TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

Diplomarbeit

Interaktionsdesign in Webapplikationen

ausgeführt am Institut für
Gestaltungs- und Wirkungsforschung

unter der Betreuung von
Ao. Univ.-Prof. Dr. Gerald Steinhardt
Univ.-Ass. Dipl.-Ing. Dr. Hilda Tellioglu

eingereicht
an der Technischen Universität Wien
Fakultät für Informatik

von
Marius Lessiak
Matrikelnummer 9926236

Wien, im Mai 2007

Inhaltsverzeichnis

Abbildungsverzeichnis

Abstract

Usability in modern web applications suffers from bad interaction. With the advent of AJAX it seems possible to improve the user experience by implementing interaction techniques which have been used successfully in desktop applications. Therefore this thesis focuses on interaction design as a strong concept to improve usability. Based on the comprehensive knowledge in the research area Human Computer Interaction (HCI) an interaction model for web applications is developed: it combines adequate interaction techniques for the different interactive components of a web application in order to ensure a high level of usability. Using state of the art web technologies as defined by AJAX a web application is engineered to prove the real life applicability of the interaction model. Following the development process a technical evaluation of the web application shows whether and how the interaction model can be used in practice.

Kurzfassung

In der vorliegenden Arbeit werden Konzepte der Usability, sowie deren Einbindung in den Softwareentwicklungsprozess erläutert. Die mit Usability eng verbundenen Begriffe Interface und Interaktion werden anhand wissenschaftlicher Arbeiten abgegrenzt und definiert. Das reichhaltige Wissen , welches aktuell im Forschungszweig der Human Computer Interaction (HCI) vorhanden ist, wird anhand der Arbeiten von Beaudouin-Lafon, Cooper und Shneiderman erklärt und beschrieben. Darauf aufbauend wird ein Interaktionsmodell für Webapplikationen entwickelt: in diesem werden geeignete Interaktionstechniken für die verschiedenen interaktiven Komponenten einer Webapplikation kombiniert, um ein hohes Maß an Usability zu versichern. Die Umsetzbarkeit des entwickelten Modelles - mit aktuell verfügbaren Web Technologien - wird anhand des Entwicklungsprozesses einer Webapplikation geprüft. Am Ende der Entwicklung folgt eine technische Evaluierung der Implementierung, die zeigt, ob und wie das entwickelte Interaktionsmodell in der Praxis anwendbar ist.

1 Einleitung

In den Anfängen der Software Entwicklung schenkt man dem Gedanken an die zukünftigen User wenig Aufmerksamkeit. Dies passiert nicht etwa aufgrund von fehlerhaften Planungsmodellen oder ungenügendem Wissen, sondern schlicht und einfach, weil die technischen Möglichkeiten ohnehin nur eine Usergruppe erlauben: die ExpertInnen. Die Ein- und Ausgabegeräte, mit denen die Programme arbeiten können, beschränken sich z.B. auf die Tastatur und einen 80 mal 24 Zeilen monochromen Bildschirm, wie im IBM 5271 [Ell05] aus dem Jahr 1970 (Abb. 1.1). Diese Geräte erlauben keine anderen Interaktionsformen als eine Kommandozeilenein- und ausgabe, was ihre Bedienbarkeit auf eine kleine Gruppe von Menschen reduziert. Das gängige Paradigma damals war, dass die User sich in ihrer Sprache und ihrem Verhalten an den Computer anpassen müssen, um ihn bedienen zu können.

