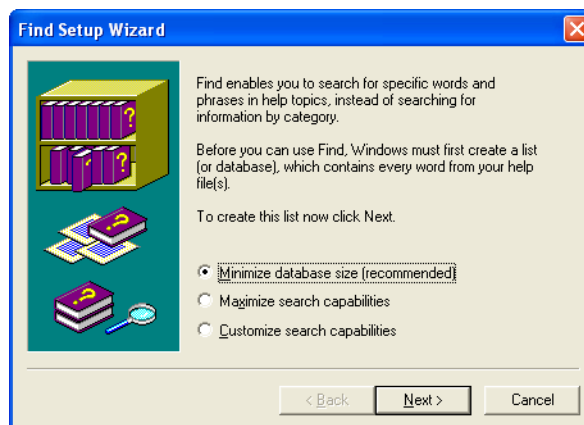


Abbildung 2.5: Microsoft Find Setup Wizard.

### 2.4.3 Alternatives Layout

Wie in Kapitel 1.3 erwähnt, ist es wünschenswert dem User verschiedene Layouts für die Applikation zur Verfügung zu stellen. Eine Möglichkeit eine Applikation wandelbar wie ein Chamäleon zu machen ist die *Skin*-Technologie. Hierbei wird das grafische Interface der Applikation vollkommen von der Logik der Applikation extrahiert und durch definierte Schnittstellen angekoppelt. Verschiedene *Skins* (Layout, Aussehen) können nachträglich installiert und ausgewählt werden. Abb. 2.6 zeigt Skins für Microsoft Mediaplayer 9. Der Wechsel des Aussehens muss nicht immer so drastisch wie in der Abbildung sein. Alternative Darstellungen können sich auch auf einzelne Dinge beschränken. Man könnte zB den User nur die Farbgebung oder die Schriftgröße auswählen lassen, aber das Gesamtbild nicht verändern.



**Abbildung 2.6:** Microsoft Mediaplayer 9. Nicht alle Skins sind auf Usability getrimmt.

### 2.4.4 Verschachtelte Dialog-/Fensterhierarchien

Eine wahre Boshaftigkeit dem User gegenüber zeigt Abb. 2.7. Zu verschachtelte Dialog- oder Fensterhierarchien sollten vermieden werden [8, S. 309ff]. Die Abbildung zeigt den Versuch beim Betriebssystem Microsoft Windows XP den Treiber einer Grafikkarte upzudaten. Fairerweise muss

gesagt werden, dass Windows XP auch schnellere Wege zur selben Funktion bietet, jedoch ist auch die redundante Integration von Befehlen in ein Interface nicht positiv. Ein Grund, warum es zu solchen Verschachtelungen kommen kann, ist, dass der Entwickler zwar sehr viel Zeit mit der Konzeption einzelner Fenster verbracht hat, aber bereits den Blick für das System als Ganzes verloren hat. Das Problem kann aber auch von einer mangelhaften Gesamtkonzeption herrühren und zeigt auf wie wichtig die Designphase - die Phase welcher der Implementierung vorausgeht - wirklich ist.

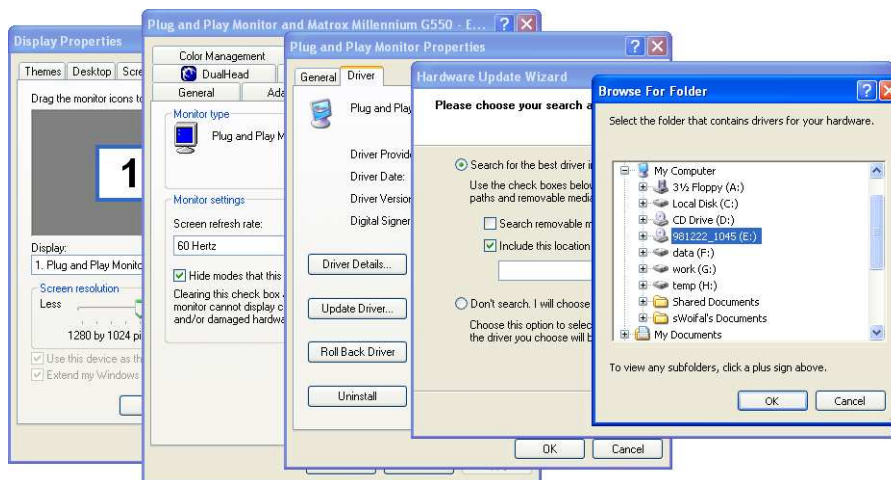


Abbildung 2.7: Microsoft Windows XP - Dialogfenster-Overkill.

### 2.4.5 Speaking Geek

Abb. 2.8 zeigt ebenfalls ein Negativbeispiel. Was in [8, S. 203ff] unter dem Begriff *Speaking Geek* zusammengefasst wird, ist eine dem User unverständliche Ausdrucksweise. Oftmals begegnet der User kryptischen Fehlermeldungen wie in der Abb. 2.8, oder die Namensgebung von Befehlen entspricht eher dem Griechischen als der Sprache des Users. Der Vorteil, wenn es sich um Griechisch und nicht um Programmiererlatein handelt, ist, dass der User wahrscheinlich einfach eine andere Sprachversion herunterladen kann. Übersetzungen in der selben Sprache, jedoch mit einer anderen Ausdrucksweise, sind leider eher die Ausnahme. Dies wäre eine einfache Methode um ein Programm verschiedenen Usergruppen anzupassen. Allerdings ist eine bloße Änderung der Ausdrucksweise nicht ausreichend um einer anderen Benutzergruppe (zB Anfänger, Erfahrene) gerecht zu werden. Besonders bei der Fehlerbehandlung sollte textuellen Ausgaben Beachtung geschenkt werden. Die Fehlermeldung kann ohne weiteres Hinweise auf Codes, welche dem Support-Team oder dem Programmierer Hinweise liefern, enthalten,

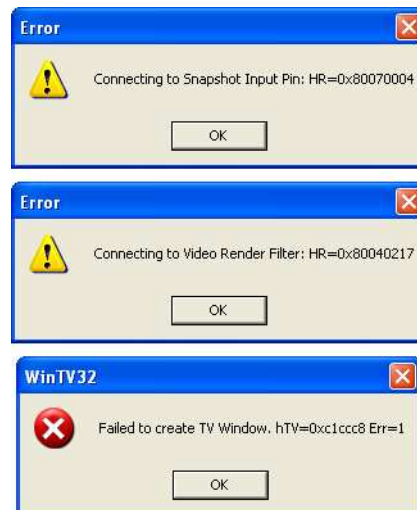
jedoch sollte auch der User über die Ursache des Fehlers informiert werden. Unter Umständen ist dann der User selbst in der Lage die Fehlerquelle zu eliminieren und spart so unnötigen Zeitaufwand und Kosten.



**Abbildung 2.8:** „Speak Geek“: Programmiererlatein aus dem Programm DScaler. Und wo genau liegt jetzt der Fehler?

Bei Versuchen Fehlermeldungen absichtlich hervorzurufen konnte ein weiteres Negativ- und ein gutes Positiv-Beispiel gefunden werden. Um eine Fehlermeldung zu provozieren wurden hintereinander zwei Programme gestartet, welche ein und die selbe Hardware-Ressource verwenden. Allerdings erlaubt die Hardware-Ressource nur den Zugriff einer einzigen Applikation zur gleichen Zeit. Es wurden die Programme *VirtualDub* und *Hauppauge WinTV 2000* herangezogen, welche beide auf die vorhandene Hauppauge TV-Karte Zugriff erlangen sollten.

Beim ersten Versuch wurde zuerst *VirtualDub* und dann *WinTV* gestartet. Dadurch wurde die Fehlerbehandlungs-Routine der Applikation *WinTV* aktiv. Diese zeigte lediglich Fehlermeldungen, welche keinen Rückschluss auf die Ursache des Problems zulassen (siehe Abb. 2.9).



**Abbildung 2.9:** Fehlerbehandlungs-Routine der Applikation Hauppauge WinTV 2000.

Wurde *WinTV* zuerst gestartet und so eine Fehlermeldung bei *VirtualDub* provoziert, so lieferte *VirtualDub* zuallererst eine aussagekräftige Fehlermeldung. Anschließend stellte das Programm ein Auswahl-Dialogfenster zur Auswahl einer alternativen Hardware-Ressource und lieferte bei der wiederholten Auswahl der selben Ressource erneut drei Fehlermeldungen. Diese waren zwar nicht so aussagekräftig wie die erste, jedoch liegt die Herkunft dieser Fehlermeldungen auf der Hand, da der Rückschluss auf die erste Meldung gezogen werden kann. Die einzelnen Fenster zeigt Abb. 2.10.

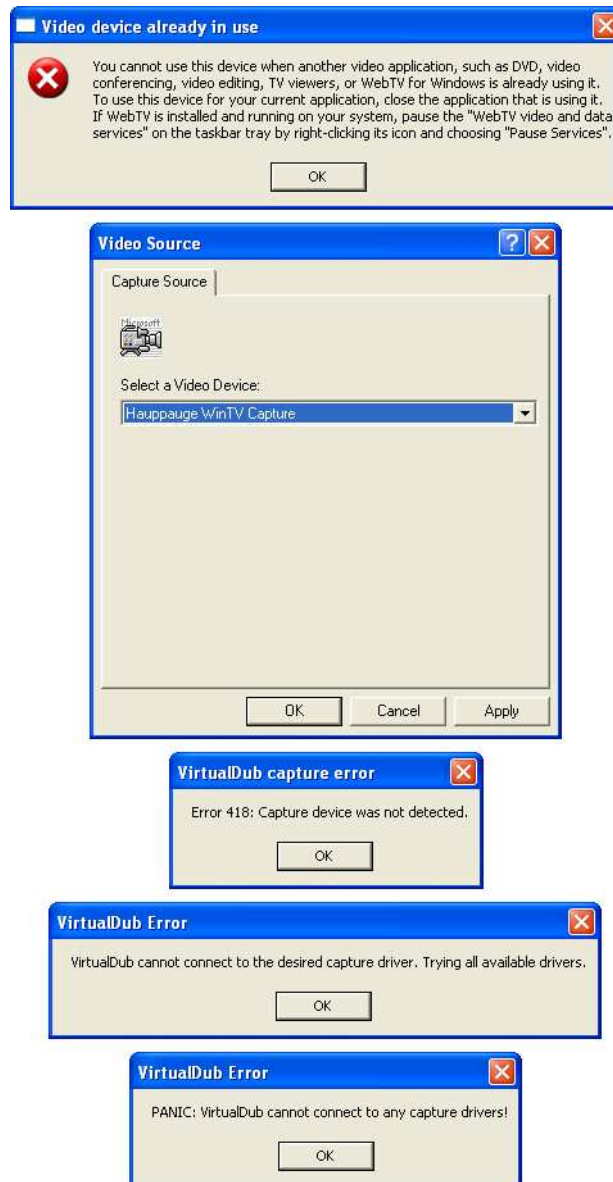


Abbildung 2.10: Fehlerbehandlungs-Routine der Applikation VirtualDub.