Die Weiterentwicklung der Rechenleistung, mit der Möglichkeit komplexer grafischer Benutzeroberflächen, und die Verfügbarkeit neuer Ein- und Ausgabegeräte - Maus und Farbbildschirm - führt ab 1973 bis 1984 zu den Grundlagen moderner PCs. Der erste Computer mit grafischer Benutzeroberfläche ist der, 1973 im Forschungszentrum Xerox PARC (Palo Alto Research Center) entwickelte, Xerox Alto [Mus07], der jedoch nur in der Forschung eingesetzt wird. Von den Konzepten des Xerox Alto inspiriert entwickelt die Firma Apple ihr grafisches Betriebssystem Mac OS, das erstmals 1984 als Bestandteil des Apple Macintosh vorgestellt wird. Es beinhaltet Features, die in heutigen Applikationen allgegenwärtig sind:

- *Direct Manipulation Objects.* Direkte Manipulation von grafischen Elemente des User Interface durch z.B. Drag&Drop.
- *Maus.* Ein 2-dimensionales Eingabegerät mit mehreren Tasten.
- *Windows.* Im User Interface erhält jede Applikation ein Fenster, mehrere parallel laufende Applikationen erzeugen überlappende Fenster.

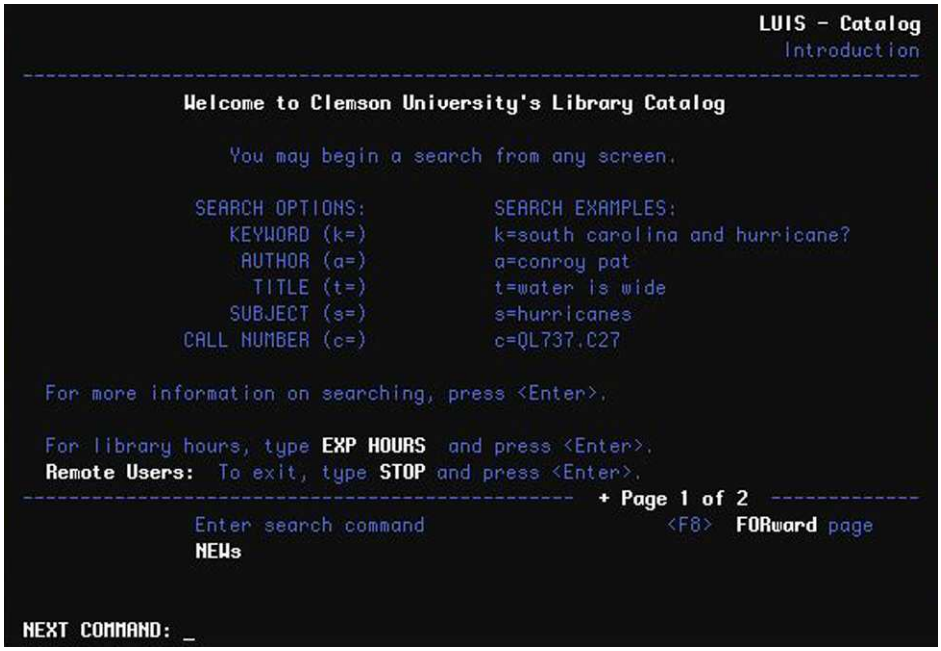

Abbildung 1.1: Bibliothekskatalog auf der Anzeige des IBM 5271

Die Firma Microsoft erkennt die Vorteile dieser Techniken und entwickelt wiederum ein derart inspiriertes Betriebssystem, das 1995 unter dem Namen Windows95 (Abb. 1.2) veröffentlicht wird.

Die Möglichkeiten der Nutzung des Computers haben sich damit radikal geändert: die einstige Maschine für spezielle Aufgaben, die nur von ausgebildetem Fachpersonal bedient werden konnte, wird zu einer Plattform für Anwendungen aller Art. Besonders die Textverarbeitung (WordStar, MacWrite) und Tabellenkalkulation (Lotus 1-2-3, Excel) sind Kaufargumente, die den Apple Macintosh (und ähnliche Geräte) nicht nur in viele Büros, sondern auch private Haushalte bringen. Der Personal Computer, kurz PC, wird zu einem kommerziellen Erfolg der 90er.

In der Softwareentwicklung spiegelt sich dies in einem Umdenken wider. Der Computer wird zum universellem Werkzeug, mit dem Aufgaben bewältigt werden können: einen Brief schreiben, ein Bild malen, das Budget erstellen etc. Der Lebenszyklus derartiger Anwendungen, von ihrer Konzeption bis zur Verwendung, wird Gegenstand einer neuen wissenschaftlichen Disziplin. Software Engineering ist in [Boe76, zit. nach KPRR03, S.4] definiert als

> „the application of science and mathematics by which the capabilities of com-

puter equipment are made useful to man via computer programs, procedures, and associated documentation.“

Da der PC mittlerweile zum Massenprodukt geworden ist, gibt es auch keine überschaubare und homogene Usergruppe mehr. Stattdessen will man die Applikationen für so viele User wie möglich nutzbar machen, nicht zuletzt um die kommerzielle Reichweite zu steigern. Die User sind nicht mehr die bekannten ExpertInnen, sondern eine neue, unbekannte aber wichtige Komponente in der Software Entwicklung.

Das verstärkte Interesse an der Rolle der User führt 1982 zu der ersten Konferenz der ACM's Special Interest Group on Computer-Human Interaction, einer weltweiten Organisation von ForscherInnen, deren Interesse der Interaktion zwischen Mensch und Computer gilt. Um mehr über die Möglichkeiten, Anforderungen und Einschränkungen der User herauszufinden, bedient man sich in den folgenden Jahren einer Kombination aus Methoden und Wissen der Informatik und anderer Disziplinen: Psychologie, Medizin, Soziologie. Damit formt sich der interdisziplinäre Bereich der Human Computer Interaction, kurz HCI. In [MHC+96, S.794] wird HCI wie folgt erklärt:

„Human-computer interaction (HCI) is the study of how people design, implement, and use interactive computer systems and how computers affect individuals, organizations and society.“

Die Erkenntnisse führen zu einer verbesserten Nutzbarkeit von Anwendungen sowie neuen Interaktions- und Kommunikationsformen in Computersystemen.

1989 schlägt Tim Berners-Lee im CERN, einer europäischen Großforschungseinrichtung, die Umsetzung eines Global Hypertext Space [BL98] vor, um Informationen besser zugänglich zu machen. Das Projekt wird genehmigt und im Jahr darauf geht im CERN der erste HTTP Server ans Netz. Die Informationen sind mit dem ebenfalls veröffentlichten Browser WorldWideWeb, der HTML Seiten anzeigt und per Maus bedient werden kann, abrufbar. In den darauffolgenden Jahren erweckt das Projekt Internet die Aufmerksamkeit von Firmen, die die kommerziellen Möglichkeiten erkennen: 1993 wird der NCSA Mosaic veröffentlicht, der erste Browser mit grafischer Benutzeroberfläche für Unix, Mac und Windows. Basierend auf Mosaic stellt die Firma Netscape 1994 die erste Version des Netscape Navigator vor, ein Jahr später bringt Microsoft den Internet Explorer auf den Markt. Spätestens 1995 mit Erscheinen des neuen Microsoft Betriebssystems Windows95, das den Internet Explorer sowie die technischen Vorraussetzungen für eine