# Kapitel 3

## Konzept

### 3.1 Spezifiziertes Szenario

Ausgehend von der Grundidee - Dienste und Geräte des modernen Alltags mittels unterschiedlicher Eingabegeräte fernzusteuern und zu nutzen - wurde das umzusetzende Szenario im Detail formuliert. Mögliche Realisierungsszenarien wurden ausgearbeitet und zahlreiche Einsatzmöglichkeiten besprochen.

Die schier unbegrenzte Anzahl an möglichen Eingabegeräten, steuerbaren Haushaltsgeräten und verschiedensten Informationsdiensten machte das Einschränken des technisch Machbaren auf ein überschaubares Diplomarbeitsszenario zu keiner leichten Aufgabe.

Im Wesentlichen finden sich in modernen Haushalten drei große Gruppen von Eingabegeräten, die für das Diplomarbeitsszenario interessant sind.

1. Die erste Gruppe stellen die *Personal Computer* dar. Darunter fallen alle möglichen Varianten, zB herkömmliche Desktop PCs, handelsübliches Notebooks, sowie „state of the art“ Tablet PCs.
2. Die zweite große Gruppe ist die Gruppe der *Personal Digital Assistents*. Hier ist die Variantenvielfalt im Vergleich zur ersten Gruppe nicht so extrem ausgebildet. Die große Masse der Geräte verfügt über ähnliche Leistungsmerkmale und unterscheidet sich nur in kleinen Details.
3. Die dritte Gruppe ist die Gruppe der *Mobiltelefone*. Hier ist die Palette an Geräten wieder größer und eine sinnvolle Klassifizierung in Teilgruppen fällt schwer, da es keine klare Abgrenzung zwischen den einzelnen Geräten gibt. In jeder Evolutionsstufe vom Minimal-GSM-Mobiltelefon bis zum High-Tech-UMTS-Gerät finden sich zahlreiche Produkte.

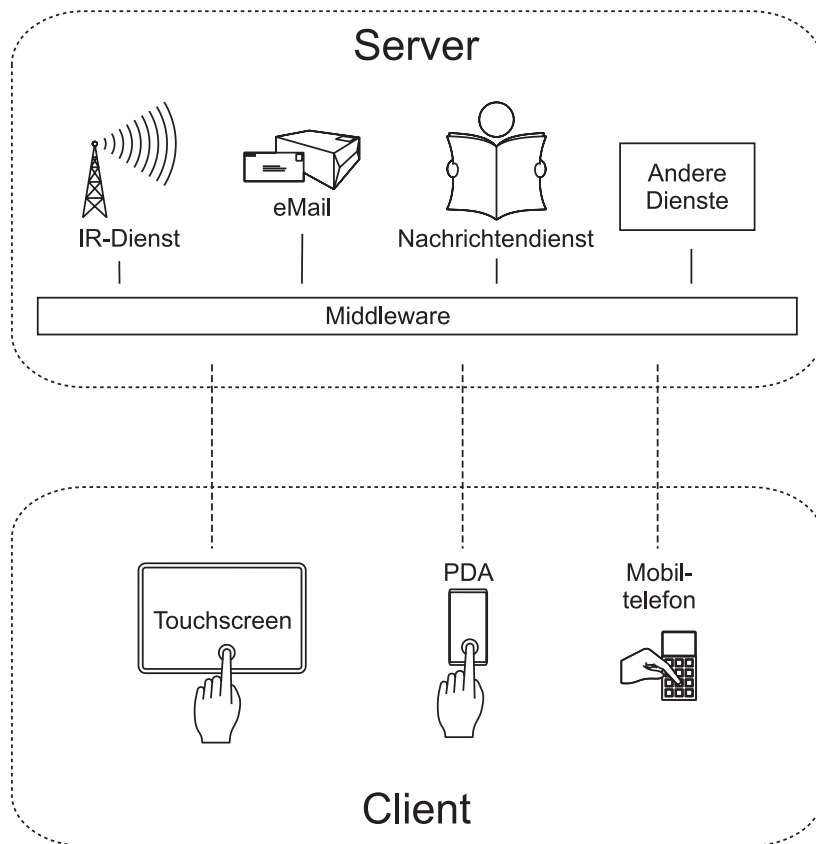


Abbildung 3.1: Mögliches Client/Server Szenario

### 3.1.1 Oberflächen

Das spezifizierte Diplomarbeitsszenario beschränkt sich auf zwei zu entwickelnde Benutzeroberflächen.

Eine Oberfläche soll auf der ersten Gruppe der Geräte aufsetzen und per Touchscreen bedient werden. Daraus ergibt sich die Erfordernis, dass das Interface ausschließlich mittels „Standard Klicks“, dh. Klicks, die der linken Maustaste entsprechen - worauf der Touchscreen limitiert ist - bedient werden kann. In dieser Gruppe ist die Zielgruppe zur Benutzung der Oberfläche sehr breit gefächert. Sie reicht vom Kleinkind, welches noch nicht lesen und schreiben kann, bis hin zum Erwachsenen im hohen Alter. Auch diese Erfordernis stellt hohe Ansprüche an das Applikationsdesign.

Die zweite Oberfläche soll auf der zweiten Gruppe - der Gruppe der PDAs - laufen. Eine Einschränkung der Eingabemöglichkeit ist hier - im Gegensatz zur ersten Oberfläche - nicht zu treffen. Der große Unterschied zur ersten Gruppe ist in der wesentlich kleineren Displaygröße gegeben.

Bei allen Eingabegerätegruppen soll es möglich sein spezifizierte Basis-

funktionen für dem System unbekannte Benutzer zur Verfügung zu stellen und die einzelnen Benutzer mit personalisierten und passwortgeschützten Steuerungsmenüs zu versorgen.

### 3.1.2 Geräte und Dienste

Die Palette an bedien- und nutzbaren Geräten und Diensten ist schier unendlich. Folgende Elemente wurden zur Umsetzung im Rahmen der Diplomarbeit bestimmt:

1. *Die Steuerung und Interaktion einem Building-Control-Moduls.* Dadurch wird es möglich unterschiedliche Medien-Renderer anzusteuern (zB Videorekorder, DVD-Player, Stereoanlage, etc.) und zB Playlisten remote auf den Geräten abzuspielen. Dieses Modul stellt auch die Verbindung zum European Instabus her und erweitert das System in einem sehr hohen Ausmaß. Das EIB System ist bereits in der Lage sehr viele Geräte zu verwalten und Dienste zur Verfügung zu stellen. Beispielsweise könnten so zB Lichter, die Heizung und Jalousien gesteuert werden.
2. *Die Anzeige von Online-Informationskanälen im RSS/RDF-Format.* Der Benutzer kann aus einem Pool von Online-Informationsangeboten auswählen und neue Angebote hinzufügen. Informationsdienste sehen so aus, dass ein RSS/RDF-Dokument einen Kanal darstellt. Dieser beinhaltet unterschiedliche Artikel, welche eine Überschrift, eine Beschreibung, ein Bild und einen Link zu mehr Informationen beinhalten können.
3. *Die Einbindung eines Kompass/GPS-Moduls.* Durch das Auslesen der GPS/Kompass-Daten am Server kann so der Aufenthalt von im Kompass/GPS-Modul registrierten Sendern ermittelt und an den Benutzer geschickt werden. Mögliche Anwendungen wären zB das Anbringen eines Senders am Wagen oder in der Schultasche der Kinder.
4. *Die Abfrage von erhaltenen Emails von IMAP Emailkonten.* Der eingeloggte Benutzer kann beliebige IMAP Emailkonten spezifizieren und kann sich diese über die Oberfläche ansehen.

Da die Möglichkeiten wie schon erwähnt schier unbegrenzt sind, war eine Beschränkung der Diplomarbeit auf ein in der Kürze der Diplomarbeit umsetzbares Szenario notwendig. Die eben erwähnten Dienste stellen die gewünschte Funktionalität der Diplomarbeit dar. Im Nachfolgenden werden weitere angedachte Erweiterungsmöglichkeiten geschildert:

1. Die Anzeige eines Videostreams von einer Überwachungskamera. Die Einbindung einer Kamera in das System kann für verschiedenste Aufgaben genutzt werden. Die Möglichkeiten reichen von der Kamera im

Bereich der Eingangstür bis zur Überwachung von Bereichen, die eine besondere Aufmerksamkeit erfordern – und sei es nur der videoüberwachte Küchenherd.

2. Die Steuerung von Geräten des Haushalts mittels Infrarotverbindung (zB Fernsehgerät, Stereoanlage, DVD-Player, etc.). Dieses Modul soll wie eine mittlerweile sogar von Lebensmitteldiscountern vertriebene und bereits überall bekannte All-In-One Infrarotfernsteuerung funktionieren.
3. Die Umsetzung eines Multimediasystems, welches über einen zentralen Fileserver Audio- und Videostücke zur Verfügung stellt.
4. Die Integration von einem Interface zur Sprachtelefonie.
5. Die Erweiterung des Systems um eine Möglichkeit, von Netzwerken außerhalb des Systems mit dem System zu kommunizieren, wie zB ein Chat- oder Instant-Messaging-Modul.
6. Das Bereitstellen eines Verfügbarkeitsstatus von Bewohnern des Haushaltes für Webuser.

Nach der Einigung auf ein für beide Diplomanden erstrebenswertes Szenario wurde das Gesamtprojekt auf zwei voneinander getrennte Diplomarbeiten aufgeteilt.

Bei der Abgrenzung wurde festgelegt, dass sich diese Diplomarbeit ausschließlich mit der Benutzerschnittstelle befasst und die andere Diplomarbeit die Middleware zur Steuerung und Nutzung der Dienste und Geräte bereitstellt.

Die einzige Überschneidung der beiden Diplomarbeiten ist der spezifizierte Kommunikationsweg zwischen Benutzeroberfläche und Middleware.

## 3.2 Grundkonzept

Das Grundkonzept sieht vor, ausschließlich freie Technologien zu verwenden. Die Kommunikation zwischen Client und Server basiert auf dem HTTP-Protokoll. Der Client sendet einen HTTP-GET-Request an den Server und erhält als Response eine XML-Datenstruktur. Diese Datenstruktur enthält alle Informationen, die es dem Client ermöglichen alle nötigen Elemente am Display darzustellen, damit der User mit dem Server interagieren kann.

Die Wahl des Protokolls und der Datenbeschreibungssprache war einfach. Beide Standards sind längst auf allen Plattformen etabliert und äußerst stabil. Zahlreiche komfortable HTTP-Funktionalitäten sind bereits für alle gängigen Programmiersprachen in teils integrierten, teils externen APIs verfügbar. So finden sich für jedes Betriebssystem und jede Programmiersprache APIs, welche HTTP zur Verfügung stellen - unabhängig vom Gerät.

Die Palette reicht vom DesktopPC über PDAs bis zum kleinen Mobiltelefon. DesktopPCs und PDAs unterstützen meist verschiedene Entwicklungsmöglichkeiten, jedoch auch das im Vergleich zu den anderen Gerätetypen sehr eingeschränkte System bietet mit dem MIDP (siehe Kapitel 4.4) bereits ab Version 1.0 - sofern das Mobiltelefon Mobile Java<sup>1</sup> unterstützt - eine komfortable API. Auch jeder Standard-Webbrowser beherrscht das HTTP-Protokoll. Dieser Umstand ist vor allem in Verbindung mit XML interessant, da bereits zahlreiche Browser XSL-Transformationen anwenden können und so eine einfache Transformation von XML-Datenstrukturen in andere Strukturen zulassen.