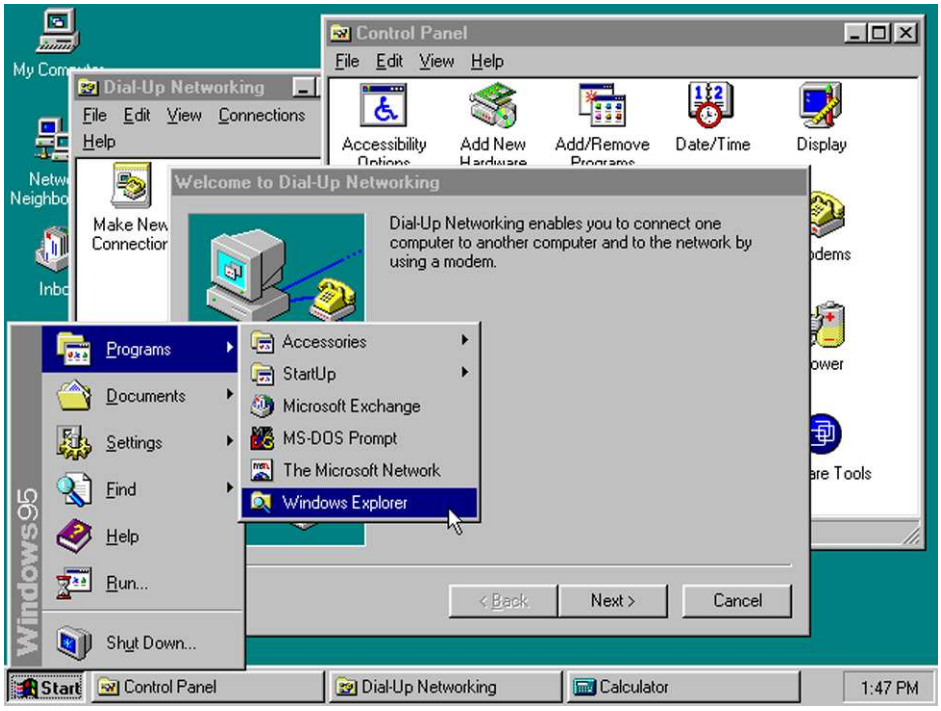

Abbildung 1.2: Benutzeroberfläche von Windows95

Verbindung zum Internet (TCP/IP) integriert, wird die Weltöffentlichkeit aufmerksam. Die Nutzung des stark wachsenden Internets mit seinen vielfältigen Inhalten sorgt für den Einzug des PCs in viele Lebensbereiche.

Der Softwaremarkt bietet mittlerweile eine große Anzahl von konkurrierenden Produkten, die um die Gunst der immer anspruchsvolleren User wetteifern. In der Software Entwicklung benötigt man daher etwas, mit dem sich die Zufriedenheit der zukünftigen User versichern lässt. Das Konzept der Usability von Software, in dessen Vordergrund das durchdachte und den Usern angemessene Design von Interaktion und Interfaces einer Applikation steht, bietet die gewünschten Eigenschaften. Der HCI Bereich wird somit in Industrie und öffentlichen Forschungseinrichtungen immer wichtiger. In Desktopapplikationen ist die Realisierung von Interfaces und Interaktionstechniken, die bei den Usern die größtmögliche Usability versichert, technisch gut umsetzbar: Die Möglichkeiten unterliegen prinzipiell keinen Beschränkungen, abgesehen von der verwendeten Hardware und des Betriebssystems. Dies zeigt sich in einer schnellen Entwicklung neuartiger Möglichkeiten, wie z.B. dreidimensionaler Navigationsräume oder drahtloser Eingabegeräte.

Das Internet, kurz das Web, entwickelt sich jedoch hinsichtlich der Usability vorerst

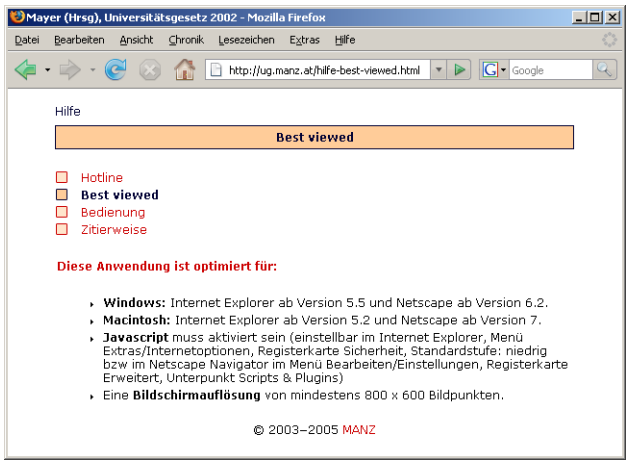

Abbildung 1.3: Browserspezifische Implementierungen infolge der Inkompatibilitäten