Die Spezifizierung des *Hypertext Transfer Protocol (HTTP)* lautet wie folgt:

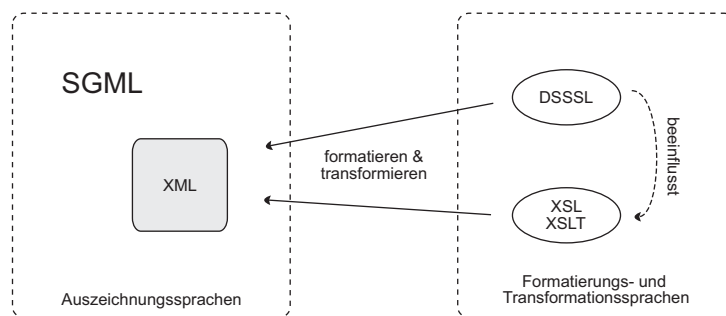
„The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, protocol which can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems, through extension of its request methods, error codes and headers [..]. A feature of HTTP is the typing and negotiation of data representation, allowing systems to be built independently of the data being transferred. [13]“

Das Protokoll unterstützt verschiedene Request-Methoden, wobei sich das Konzept auf die GET-Methode stützt. Hierbei werden allfällige zu übermittelnde Variablen in die Request-URL codiert und so transportiert. Weiters werden bei jedem HTTP-Request Headerinformationen übermittelt. Diese können vor dem Request manipuliert werden und zB Daten zum User-Agent, welcher den Request auslöst, Betriebssysteminformationen, etc. enthalten.

Die *Extensible Markup Language (XML)* (Abb. 3.2) ist eine Dokument-Beschreibungssprache und stellt eine Teilmenge von SGML dar. Auf beide Sprachen sind DSSSL-Stylesheets (Document Style Semantics and Specification Language) anwendbar. XML-Dokumente bestehen aus dem reinen Markup, unterliegen aber einer formalen Beschreibung der Datenstruktur - der Dokumenttyp-Definition (DTD). Die DTD beschreibt Aufbau und Strukturierung eines Dokumentes, nicht die Formatierung. XSL ist die Stilsprache für XML. Sie stellt im Wesentlichen zwei Eigenschaften zur Verfügung: Formatierobjekte (XSL-FO) und Transformationen (XSLT). XSL-FO erlauben es zum Beispiel aus XML-Dokumenten PDF-Dokumente zu erzeugen. XSLT ermöglicht die Umwandlung von XML-Dokumenten in eine andere Struktur (zB in eine HTML-Struktur) [1].

---

<sup>1</sup><http://wireless.java.sun.com/>



**Abbildung 3.2:** Grafische Darstellung der Extensible Markup Language (XML)

Das Konzept des Systems sieht so aus, dass die komplette Logik und auch alle Gestaltungselemente am Server liegen. Der Client ist lediglich in der Lage die XML-Datenstruktur auszuwerten und in ein Interface umzuwandeln. Der Aufbau und die Inhalte der Struktur, welche in der Diplomarbeit von zentraler Bedeutung ist, sind detailliert in Kapitel 3.3 beschrieben.

Ein zweiter zentraler Baustein bei der Entwicklung ist die Entwicklung und Implementierung einer Alternative zu herkömmlichen Passwordeingaben. Das Ziel war es eine visuelle Passwordeingabe, welche keine Tastatur als Eingabehilfe benötigt, zu erschaffen. Leider wurde diese Methode der Passwordeingabe im Rahmen dieser Diplomarbeit nicht neu erfunden. Das Patent („*Graphical passwords for use in a data processing network.*“ [6]) wurde bereits 2001 eingereicht und 2003 veröffentlicht. Es konnte allerdings keine Anwendung recherchiert werden, welche sich genau dieser Methode der Authentifizierung bedient. Auf <http://www.onetouchpass.com/> findet sich eine andere Alternative für visuelle Passwörter. Dieses System ist speziell für PDAs mit PalmOS<sup>2</sup> oder Windows PocketPC<sup>3</sup>-Betriebssystem zugeschnitten. Der Vorteil dieser Methode, welcher für die Verwendung in der Diplomarbeit spricht, ist, dass diese Art der Passwordeingabe bereits von Kleinkindern durchgeführt werden kann, da sie über keinerlei Wissen über die Tastatur verfügen müssen. Personen, die im Umgang mit dem PC nicht so versiert sind, dürften diese Methode ebenfalls begrüßen. Außerdem ist es dem Menschen leichter möglich sich an visuelle Passwörter zu erinnern als an textuelle Passwörter [5]. Die genaue Funktionsweise der Passwordeingabe wird in Kapitel 3.4 erläutert.

<sup>2</sup><http://www.palm.com/>

<sup>3</sup><http://www.pocketpc.com/>

### 3.3 Client/Server-Kommunikation

Die Client/Server-Kommunikation war eine der zentralen Aufgaben dieser Diplomarbeit. Besonders das zu entwickelnde Datenformat, welches zur Generierung der Benutzeroberfläche verwendet wird, spielte eine große Rolle. Die entwickelte XML-Struktur beinhaltet die Daten, welche der Client auswertet, um anschließend die Oberfläche am Ausgabegerät darzustellen. Damit der Client eine XML-Datei erhält, sendet er einen Request an den Server. Lediglich die initiale URL muss am Client konfiguriert werden. Alle darauf folgenden URL-Aufrufe werden durch den Server selbst gesteuert, indem er Aktionselemente mit entsprechenden URLs in der XML-Struktur definiert. Durch das Setzen der Header-Information *User-Agent* beim Client-Request auf einen beim Server registrierten Schlüsselwert kann der Server feststellen, welche Art von Client den Request durchführt. Ist diese Information kein vordefinierter Schlüsselwert, so gibt der Server standardmäßig eine zur Darstellung in Browsern optimierte XML-Struktur zurück. Der Client parst die XML-Struktur und erzeugt die Oberfläche. Der Server stellt ebenfalls alle benötigten Grafiken und Sounds zur Verfügung. Dadurch ist eine Änderung der Programmlogik oder des Erscheinungsbildes nur am Server durchzuführen. Eine Anpassung der in Verwendung befindlichen Clients an Neuerungen des Komplettsystems ist nach der Installation nie mehr erforderlich.

Die Datenstruktur wurde so definiert, dass alle nötigen Elemente beschrieben wurden, gleichzeitig wurde eine Möglichkeit zur einfachen, problemlosen Erweiterung der bestehenden Struktur in der Zukunft berücksichtigt. Im Grunde konnten alle möglichen darzustellenden Oberflächen in zwei grundlegende Typen unterteilt werden. Zwar stellt das Interface immer nur eine Reihe einfacher Elemente, welche der Interaktion mit dem Benutzer dienen, dar, jedoch wurde ein Spezialfall als eigener Bildschirm-Typ herausgenommen. Der Spezialfall ist die Passworteingabe, welche im nächsten Kapitel erläutert wird. In allen anderen Fällen richtet sich die Anzeige der Inhalte nach nur einem einzigen entwickelten Schema: der Dokumenttyp-Definition (DTD) `Screen.dtd` (Abb. 3.3).

Ein `Screen` hat einen Namen und einen Typ. Der Typ signalisiert aus Effizienzgründen dem Client bereits vor der Auswertung der Daten um welche Art von Ausgabe es sich handelt. Es gibt drei Typen: `ClientConfig`, `Screen` und `Rss`. Die letzten beiden Typen bewirken grafische Ausgaben am Ausgabegerät. Der erste Typ ist ein Spezialfall. Er bedeutet, dass im XML-Dokument ausschließlich Konfigurations-Werte mitgeliefert werden. Im Anschluss an die Verarbeitung der Werte muss der Start-Screen der Anwendung geöffnet werden. Abb. 3.4 zeigt zeigt den Quelltext für ein XML-Dokument des Typs `ClientConfig`, welches sich der Dokumenttyp-Definition `Screen.dtd` unterwirft.

Der `Screen` setzt sich also aus beliebig vielen `Meta`- und je einem oder keinem `Controls`- und `Items`-Element zusammen. `Meta`-Elemente trans-

```

1 <!ELEMENT Screen (Meta*, Controls?, Items?) >
2 <!ATTLIST Screen
3     name      CDATA      #IMPLIED
4     type      CDATA      #REQUIRED
5 >
6
7 <!ELEMENT Meta EMPTY >
8 <!ATTLIST Meta
9     name      CDATA      #REQUIRED
10    value     CDATA      #REQUIRED
11 >
12 <!ELEMENT Controls (Item)+ >
13 <!ELEMENT Items (Item, Slider)+ >
14
15 <!ELEMENT Item EMPTY>
16 <!ATTLIST Item
17     type      CDATA      #REQUIRED
18     text      CDATA      #REQUIRED
19     graphic   CDATA      #REQUIRED
20     value     CDATA      #REQUIRED
21 >
22
23 <!ELEMENT Slider EMPTY >
24 <!ATTLIST Slider
25     text      CDATA      #REQUIRED
26     graphic   CDATA      #REQUIRED
27     initial   CDATA      #REQUIRED
28     minimum   CDATA      #REQUIRED
29     maximum   CDATA      #REQUIRED
30     increment CDATA      #REQUIRED
31     value     CDATA      #REQUIRED
32     cancelValue CDATA      #REQUIRED
33 >

```

Abbildung 3.3: Quelltext der Dokumenttyp-Definition *Screen.dtd*.

portieren Daten wie zB Textnachrichten, die ausgegeben werden sollen, und können auch für zukünftige Eigenentwicklungen herangezogen werden. Sie ermöglichen die Entwicklung neuer Features ohne eine Anpassung der XML-Strukturbeschreibung durchführen zu müssen. `Items` und `Controls` bestehen aus mindestens einem oder mehreren `Item`- und `Slider`-Elementen. Ein `Item`-Element verkörpert nichts anderes als ein Interaktionselement. Ein Interaktionselement kann als `type` zur Zeit folgende Werte aufweisen: *noAction*, *action*, *screenSaver*, *colorChooser* und *fontSizeChooser*. Die Schlüsselwörter *screenSaver*, *colorChooser* und *fontSizeChooser* rufen interne Client-Funktionalitäten auf. Diese Elemente bringen Screens zur Konfiguration des

```

1 <?xml version="1.0"?>
2 <Screen name="InitialScreen" type="ClientConfig">
3   <Meta name="multiScreenGraphic"
4     value="http://localhost/thoean/screen.svg" />
5   <Meta name="plusGraphic"
6     value="http://localhost/thoean/plus.svg" />
7   <Meta name="minusGraphic"
8     value="http://localhost/thoean/minus.svg" />
9   <Meta name="scrollUpGraphic"
10    value="http://localhost/thoean/scrollup.svg" />
11  <Meta name="scrollDownGraphic"
12    value="http://localhost/thoean/scrolldown.svg" />
13  <Meta name="okGraphic"
14    value="http://localhost/thoean/ok.svg" />
15  <Meta name="cancelGraphic"
16    value="http://localhost/thoean/cancel.svg" />
17  <Meta name="backGraphic"
18    value="http://localhost/thoean/back.svg" />
19  <Meta name="colorGraphic"
20    value="http://localhost/thoean/color.svg" />
21  <Meta name="fontGraphic"
22    value="http://localhost/font.svg" />
23  <Meta name="loginGraphic"
24    value="http://localhost/login.svg" />
25  <Meta name="screenSaverGraphic"
26    value="http://localhost/screenSaver.svg" />
27  <Meta name="exitGraphic"
28    value="http://localhost/exit.svg" />
29  <Meta name="clickSound"
30    value="http://localhost/sound.wav" />
31  <Meta name="logonPath"
32    value="http://localhost/userList.xml" />
33  <Meta name="User-Agent"
34    value="PCGUIDE" />
35  <Meta name="message.Loading"
36    value="Loading..." />
37  <Meta name="message.FontSizeChooserText"
38    value="The quick brown fox jumps over the box." />
39 </Screen>

```

**Abbildung 3.4:** Quelltext eines XML-Dokumentes zur Konfiguration des Clients.

Layouts des Clients zum Vorschein. Das Schlüsselwort *action* verwandelt das Element ebenfalls in einen Button, jedoch führt es in diesem Fall dazu, dass der Client die als *value* gesetzte URL requestet. Durch den Wert *noAction* wird das Element als reines Anzeige-Element deklariert. *Slider*-Elemente

sind eine andere Art von Interaktionselement als `Item`-Elemente. Sie können keine Client-internen Funktionen aufrufen, sondern lediglich URLs am Server. Sie funktionieren ähnlich einem Potentiometer. Es gibt einen Startwert und einen Endwert, wobei man sich vom Initialwert in definierten Schrittwerten dem Start- und Endwert nähern kann.