zurück. Zwischen 1995 und 1998 entbrennt zwischen den Firmen Microsoft und Netscape der so genannte Browserkrieg - ein Ringen um die Marktvorherrschaft zwischen dem Internet Explorer und dem Netscape Navigator. Diese Auseinandersetzung führt zu einer gravierenden Vernachlässigung der standardisierten Implementierung von Web Technologien, wie z.B. XHTML, JavaScript oder CSS, innerhalb der Browser. Stattdessen wollen die Firmen WebentwicklerInnen mit neuen, aber proprietären Erweiterungen für das eigene Lager gewinnen. Da sich die Entwicklung von Webapplikationen innerhalb der Grenzen und Möglichkeiten des verwendeten Browser bewegt, führt der Konkurrenzkampf zu einer problematischen Situation für die EntwicklerInnen: Die Produkte müssen mehrfach geplant, implementiert und getestet werden. Außerdem ändern sich die Browser-Implementierungen laufend, was wiederum die Funktionalität von schon erfolgreich abgeschlossenen Projekten beeinflussen kann. Diese Rahmenbedingungen schränken den erfolgreichen Einsatz von Interaktion und Interfaces zu Gunsten einer erhöhten Usability stark ein. Wo in Desktopapplikationen Konzepte, wie z.B. kontextsensitive Hilfe oder objektabhängige Menüs, schon angewendet werden, kämpfen Webapplikationen mit Kinderkrankheiten wie z.B. der korrekten browserübergreifenden Darstellung von Text und Bildern (Abb. 1.3).

Mit dem Verkauf von Netscape und der Einstellung der Weiterentwicklung des Netscape Navigator ist 1998 das Ende der Konfliktsituation mit Microsoft erreicht. Bis 2003 steigt der Marktanteil des Internet Explorers auf 95% [Bor03]. Ab 2004 wenden sich die User jedoch wieder verstärkt alternativen Browsern zu: Mozilla Firefox, Opera oder Safari. Diese Entwicklung beruht auf dem regelmäßigen Auftauchen von Sicherheitslücken

im Internet Explorer, der zunehmenden Verwendung alternativer Betriebssysteme sowie neuer internetfähiger Geräte (PDA, Mobiltelefon). Der Konkurrenzkampf am Weltmarkt ist diesmal allerdings von der korrekten Umsetzung verfügbarer Web Standards des World Wide Web Consortiums [Con06b], kurz W3C, geprägt und nicht durch laufende proprietäre Eigenentwicklungen. Vor allem die standardgetreue Implementierung von JavaScript [Ass99] und dem Document Object Model [Con05], in den gängigen Browsern, verbessern die Rahmenbedingungen für die Entwicklung von Webapplikationen erheblich.

Die eben skizzierte Situation führte in den letzten 2 Jahren dazu, dass gezielt neue Techniken zur Verbesserung der Usability von Webapplikationen umgesetzt werden können. Dabei greifen die EntwicklerInnen auf Bibliotheken und Frameworks zurück, welche die plattformübergreifende Programmierung von Browsern, auf einem erhöhten Abstraktionsniveau, erlauben. In Desktopapplikationen bekannte und bewährte Konzepte von Interaktion und Interfaces, wie z.B. Direct Manipulation, sollen damit in Webapplikationen realisierbar werden.

Das alleinige Umsetzen von Interaktionstechniken aus Desktopapplikationen reicht jedoch nicht aus, um die Ansprüche einer hohen Usability in Webapplikationen zu erfüllen. Dies folgt aus der eigenständigen Entwicklung des Web und den zugehörigen Technologien und Standards. Einerseits ist der Umgang der User mit Webapplikationen von eigenen Erwartungen und Ansprüchen geprägt, wie z.B. der leicht verständlichen Navigierbarkeit. Andererseits unterliegen die EntwicklerInnen bei der Implementierung von Interaktionstechniken restriktiveren Einschränkungen: Webapplikationen werden innerhalb der Grenzen und Möglichkeiten eines Browser Programmes ausgeführt und sind auf die permanente Kommunikation mit einem Applikationsserver angewiesen. Nicht jede bekannte und bewährte Interaktionstechnik eignet sich daher für die Umsetzung in einer Webapplikation. Das Ziel dieser Arbeit ist es, ein Interaktionsmodell für Webapplikationen zu entwickeln, das sowohl die besonderen Anforderungen der User berücksichtigt, als auch die Usability durch tatsächlich realisierbare Interaktionsformen steigert.