Am Beispiel des Client für Personal Computer werden zB `Controls` immer am unteren Rand des Bildschirms eingeblendet. `Items` befüllt den restlichen vorhandenen Platz am Bildschirm. In anderen Clients könnte die Aufteilung anders aussehen - beide Elemente verkörpern nur eine logische Trennung.

### 3.4 Grafische Passworteingabe

Ein weiterer zentraler Bestandteil dieser Diplomarbeit ist die Entwicklung und Umsetzung einer grafischen Passworteingabe. Die entwickelte Form der grafischen Passworteingabe erfüllt zwei - für das komplette System wichtige - Anforderungen. Sie ist in einem wesentlich größeren Maße unabhängig von Alter und Erfahrung als herkömmliche Methoden zur Passworteingabe. Bereits Kinder und auch ältere Menschen, welche keine Erfahrung im Umgang mit Software haben, sind in der Lage diese alternative Eingabeform durchzuführen.

Die technische Funktionsweise ist folgende: am Bildschirm wird eine Anzeigefläche aus  $n \times m$  Zellen generiert. Jede Zelle des Grids, welche eine Grafik beinhaltet, fungiert als *Digit*. Das Passwort besteht aus so vielen *Digits*, wie es einzelne *Digit*-Elemente gibt. Jeder *Digit* verfügt über verschiedene *Values*. Ein *Value* aus allen verfügbaren *Values* des *Digits* kann dem *Digit* zugewiesen werden. Fügt man nun die selektierten *Values* der einzelnen *Digits* zusammen, so erhält man das endgültige Passwort. Abb. 3.5 veranschaulicht die Funktionsweise grafisch.

Praktisch (für den Benutzer) sieht der Vorgang wie in Abb. 3.6 aus. Der User sieht am Bildschirm das Grid mit einigen Grafiken darin. Jede Grafik kann durch eine Grafik aus der zugehörigen Menge an Alternativ-Grafiken ersetzt werden.

Das Root-Element *PasswordTheme* beschreibt den Namen und die repräsentative Grafik des Passworteingabe-Themas, die Größe des Grids und URLs zum Abbruch der Passworteingabe und zur Verifizierung des Passworts. Das *PasswordTheme* besitzt beliebig viele *Digit*-Elemente, wobei jedes Element seine Position im Grid, eine Initial-Grafik, seine Position im Passwort und einen Initial-Wert definiert. Jeder *Digit* kann beliebig viele *Value*-Elemente haben, welche die möglichen Alternativ-Grafiken mit zugehörigem Wert darstellen. Abb. 3.7 zeigt den Quelltext der DTD.

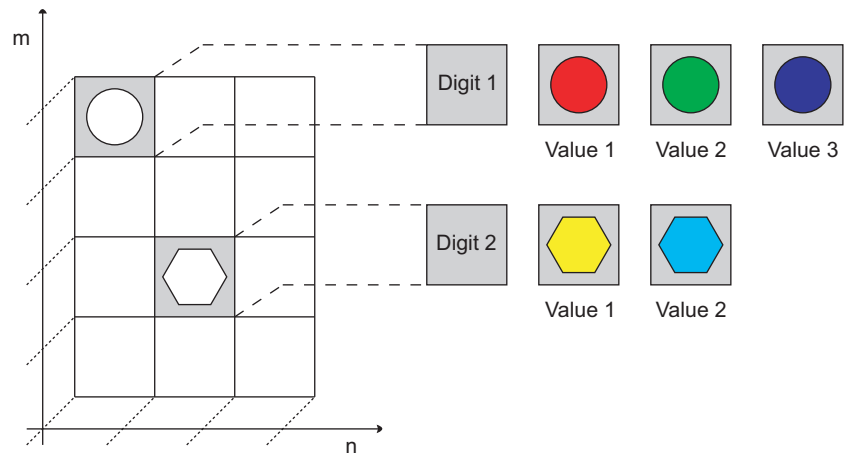


Abbildung 3.5: Funktionsweise der grafischen Passworteingabe.

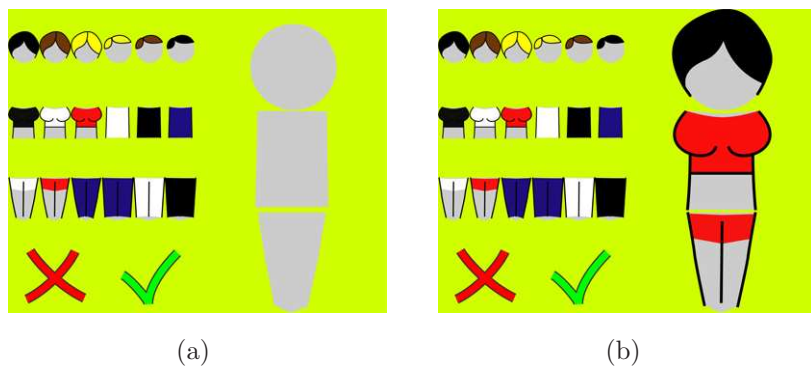


Abbildung 3.6: Beispiel einer grafischen Passworteingabe. (a) Bildschirm unmittelbar nach der Initialisierung, (b) Bildschirm mit eingegebenem Passwort.

```
1 <!ELEMENT PasswordTheme (digit+) >
2 <!ATTLIST PasswordTheme
3     name          CDATA    #REQUIRED
4     cols          CDATA    #REQUIRED
5     rows          CDATA    #REQUIRED
6     imagePath     CDATA    #REQUIRED
7     verifyURL     CDATA    #REQUIRED
8     cacelURL      CDATA    #REQUIRED
9 >
10
11 <!ELEMENT Digit (Value*) >
12 <!ATTLIST Digit
13     xPos          CDATA    #REQUIRED
14     yPos          CDATA    #REQUIRED
15     digitPos      CDATA    #REQUIRED
16     defaultValue CDATA    #REQUIRED
17     imagePath     CDATA    #REQUIRED
18 >
19
20 <!ELEMENT Value EMPTY >
21 <!ATTLIST Value
22     digitValue    CDATA    #REQUIRED
23     imagePath     CDATA    #REQUIRED
24 >
```

**Abbildung 3.7:** Quelltext der Dokumenttyp-Definition *PasswordTheme.dtd*.

# Kapitel 4

## Implementierung

### 4.1 Entwicklungs- und Testumgebung

Aber nicht nur bei der Wahl der einzusetzenden Technologien (wie im nächsten Kapitel zu sehen ist), sondern auch bei der Wahl der Entwicklungsumgebungen gibt es frei verfügbare Alternativen, die der kostenpflichtigen Konkurrenz um nichts nachstehen. Da als Entwicklungs-Rechner ein PC mit Windows XP als Betriebssystem verwendet wurde, wurden nur Produkte für den Einsatz auf diesem Betriebssystem recherchiert.

#### 4.1.1 Entwicklungsumgebung

Die Java-Applikationen wurden auf dem Java SDK Version 1.4.2.03 von Sun Microsystems entwickelt. Als Entwicklungsumgebung für Applikationen, die auf Java basieren, wurde Eclipse<sup>1</sup> verwendet. Eclipse ist frei verfügbar und bietet Features, welche auch in den Standard-Entwicklungsumgebungen für Java-Projekte sind. Diese Entwicklungsumgebung ist für eine Vielzahl von Betriebssystemen erhältlich: Windows, Linux, Solaris 8, Mac OSX, etc. So ist ein Projektaustausch zwischen Programmierern auf unterschiedlichen Plattformen problemlos möglich.

Als Entwicklungsumgebung für den Client für mobile Endgeräte wurde der Crimson Editor<sup>2</sup> verwendet. Der Crimson Editor ist ein Plain-Text-Editor mit erweiterten Textfunktionen und Syntax-Highlighting für zahlreiche Programmiersprachen. Zusätzlich können Makros über immer wiederkehrende Programmabläufe aufgezeichnet und externe Programmaufrufe mit zahlreichen Aufrufparameter-Möglichkeiten konfiguriert werden. Zum Testen der erstellten Dateien wurde der Browser mit der meisten Verbreitung - Microsofts Internet Explorer (Version 6.0) herangezogen.

---

<sup>1</sup><http://www.eclipse.org/>

<sup>2</sup><http://www.crimsoneditor.com/>

### 4.1.2 Testumgebung

Die *Testumgebung* verfügte über diverse Standard-PC-Desktop-Systeme, einige Standard-Notebooks, einen TabletPC und einen PDA. Bei den PCs und Notebooks kamen auch verschiedene Betriebssysteme zum Einsatz (Microsoft Windows XP Professional, Windows XP Tablet PC Edition für mobile Computer, Microsoft Windows Mobile 2003 Premium for Pocket PC, Linux Mandrake 10.0 Community - Linux Kernel 2.6.3). Sämtliche Komponenten dieser Geräte waren herkömmliche Standardkomponenten, wie sie heutzutage in jedem Rechner zu finden sind. Auf eine genauere Spezifizierung wird deshalb verzichtet. (Abb. 4.1 zeigt den verwendeten *PDA* und den verwendeten *TabletPC*.)

### 4.1.3 TabletPC

Beim *TabletPC* handelte es sich um einen Acer Travelmate C302XMi der C300 Serie. Die Serie verfügt standardmäßig über einen 14,1 Zoll TFT-Bildschirm, welcher eine maximale Auflösung von  $1024 \times 768$  Pixeln und über 16,7 Millionen (24 Bit) Farben darstellen kann. Der Bildschirm kann unter der Verwendung eines Stiftes als Eingabegerät benutzt werden. Die Leistungsmerkmale des Prozessors sowie des Arbeitsspeichers und der Festplatte stehen in keiner Weise der Leistung von Standgeräten nach (Mobiler Intel Pentium 4 Prozessor mit 1,4 GHz, 512 MB DDRAM Arbeitsspeicher und 60 GB Festplatte). Die Anbindung an Netzwerke ist über die bereits integrierten Netzwerkgeräte (Modem, Ethernet-Netzwerkkarte, Wireless LAN, Wireless PAN: Bluetooth) möglich. Als Betriebssystem wurde Microsofts Windows XP Tablet PC Edition für mobile Computer eingesetzt.

### 4.1.4 PDA

Der verwendete *PDA* war ein HP iPAQ Pocket PC h5550. Er verfügt über einen Intel 400 MHz Prozessor, 128 MB SDRAM und 48 MB Flash ROM. Das Display kann über 65.000 (16 Bit) Farben darstellen und hat eine Auflösung von  $240 \times 320$  Pixeln bei einer Bilddiagonale von 3,8 Zoll. Der PDA verfügt über das Betriebssystem Microsoft Windows Mobile 2003 Premium for Pocket PC.

## 4.2 Eingesetzte Technologien

Die Implementierung des gesamten verteilten Systems unterwirft sich einem einzigen Grundgedanken: Der Umsetzung des gesamten Systems ausschließlich mit stabilen und frei verfügbaren Technologien. Natürlich spielen auch andere Faktoren eine Rolle, beispielsweise ist auch die Plattformunabhängigkeit der einzelnen Clients wünschenswert. Die Entwicklung eines



**Abbildung 4.1:** HP iPAQ Pocket PC h5550 und Acer Travelmate C302XMi.

Clients, welcher auf allen Plattformen (Betriebssystemen und Endgeräten) läuft ist unter der Berücksichtigung der wichtigsten Regeln des User Interface Designs nicht möglich. Allerdings kann man mit den vorhandenen frei verfügbaren Technologien sehr gute Ergebnisse im Bereich der Betriebssystemunabhängigkeit erzielen. Schwieriger ist es ein von Endgeräten absolut unabhängiges Resultat zu erzielen, da die verschiedenen Endgeräte sehr unterschiedliche Spezifikationen haben. Eine Lösung, welche für nahezu alle der bereits in Kapitel 3.1 eingegrenzten Gruppe von Endgeräten umsetzbar wäre, ist ein WAP-Client. Der Vorteil ist klar: Diese Lösung würde alle Endgeräte beginnend bei Billig-Mobiltelefonen bis hin zu High-End-PCs abdecken. Allerdings würde diese rein auf Textelementen basierende Lösung viele Umsetzungsmöglichkeiten und Annehmlichkeiten, die Endgeräte mit größeren Bildflächen und anderen Hardware-Vorteilen bieten, ausschalten. Deshalb wurden zwei Arten von Clients implementiert. Ein Client zum Einsatz auf schnellen Rechnern mit großer Anzeigefläche und ein Client für alle Endgeräte mit voll funktionsfähigem Webbrowser.

#### 4.2.1 Client für Personal Computer

Der erste Client - der Client für Personal Computer - bietet den größten Freiraum und besitzt die wenigsten Einschränkungen - seine Zielgruppe beginnt ab den herkömmlichen Desktop PCs. Der PC ist weder in Bezug auf Speicher

noch in Bezug auf die Größe der Anzeigefläche noch in irgendeiner sonstigen Weise eingeschränkt. Man kann hier aus einem unendlich großen Technologieangebot schöpfen. Doch das große Angebot bringt auch einen nicht zu unterschätzenden Nachteil: Die Entscheidungsfindung über die einzusetzenden Technologien wird erheblich aufwändiger und dauert entsprechend länger. Die entscheidungsbefugte Person benötigt einen Überblick über aktuelle und alte Entwicklungen und muss in der Lage sein die verschiedenen Technologien zu bewerten.

In diesem Falle wurde der Client auf der und für die Microsoft Windows XP Plattform entwickelt. Da liegt es sofort nahe den Client in C oder C++ umzusetzen und ein kompiliertes, selbstständig lauffähiges Programm zu entwickeln. Jedoch entspricht diese Vorgehensweise nicht den auferlegten Kriterien des Clients, da dieser Ansatz keineswegs plattformunabhängig wäre. Ein Ansatz mit vielen positiven Eigenschaften ist der gewählte Ansatz: die Umsetzung des Clients in Java. Java bietet die Plattformunabhängigkeit und ist absolut frei von lizenzbedingten Restriktionen. Was in der Vergangenheit oftmals gegen den Einsatz von Java sprach, ist die im Vergleich zu anderen Programmiersprachen eher behäbige Performance und ein relativ großer Bedarf an Arbeitsspeicher während der Laufzeit. Doch durch die laufenden Performancesteigerungen bei den PCs generell und die Weiterentwicklung von Java selbst spielen diese Argumente heutzutage nur noch eine untergeordnete Rolle.

An den Client für Personal Computer wurden verschiedene technologische Anforderungen gestellt. Es war zB nötig Vektorgrafiken darzustellen und XML Daten zu parsen. Außerdem sollte er über eine HTTP-Verbindung kommunizieren. Im Zuge der Recherche über das Angebot an Technologien konnte festgestellt werden, dass Java all diesen Anforderungen entspricht. Zwar sind einige Funktionalitäten nicht von Haus aus in der Java API zu finden, jedoch stellt eine große Community rund um die Programmiersprache diverse Open Source Projekte zur freien Verfügung. So konnten schnell geeignete Erweiterungen zur Standard Java API gefunden werden, welche alle nötigen Funktionen abdecken.

Der Trend zu Vektorgrafik-GUIs wurde bereits in Kapitel 2.1 vorgestellt und verweist auf einige zukünftige Technologien. Zum aktuellen Zeitpunkt sind die Möglichkeiten jedoch noch etwas beschränkter. Es gibt nur einen einzigen herausragenden Vektorgrafik-Standard, welcher optimal für die Bedürfnisse des Clients geeignet ist. Viele Vektor-Formate sind proprietäre Herstellerformate und oftmals nicht frei verfügbar. SVG stellt so gesehen eine Ausnahme dar. SVG wurde vom W3C veröffentlicht und jedem frei zugänglich gemacht. Trotz des mehrjährigen Bestehens findet SVG zur Zeit keinen großen Einsatz, viele SVG-Projekte werden seit Jahren nicht mehr weiterentwickelt und es darf auch an der Zukunft von SVG aufgrund der mächtigen Konkurrenz gezweifelt werden. Allerdings ist SVG bereits jetzt eine ausgereifte, stabile und freie Technologie und es gibt noch eine

kleine aktive Community.

Um SVG-Grafiken in Java Applikationen darstellen zu können kann man entweder die XML-konformen SVG-Dateien mittels eines XML Parser verarbeiten und dann in Java grafisch darstellen, oder man bedient sich einer der wenigen Java-basierten SVG-Projekte. Für den Client brauchbare und nennenswerte SVG-Projekte in Java, die recherchiert werden konnten, sind die Open Source Projekte Jazz, aus welchem später das Projekt Piccolo hervorging, der Csiro SVGToolkit und das Projekt Batik von der Apache Group.

Jazz und Piccolo<sup>3</sup> wurden an der University of Maryland unter der Leitung von Professor Ben Benderson entwickelt. Sie stellen Toolkits dar, welche es erlauben komplexe zoomable GUIs zu erschaffen. Die SVG-Darstellungsfunktion stellt also nur einen sehr kleinen Bereich dieser Projekte dar. Am 21. April 2003 wurde die Entwicklung von Jazz eingestellt. Aus Jazz ging aber Piccolo hervor und am 25. April 2003 wurde Piccolo Version 1.0 veröffentlicht. Von der Java-Implementierung von Piccolo gibt es bis dato noch keine neue Version. Jazz und Piccolo unterstützen beide die gleichen Basic Features, wie zB strukturierte Grafiken und Kameras, welche die Ansicht darstellen. Die großen Unterschiede finden sich in der Komplexität und in der Geschwindigkeit, wobei Jazz das Nachsehen hat. Für Applikationen, welche eine zoomable GUI erfordern, sind diese Projekte hervorragend geeignet, jedoch um nur die SVG-Funktionalität zu nutzen scheint die Integration der Projekte in den Client etwas übertrieben. Außerdem ist nur eine begrenzte SVG-Funktionalität implementiert.

Letztendlich blieben noch der SVGToolkit und Batik als direkte Konkurrenten übrig. Die letzte Version des Csiro SVGToolkit<sup>4</sup> vom 12. März 2003 kann unter JDK 1.4 kompiliert und ausgeführt werden. Das Toolkit verwendet Xerces<sup>5</sup> 2.0.1 und Rhino<sup>6</sup> Version 1.5 release 3. Xerces wird später noch genauer vorgestellt. Rhino ist eine Implementierung von JavaScript, welche in Java geschrieben ist.

Die aktuelle Version 1.5.1 des Projekts Batik<sup>7</sup> der Apache Group integriert zahlreiche externe Packages, wie zB Xerces 2.5.0, und benötigt somit auch relativ viel Speicherplatz. Allerdings unterstützt Batik sehr viele SVG-Features und ist nicht nur auf die Anzeige von SVG-Dokumenten beschränkt, sondern kann diese auch erstellen und manipulieren.

Die wesentlich größere Anzahl an unterstützten SVG-Features, wie zB die Anzeige von Filtern, etc., welche bereits zum aktuellen Zeitpunkt korrekt interpretiert werden, führte zur Entscheidung Batik für den Client einzusetzen.

---

<sup>3</sup><http://www.cs.umd.edu/hcil/jazz>

<sup>4</sup><http://sis.cmis.csiro.au/svg/>

<sup>5</sup><http://xml.apache.org/xerces2-j/>

<sup>6</sup><http://www.mozilla.org/rhino/>

<sup>7</sup><http://xml.apache.org/batik/>

Zum Datenaustausch wird eine HTTP-Verbindung benötigt, über welche XML-Daten empfangen werden. Das HTTP-Protokoll wird bereits von der Standard-Java-API unterstützt. Allerdings muss zum Parsen der XML-Dokumente ein externes Projekt herangezogen werden. Auch hier gibt es bereits zahlreiche Open Source Projekte, jedoch war hier keine Recherche notwendig. Der Grund dafür ist, dass das Projekt Xerces der Apache Group bereits für die Batik SVG API benötigt wird und durch die Integration von Batik in den Client auch Xerces bereits integriert ist. Deshalb wurde zur Implementierung des XML-Parsers Xerces herangezogen.

Die Soundausgabe für das akustische Feedback der GUI wird von der Standard-Java-API übernommen.

Während der Implementierung konnten zahlreiche Erfahrungen gesammelt werden. Zum Beispiel ist der implementierte SVGLoader, welcher sich auf das Batik-Framework stützt, um einiges schneller im Laden von SVG-Grafiken, wenn das geladene Dokument direkt in eine Pixelgrafik umgewandelt und ausgegeben wird. Leider verliert man so eine Vielzahl von SVG-Funktionalitäten, jedoch werden diese beim Client für Personal Computer nicht unbedingt benötigt. Beispielsweise verliert man die Scripting-Funktionalität und die Möglichkeit zur Darstellung von Animation innerhalb eines SVG-Dokuments. Einige Funktionalitäten wären sicherlich wünschenswert, jedoch erfordert der Einsatz von SVG in vollem Umfang ein relativ umfangreiches Exception-Handling, damit mögliche Fehler im SVG-Code von Designern zu keinen Fehlern im Client selbst führen. Die Einschränkung der SVG-Funktionalität im Client für Personal Computer geht leider direkt mit der begrenzten Entwicklungszeit einher.