2 Usability

Die Entwicklung von Software beginnt mit der Idee einer neuen oder verbesserten Anwendung, welche die Menschen in einer schon bekannten Tätigkeit unterstützt oder ihnen neue Möglichkeiten eröffnet. Da der Computer mittlerweile in unzähligen Arbeits- und Lebensbereichen eingesetzt wird, sind der Anwendungsart theoretisch keine Grenzen gesetzt: Dokumentenerstellung, Visualisierung medizinischer Daten, Bloggen, E-Mails schreiben, Musik tauschen etc. Inwiefern das entwickelte Produkt, die Applikation, gebrauchstauglich ist, zeigt sich dabei vor allem an der Reaktion der User, die erst in einer späteren Phase des Entwicklungsprozesses bzw. -zyklus beobachtet und ausgewertet werden kann. Ist diese positiv, z.B. infolge von intuitiver Bedienung, guter Verständlichkeit und rascher Erlernbarkeit der Applikation, spricht man von einer hohen Usability. Doch was meint Usability eigentlich genau?

In [KPRR03] wird die Übersetzung von Usability mit Benutzerfreundlichkeit abgelehnt, da dieser Begriff, durch den Gebrauch seitens der Werbung im deutschen Sprachraum, seiner inhaltlich adäquaten Bedeutung beraubt wurde. Stattdessen wird die Verwendung des „deutschen Terminus Technicus“ *Gebrauchstauglichkeit* empfohlen. Eine etwas längere inhaltliche Definition gibt die internationalen Norm ISO/IEC 9241-11 [KPRR03, S.267]:

> „Das Ausmaß, in dem ein Produkt durch bestimmte Benutzer in einem bestimmten Nutzungskontext genutzt werden kann, um bestimmte Ziele effektiv, effizient und zufrieden stellend zu erreichen.“

Die häufige Verwendung des Wortes „bestimmt“ verdeutlicht, dass es keine generell gültigen Richtlinien gibt, mit denen eine hohe Usability erzielt werden kann. Viel mehr müssen mit der Applikation verknüpfte Elemente sorgfältig untersucht und analysiert werden, um das bestmögliche Design erarbeiten zu können. Jene Elemente aus der obigen Definition führen zur Klärung von wichtigen Aspekten der Entwicklung: Wie alt sind

die BenutzerInnen? Welches Vorwissen haben die BenutzerInnen? Wird die Applikation in einem Büro verwendet? Wird die Applikation auf einem mobilen Gerät verwendet? Wozu wird die Applikation verwendet?

Je genauer diese Fragen in einer frühen Phase eines Entwicklungszyklus beantwortet werden, desto höher wird später die Gebrauchstauglichkeit der fertigen Applikation sein. Die Anworten sollen eben dazu führen, den Gebrauch effektiv, effizient und zufrieden stellend zu gestalten. „Effektiv“ meint hier eine vollständige und genaue Ausführung, „effizient“ bezieht sich auf den Aufwand, der in Relation zu den erzielten Ergebnissen stehen muss, und „zufrieden stellend“ bezeichnet eine Nutzung, welche frei von Beeinträchtigungen erfolgt und bei den NutzernInnen positive Reaktionen auslöst.