Im Laufe der Implementierung wurde aus dem SVGLoader, welcher ein Canvas-Objekt (welches wiederum den Inhalt des SVG darstellt) zurückliefert, ein sehr umfangreiches Objekt, welches entweder als reines Darstellungsobjekt von SVG-Dokumenten oder als Objekt mit Button-ähnlichen Eigenschaften verwendet werden kann. Zur Darstellung eines SVG-Dokuments genügt es das Objekt mit einer URI zu initialisieren. Danach wird der Inhalt der unter der URI zu findenden SVG-Datei dargestellt. Um das Objekt als Button zu betreiben kann man auf eine Funktion zugreifen, welche in den Vordergrund der Canvas für eine kurze Zeit eine geometrische Form einblendet. Diese Funktion liefert also das grafische Feedback. Sinnvollerweise ruft man diese Funktion dann auf, wenn der User auf das Objekt geklickt hat. Damit das grafische Feedback tatsächlich angezeigt wird, ist es nötig, die Funktion, die das Neuzeichnen der Grafik veranlasst, bei `mousePressed` und die anschließende Aktion erst bei `mouseReleased` oder `mouseClicked` auszuführen. Der Grund hierfür liegt darin, dass die Methode `repaint()` des Grafik-Objektes asynchron ist und nicht ausgeführt wird. Der Batik SVG-Loader liefert automatisch das Feature, dass die SVG-Grafik immer das richtige Höhen- und Breitenverhältnis aufweist, egal in welcher Größe man die Grafik ausgibt. Hat das Zielbild ein anderes Verhältnis von Höhe

und Breite, dann wird der überschüssige Bereich mit der Hintergrundfarbe gefüllt. So muss sich der Programmierer auch nicht mehr um eine unterschiedliche Behandlung unterschiedlicher Bildschirmauflösungen kümmern. Das Standardverhältnis von 4 : 3 wird mittlerweile schon von vielen Grafikkarten um andere Verhältnisvarianten (wie zB 4 : 3,2) erweitert. Im Falle des Clients ist der Effekt der „Nichtverzerrung“ zwar ein großer Vorteil, aber es gibt auch Anwendungsfälle, in denen dieses Verhalten nicht wünschenswert ist. Beispielsweise können durch dieses Verhalten Anschlusslücken zwischen Grafiken entstehen, welche eigentlich Kante an Kante anliegen müssten. Auf diesen Umstand muss man beim grafischen Design Rücksicht nehmen oder es müssen programmiertechnische Vorkehrungen getroffen werden.

#### 4.2.2 Client für mobile Endgeräte

Beim zweiten Client - dem Client für mobile Endgeräte - wurde ebenfalls auf eine freie Technologie zurückgegriffen. Außerdem ist die Realisierung insofern ressourcensparend, dass keine zusätzlich Software am Client installiert werden muss. Es handelt sich um einen Client, welcher per Browser visualisiert wird. Im Prinzip kann die Oberfläche auf jedem Endgerät angezeigt werden. Die einzige Voraussetzung ist ein Standard-Internet-Browser. Bei der Umsetzung wurde das Erscheinungsbild für die Auflösung des PDAs optimiert, dadurch erscheint die Größe der Oberfläche und der Interaktions-Elemente bei großer Bildschirmauflösung sehr klein.

Bei der freien Technologie handelt es sich um die *Extensible Stylesheet Language Transformation (XSLT)* [1, S. 50-51]. Diese Transformation ist in der Lage die Daten einer XML-Datei in eine andere Struktur zu verwandeln. Die Vorgehensweise ist folgende: Man versieht die XML-Datei mit einer Processing Instruction names `xml-stylesheet`:

```
<?xml-stylesheet type="text/xsl"
                href="http://localhost/transform.xsl"?>
```

Der Browser wendet dadurch die in der XSL-Datei definierte XSL-Transformation auf das XML-Dokument an.

Das Problem bei der PDA-Version des Internet Explorers ist, dass die übermittelten *ISO-8859-15*-kodierte XML-Zeichenketten, in *UTF-8* Zeichenketten verwandelt werden müssen. Die Zeichentabelle *ISO-8859-15* (Abb. 4.2, auch bekannt als *Latin-9*) verschlüsselt jedes Zeichen mit 8 Bits (1 Byte). Dadurch ist die Anzahl der verwaltbaren Zeichenkodierungen wie bei zahlreichen anderen Zeichentabellen auf 256 Zeichen limitiert. Das *8-bit Unicode Transformation Format (UTF-8)* ermöglicht eine verlustfreie Zeichen-Codierung, welche pro Zeichen 1 bis 4 Bytes zur Kodierung verwendet. Dadurch können viel größere Zeichenräume abgedeckt werden als zB mit auf 256 Zeichen beschränkte Tabellen [19].

ISO/IEC 8859-15																
	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	unused															
1x	unused															
2x	SP	!	"	#	\$	%	&	'	(	)	*	±	ˆ	-	˘	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6x	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{		}	~	
8x	unused															
9x	unused															
Ax	NBSP	ı	ç	£	€	¥	Š	š	©	ª	«	¬		®	¯	
Bx	°	±	²	³	Ž	μ	¶	·	ž	ı	o	»	Œ	œ	ÿ	ı
Cx	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
Dx	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
Ex	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
Fx	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Abbildung 4.2: ISO-8859-15 Zeichentabelle aus [19].

Ein weiteres Problem ist eine Einschränkung bei XSL-Anweisungen. Es ist nicht möglich `<xsl:apply-templates />` zu verschachteln. Ein Work-around ist der Einsatz der Anweisung `<xsl:for-each />`.

Eine andere Schwachstelle ist das Debugging. Erhält man keine grafische Ausgabe am Screen, dann ist eine schrittweise, zeitaufwändige Suche nach der Fehlerquelle erforderlich.

### 4.3 Client für Personal Computer

Dieses Kapitel beschäftigt sich mit der Funktionsweise des Clients für Personal Computer aus der Sicht des Benutzers und dokumentiert alle Details zu den gezeigten Screenshots. Der Client wurde auf Windows XP und Linux getestet und lieferte das gleiche Ergebnis. Die Konfiguration auf der User-Seite erfolgt über ein Java-Properties-File. Abb. 4.3 zeigt den Quelltext einer Beispiel-Konfigurationsdatei und die Beschreibungen zu den zu setzenden Werten:

Startet der User den Client in einer Umgebung mit mehreren Anzeigegeräten, so erscheint auf jedem Gerät Abb. 4.4.

Nach der Auswahl des Anzeigegerätes oder direkt nach dem Start der Applikation in einer Umgebung mit nur einem Ausgabegerät erfolgt die Initialisierung von Standardwerten durch den Server. Dadurch ist es möglich, dass der Server Werte, die aus dem Java-Property-File stammen, über-

```

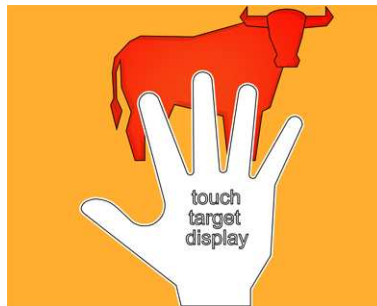
1  ## NECESSARY INITIAL VALUES AT CLIENT START
2  ##
3  ## User-Agent      = User-Agent header field value for the
4  ##                  initial http-request (may be changed by
5  ##                  server then)
6  ## fontSize        = initial font size
7  ## color            = initial main color
8  ## clickDelay      = delay of the visual click notification
9  ## messageDelay    = time of showing message windows
10 ## initialContentPath = URL to the initial xml file
11 ## multiScreenGraphic = URL to the selectScreen SVG
12 ## itemCols         = number of items on the screen horizontal
13 ## itemRows         = number of items on the screen vertical
14 ##                  if there are more items than
15 ##                  itemCols * itemRows the scroll control
16 ##                  is shown
17 ## soundOutput      = toggle audio feedback (true/false)
18 ## soundDelay       = time in ms of delay to play sound
19 ##                  without interruption
20 ## showMouseCursor = set mouse cursor visibility (true/false)
21
22 User-Agent=initial
23 fontSize=20
24 color=#D0FF00
25 clickDelay=125
26 messageDelay=2000
27 initialContentPath=http://localhost/init
28 multiScreenGraphic=http://localhost/screen.svg
29 itemCols=4
30 itemRows=3
31 soundOutput=true
32 soundDelay=500
33 showMouseCursor=false

```

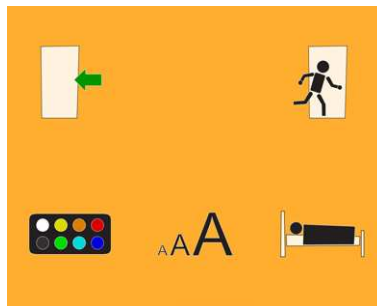
**Abbildung 4.3:** Quelltext einer Beispiel-Konfigurationsdatei des Clients.

schreibt. So können Werte für alle Clients trotz individueller Festlegung am Client selbst global gesetzt werden. Den Start-Screen der Applikation zeigt Abb. 4.5.

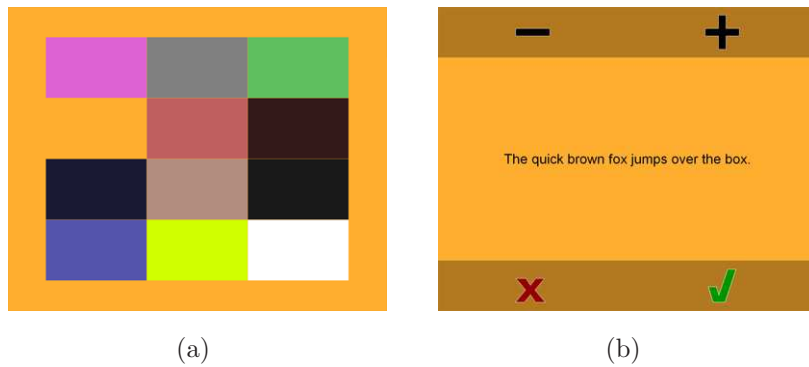
Der User hat nun fünf Möglichkeiten zur Auswahl: Mit der Anmeldeprozedur zu beginnen, das Programm zu beenden, die Hintergrundfarbe zu ändern, die Schriftgröße im System zu ändern oder den Bildschirmschoner zu starten. Der Bildschirmschoner ist nichts anderes als eine schwarze Fläche, die den Bildschirm ausfüllt. Abb. 4.6 zeigt die Konfigurationsmöglichkeiten am Client. Hintergrundfarbe (Screen in Abb. 4.6(a) und Schriftgröße (Screen in Abb. 4.6(b)).



**Abbildung 4.4:** Inhalt der Anzeigegeräte nach dem Programmstart in einer Umgebung mit mehreren Bildschirmen.



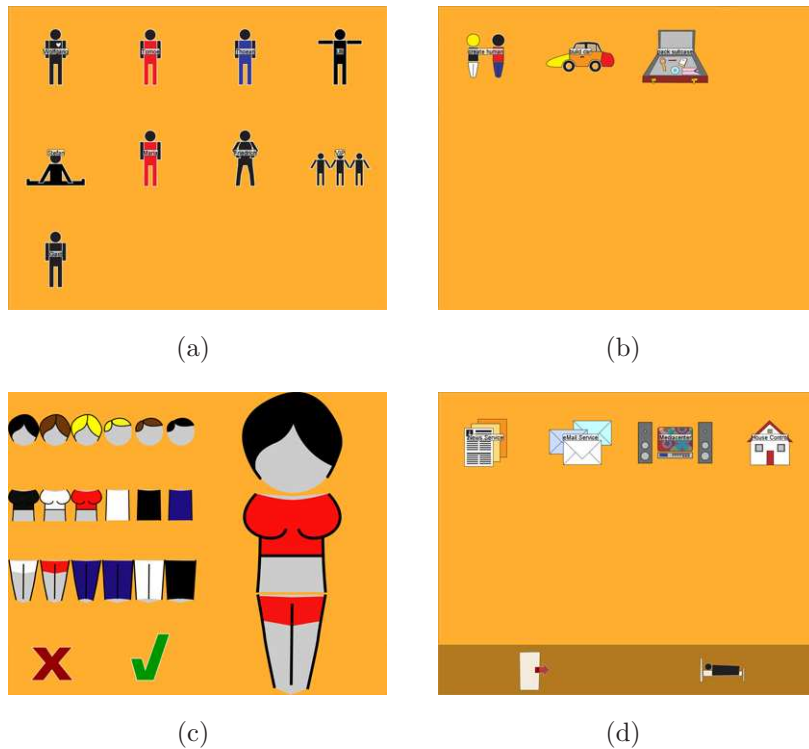
**Abbildung 4.5:** Start-Screen.



**Abbildung 4.6:** Konfigurationsmöglichkeiten am Client. (a) Screen zum Ändern der Hintergrundfarbe, (b) Screen zum Einstellen der Schriftgröße.

Die Anmeldeprozedur läuft wie folgt ab: Der User bekommt eine Liste von verfügbaren Usern angezeigt (Abb. 4.7(a)) und entscheidet sich für seinen Account. Danach erscheint ein Screen mit einer Liste aller verfügbarer

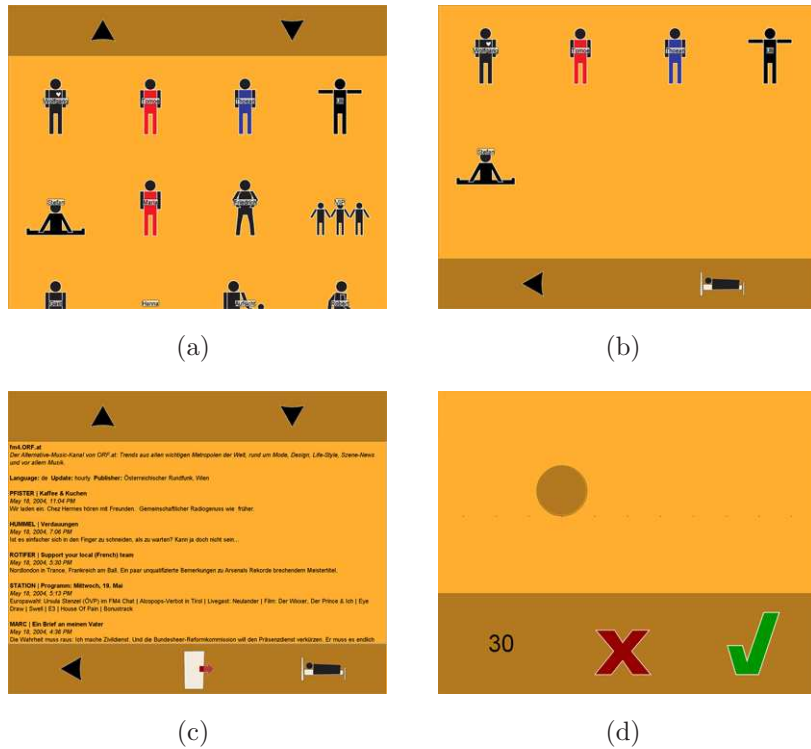
Passwort-Themen (Abb. 4.7(b)). Nach der Wahl des gewünschten Passwort-Themas gelangt er zum Eingabe-Screen des Passworts (Abb. 4.7(c)). Bei Erfolg gelangt der User zur Anzeige aller verfügbarer Services (Abb. 4.7(d)), andernfalls muss er die Anmeldeprozedur wiederholen.



**Abbildung 4.7:** Ablauf der Anmeldeprozedur. (a) Userliste, (b) Liste aller Passwort-Themen, (c) Eingabe des Passworts, (d) Nach erfolgreicher Anmeldung: Service-Liste.

Die darauffolgenden Screens entsprechen mit wenigen Ausnahmen einem einzigen Schema. Alle vom Server gelieferten Elemente werden am Bildschirm angeordnet. Übersteigt die Anzahl der Elemente die Größe der Anzeigefläche, so werden Scroll-Elemente eingeblendet (Abb. 4.8(a)). Weiters können in einem speziellen Bereich Elemente angezeigt werden, die immer sichtbar sein sollen (Abb. 4.8(b)). Eine Ausnahme vom herkömmlichen Schema zeigt Abb. 4.8(c) - den Nachrichtendienst. Hier wird der Inhalt des Nachrichtendienstes anstelle von Elementen angezeigt.

Es gibt verschiedene Arten von Elementen. Elemente die keine Interaktion auslösen - Abb. 4.7(d), Abb. 4.7(d) und Abb. 4.7(d). Ein Beispiel für einen Slider zeigt Abb. 4.8(d). All diese Screens und Elemente ermöglichen die Interaktion des Users mit dem System.

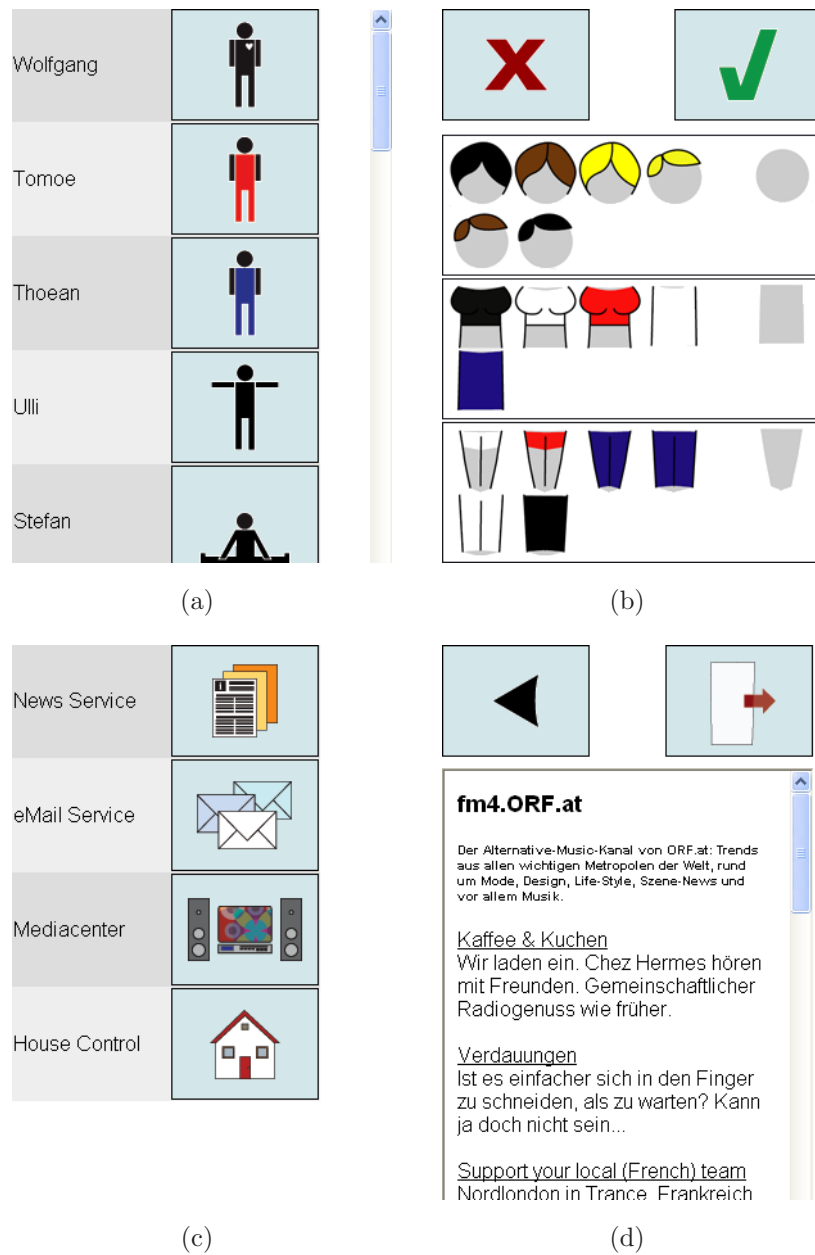


**Abbildung 4.8:** Diverse Client-Screens. (a) Screen mit Scroll-Elementen, (b) Screen mit ständig sichtbaren Eingabe-Elementen, (c) Beispiel Nachrichtendienst, (d) Eingabe-Element *Slider*.

## 4.4 Client für mobile Endgeräte

Der Client für mobile Endgeräte ist im Wesentlichen dem Client für Personal Computer nachempfunden. Allerdings verfügt er über keine eigenständige Applikation und auch über keine Konfigurationsdatei. Die Screens sind im Wesentlichen ebenfalls die gleichen wie beim Client für Personal Computer. Lediglich die Anordnung der Elemente ist teilweise anders. Abb. 4.9 zeigt ausgewählte Screens des Clients für mobile Endgeräte.

Die Implementierung des Clients für mobile Endgeräte basiert auf XSL-Transformationen. Eine Alternative wäre Sun Microsystems *Mobile Java (J2ME)*. Unter J2ME versteht man Java für kleine Endgeräte. Die Palette reicht hierbei von Pagern über Mobiltelefone bis zu Set-Top Boxen. J2ME besteht aus Konfigurationen, Profilen und optionalen APIs. Konfigurationen sind für bestimmte Typen von Geräten, basierend auf Speicherbeschränkungen und Prozessorleistung, designed und spezifizieren eine entsprechende Java Virtual Machine. Profile basieren auf der darunterliegenden Konfiguration und stellen die APIs (zB User Interface, dauerhafter Speicher, usw.)



**Abbildung 4.9:** Ausgewählte Screens des Clients für mobile Endgeräte. (a) Userliste, (b) Passwordeingabe, (c) Service-Liste, (d) Nachrichtendienst.

zur Verfügung, welche notwendig sind um ausführbare Applikationen erstellen zu können. Optionale APIs können noch zusätzliche Funktionalitäten zur Verfügung stellen [10, S. 1]. Abb. 4.10 stellt das Schema von J2ME

grafisch dar.