2.1 Anforderungen

Bei der Planung von Softwareentwicklung stützt man sich auf hinreichend erforschte und erprobte Prozessmodelle, wie z.B. das klassische Wasserfallmodell [Som82] oder jüngere objektorientierte Ansätze [GG01]. Innerhalb dieser Modelle werden die Arbeitsschritte, von der Idee bis zur fertigen Applikation, auf einer allgemein anwendbaren Basis erläutert. Einer dieser Arbeitsschritte ist die Anforderungsanalyse, bei der die EntwicklerInnen Anforderungen bestimmen und formulieren, welche die Applikation erfüllen soll. Darauf basierend können dann z.B. Prototypen entwickelt und getestet werden. Bezieht man nun Usability in diese Anforderungsanalyse, die in einer frühen Phase eines Entwicklungszyklus durchgeführt wird, ein, ergeben sich nach [SP04] folgende Bereiche, die zu klären sind:

Die *Anwendung seitens der User*, also welche Aufgaben wie durchgeführt werden sollen, muss geklärt sein. Dies wirkt sich direkt auf die Funktionalität der Applikation aus. Dabei ist es vor allem wichtig, jeden möglichen Anwendungsfall zu bestimmen, egal ob er häufig, selten oder nur in kritischen Situationen auftritt. Ist die Menge der Anwendungsfälle unzureichend, können die User ihren gewünschten Arbeitsablauf nicht ausführen und werden die Verwendung der Applikation einschränken oder verweigern.

Verlässlichkeit und Verfügbarkeit von Funktionalität und Daten. Die Funktionen der Applikation sollen immer innerhalb der geplanten und gewohnten Verhaltensweise arbeiten und die Datenintegrität soll jederzeit gewährleistet sein. Unerwartete Ereignisse wäh-

rend eines Anwendungsfalles oder inkorrekte Daten führen zu einem Vertrauensverlust bei den Usern und belasten die Verwendung der Applikation.

Standardisierung, Integration, Konsistenz und Plattformunabhängigkeit erleichtern die Handhabung von Applikationen über Produktgrenzen hinweg. Ähnliche Verhaltensweisen bei unterschiedlichen Applikationen erlauben den Usern eine intuitive und rasche Bedienung. Die Belastung des Einarbeitens in unbekannte Anwendungen und die Fehleranfälligkeit der Bedienung wird reduziert.

Plangemäße Durchführung des Projektes. Der Zeit- und Budgetrahmen der Entwicklung soll nicht überschritten werden, damit die Applikation nach Plan eingesetzt werden kann. Infolge eines nicht eingehaltenen Projektplanes fehlerhafte, unvollständige oder verspätetet Applikationen verlieren das Vertrauen der User.

Das Auseinandersetzen und Erarbeiten der Inhalte dieser 4 Punkte bildet in einer frühen Phase eines Entwicklungszyklus die Basis für weitere Entscheidungen und Vorhergehensweisen. Damit soll versichert werden, dass das Design und die Implementierung der Applikation ein hohes Maß an Usability zur Verfügung stellen. So stellt sich die Frage, wie Usability analysiert und bewertet werden kann - um festzustellen, ob die erwünschten Ziele und Anforderungen erreicht worden sind.

2.2 Messbarkeit von Usability

Die Definition von Usability in der ISO Norm 9241 liefert keine direkten Kenngrößen für eine Evaluation. Die 3 beschriebenen Qualitätsmerkmale „effektiv“, „effizient“ und „zufrieden stellend“ sind dafür zu weitläufig interpretierbar. In [SP04] werden daher 5 praktisch besser anwendbare Kriterien vorgestellt, die sich vor allem auf Effizienz und NutzerInnenzufriedenheit der Applikation konzentrieren. Vorraussetzung für die Anwendung dieser Kriterien ist die Kenntnis der User sowie eine Menge von typischen Anwendungsfällen. Dann können die Kriterien, für jeden Anwendungsfall und für jede Usergruppe, möglichst objektiv und genau festgestellt werden.

1. *Lernaufwand.* Wie lange brauchen die User, um die Bedienung der Applikation so zu erlernen, dass die gestellten Aufgaben durchführbar sind?
2. *Geschwindigkeit.* Wie schnell führen die User die typischen Anwendungsfälle aus?