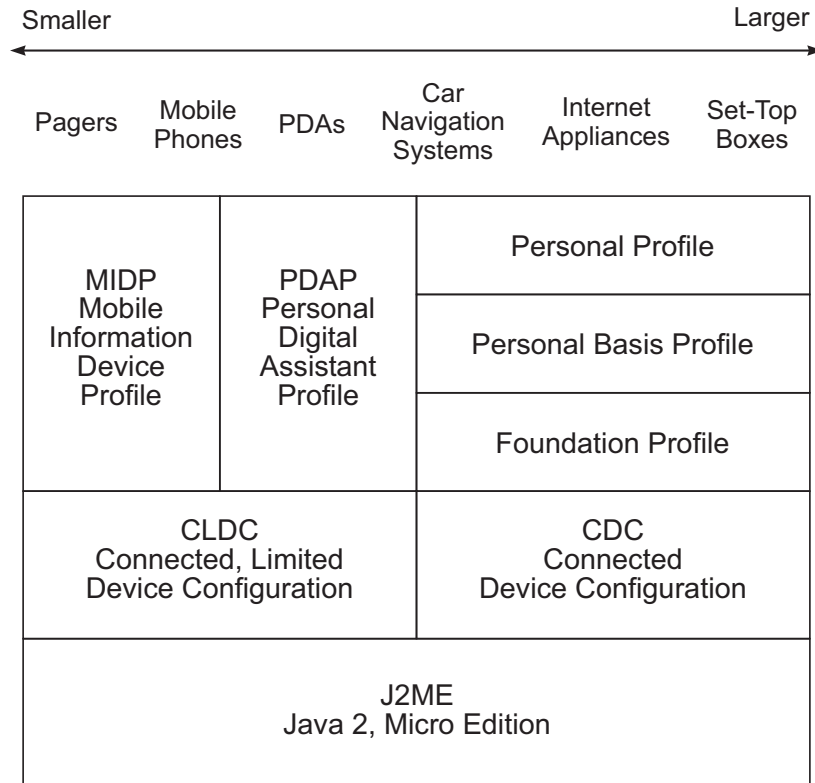


Abbildung 4.10: Mobile Java (J2ME) Schema [10, S. 2].

MIDP 1.0 ist ein von vielen Herstellern unterstütztes und heutzutage auch in der Billig-Mobiltelefon-Schiene zu findendes Profil. Es stellt zahlreiche User Interface Funktionalitäten und viele andere Features (wie zB das HTTP-Protokoll) zur Verfügung. Unter der Zuhilfenahme einer externen XML-API (zB kXML<sup>8</sup> oder Xparse-J<sup>9</sup>) wäre eine Umsetzung des Clients für Personal Computer für J2ME-fähige Endgeräte ohne weiteres möglich. Allerdings muss auf die SVG-Funktionalität verzichtet werden und man müsste stattdessen mit Pixelbildern oder reinen Textdaten arbeiten. Allerdings wäre das Ergebnis, sofern es sich an MIDP 1.0 hält, auf den MIDP 1.0-konformen Geräten verschiedener Hersteller ohne Adaption lauffähig.

Sei November 2002 ist MIDP 2.0 spezifiziert und mittlerweile finden sich vereinzelt Geräte mit der entsprechenden Unterstützung am Markt. MIDP 2.0 bietet neben zahlreichen Erweiterungen eine herausragende Eigenschaft: Es ermöglicht es Applikationen am mobile MIDP 2.0-Endgerät

<sup>8</sup><http://kxml.enhydra.org/>

<sup>9</sup><http://www.webreference.com/xml/tools/xparse-j.html>

remote zu starten. So könnte beispielsweise eine zukünftige Version des implementierten Komplettsystems beim Eintreten bestimmter Bedingungen Mobiltelefone benachrichtigen und Aktionen auf die eingetretenen Ereignisse vom Benutzer fordern. Theoretisch wäre eine Interaktion mit dem Benutzer auch via SMS möglich, jedoch gewährleistet SMS keine Benachrichtigung ohne Zeitverzögerung [9].

## Kapitel 5

# Schlussbemerkungen

Im Rahmen der Kooperation zwischen dieser und einer anderen Diplomarbeit wurde bewiesen, dass eine kostengünstige Variante des interaktiven Heims der Zukunft näher liegt als man denkt. Mit ausschließlichen Einsatz frei verfügbarer Technologien ist es gelungen ein komplexes verteiltes System zu schaffen, welches dem Benutzer über verschiedene Interaktionskanäle Geräte und Dienste des modernen Alltags zur Verfügung stellt.

Die Palette an möglichen Modulen für das System ist schier unendlich. Es wurden bereits ein Nachrichtendienst, ein Dienst zum Empfang von Emails, ein Kompass/GPS-Modul zur Positionsbestimmung von Kompass/GPS-Daten-Sendern und ein Building-Control-Modul zur Steuerung von Licht, Heizung, anderen European Instabus-kompatiblen Geräten und Medien-Renderern realisiert bzw. näher spezifiziert.

Durch die Integration von Fragmenten, Ergebnissen und Erfahrungen diverser Vorgängerprojekte und die Realisierung neuer Konzepte und Applikationen konnte ein ganzheitliches System realisiert werden, welches in der Lage ist auf einfachste Weise neue Module zu integrieren und das Gesamtsystem um zahlreiche Funktionalitäten zu erweitern. Die intuitiven Steuerungs-Oberflächen ermöglichen den einfachen Zugang zu dieser Technologie für alle Schichten und Altersgruppen von Menschen. Das System kann für jede Person individualisiert werden und jede Person mit den Diensten versorgen, die für diese Person relevant sind. Die Interaktion mit dem System kann nicht nur innerhalb des physischen Bereiches des Systems erfolgen, sondern auch über Weitstrecken-Verbindungen.

Eine Produktreife wurde noch nicht erreicht, aber das System kann durchaus als Prototyp im Haushalt einer Testfamilie verwendet werden und durch Beobachtung und Sammlung von Erfahrungswerten Feedback und Anregungen für neue Projekte in dieser Richtung liefern.

Es konnte auch die Erkenntnis aus den Diplomarbeiten gewonnen werden, dass für die Entwicklung eines komplexen verteilten System wie diesem ein Team von mehr als zwei Personen wünschenswert ist. Dies hätte die Anzahl

der entwickelten Dienste und Oberflächen erhöhen können und gleichzeitig den Arbeitsaufwand jeder Person verringert.

Speziell in dieser Diplomarbeit wurden zwei Clients, welche eine breite Bandbreite von Eingabemedien abdecken entwickelt. Vor dem Software Design der Oberflächen wurden Richtlinien für das Software User Interface Design erarbeitet und diese dann in einem hohen Grad bei der Umsetzung angewandt. Technologien, die zur Zeit noch in den Startlöchern stehen, wurden vorgestellt und die bei der Implementierung erlangten Erkenntnisse dokumentiert.

Weiters wurde der komplette Kommunikationsweg zwischen den Clients und der Middleware des Komplettsystems, welche direkt die Geräte und Dienste ansteuert, entwickelt. Das Datenformat, welches zur Kommunikation verwendet wird, bietet eine Schnittstelle und ist so für zukünftige Entwicklungen gerüstet.

Die Kooperation wurde erfolgreich durchgeführt und ein für alle Teilnehmer zufriedenstellendes Ergebnis wurde erzielt.

# Anhang A

## Inhalt der CD-ROM

**File System:** Joliet

**Mode:** Single-Session (CD-ROM)

### A.1 Diplomarbeit

**Pfad:** /thesis/

Dieses Verzeichnis enthält die Diplomarbeit im PDF-Format und Kopien von in der Diplomarbeit zitierten Internet-Quellen.

### A.2 Implementierung

**Pfad:** /impl/

mc_source.zip . . . . .	Quelltexte des Clients für mobile Endgeräte
pc_bin.zip . . . . .	Ausführbare Version des Clients für Personal Computer.
pc_source.zip . . . . .	Quelltexte des Clients für Personal Computer.
sh_server.zip . . . . .	Ausführbare Version des SmartHome-Servers.

### A.3 Software

**Pfad:** /tools/

Dieses Verzeichnis enthält die Installationsdateien diverser verwendeter Software-Produkte.

# Literaturverzeichnis

- [1] BEHME, H. U. M. S.: *XML in der Praxis*. Addison-Wesley, München, 2. Aufl., 2000.
- [2] CONRAETS, C.: *An Overview of MXML, the Macromedia Flex Markup Language*. URL, <http://www.macromedia.com/devnet/flex/articles/paradigm.html>, Mai 2004. Kopie auf CD-ROM.
- [3] COOPER, A.: *About Face: The Essentials of User Interface Design*. IDG Books Worldwide, Inc, Foster City, CA, Aug. 1995.
- [4] COULOURIS, GEORGE, J. D. U. T. K.: *Distributed Systems: Concepts and Design*. Addison-Wesley, Harlow, 3. Aufl., 2001.
- [5] DREYFUS, S.: *Forget your password? Picture this..* www.underground-book.com, October 2000. Kopie auf CD-ROM.
- [6] ESPACENET.COM: *Europe's Network of patent databases*. URL, <http://at.espacenet.com/>, Mai 2004. Kopie auf CD-ROM.
- [7] GALITZ, W. O.: *The Essential Guide to User Interface Design*. Wiley & Sons, Inc., New York, NY, 2002.
- [8] JOHNSON, J.: *GUI Bloopers: Don'ts and Do's for Software Developers and Web Designers*. Morgan Kaufmann, San Francisco, CA, 2000.
- [9] JSR 118 EXPERT GROUP: *Mobile Java Information Device Profile Version 2.0*. URL, <http://jcp.org/en/jsr/detail?id=118>, November 2002. Kopie auf CD-ROM.
- [10] KNUDSEN, J.: *Wireless Java: Developing with J2ME*. Apress, Berkeley, CA, 2. Aufl., 2003.
- [11] MACDONALD, M.: *User Interfaces in C#: Windows Forms and Custom Controls*. Apress, London, 2002.
- [12] MANDEL, T.: *Elements of user interface design*. John Wiley & Sons, Inc., New York, NY, 1997.

- [13] NETWORKING WORKING GROUP: *Hypertext Transfer Protocol – HTTP/1.1*. URL, <http://www.w3.org/Protocols/rfc2616/rfc2616.html>, Juni 1999. Kopie auf CD-ROM.
- [14] NORMAN, D. A.: *[Psychology of everyday things], The design of everyday things*. Basic Books, New York, NY, 2002 Aufl., 2002.
- [15] RASKIN, J.: *Das intelligente Interface*. Addison-Wesley, München, 2001.
- [16] SHNEIDERMAN, B.: *User Interface Design*. Basic Books, Bonn, 3. Aufl., 2002.
- [17] SPOLSKY, J.: *User Interface Design for Programmers*. Apress, Berkeley, CA, 2001.
- [18] TANENBAUM, A. S. U. M. V. S.: *Distributed Systems: Principles and Paradigms*. Prentice-Hall, Upper Saddle River, NJ, 2002.
- [19] WIKIPEDIA.ORG: *Wikipedia - The Free Encyclopedia*. URL, <http://wikipedia.org/>, Mai 2004. Kopie auf CD-ROM.
- [20] XIPOLIS.NET: *Xipolis.net - Ihre Online-Bibliothek des Wissens*. URL, <http://xipolis.net/>, Mai 2004. Kopie auf CD-ROM.