3. *Fehlerrate der User.* Wie viele und was für Fehler machen die User bei Ausführung der typischen Anwendungsfälle? Dieser Punkt kann in die Geschwindigkeit bis zur erfolgreichen Ausführung des Anwendungsfalles einfließen, sollte aber auf jeden Fall auch separat betrachtet werden.
4. *Erhaltung des Erlernten.* Wie gut behalten die User das Erlernte im Gedächtnis? Die Zeitspanne (Stunden, Tage, Wochen) sowie die Häufigkeit der Nutzung sollen mit einbezogen werden.
5. *Subjektive Zufriedenheit.* Wie ist die Einstellung der User zu der Applikation bzw. ihrer Handhabung? Die Betrachtung derartiger subjektiver Merkmale erfordert den Einsatz von qualitativen Methoden, wie z.B. Interviews oder Beobachtungen.

Die Kriterien beeinflussen sich in ihrer Umsetzbarkeit wechselseitig, so dass z.B. die schnelle Durchführbarkeit von Anwendungsfällen, mit Hilfe komplexer Macros, zu Lasten des Lernaufwandes geht. Daher sollte die Betrachtung der Ergebnisse auf Basis der Anforderungsanalyse durchgeführt werden, damit der spezielle Kontext der Applikation in die Evaluation einfließen kann. Es ist z.B. in medizinischen Applikationen durchaus wünschenswert, die Fehlerrate auf Kosten der Geschwindigkeit zu minimieren.

Eine derartige Evaluation soll noch vor der Implementierung, idealerweise während des Designprozesses, durchgeführt werden. Dazu werden Prototypen unterschiedlicher Detailtreue erstellt und mit den Usern erprobt. Gerade bei Webapplikationen ist die Herstellung von realistischen Prototypen, mit detailgetreuer Interaktion, bei niedrigem Ressourcenaufwand gut möglich. Nach der Fertigstellung des finalen Designs, eventuell mit mehreren Redesign-Zwischenschritten, folgt dann die Implementierung.

2.3 Fokus auf die User

Die Planung von Produkten mit hoher Usability rückt die User in den Fokus der Aufmerksamkeit. Bei der Entwicklung von Applikationen, die auf die Wünsche und Bedürfnisse von speziellen Usergruppen zugeschnitten sind, spielt die Erforschung derselben eine zentrale Rolle. Wenn man sich die zahlreichen unterschiedlichen Eigenschaften, Fähigkeiten, Kulturen und Persönlichkeiten des Menschen vor Augen führt, wird die Herausforderung dieser Aufgabe offensichtlich. Zur Illustration ein Beispiel aus [SP04, S.24]:

> „A right-handed female designer in India with computer training and a desire for rapid interaction using densely packed screens may have a hard time developing a succesfull interface for left handed male artists in France with a more leisurely and freeform work style.“

Dieser Fall mag extrem erscheinen, ist jedoch in der modernen, globalisierten Welt nicht undenkbar. Und gerade Webapplikationen wollen und können von ihrer globalen Verfügbarkeit profitieren. Das Internet bietet einen einfachen und kostengünstigen Zugang zu einer der weltweit größten aber auch heterogensten Usergruppe. Die Entwicklung von erfolgreichen Webapplikationen mit hoher Usability setzt also die Betrachtung der User vorraus. Unterscheidungsmerkmale, derer sich die EntwicklerInnen bewußt sein sollen, inkludieren:

Physische Einschränkungen. In [CR03] wird eine effiziente Verwendung der Maus vor allem in Abhängigkeit der Einschränkungen des menschlichen Bewegungsapparates dargestellt. Die Maus wird entweder feinmotorisch mit den Fingermuskeln (für kurze, präzise Bewegungen) oder grobmotorisch mit der Armmuskulatur (für lange Bewegungen) bewegt. Das Umschalten zwischen diesen Bewegungsmodi erfordert Konzentration und erzeugt Ablenkung. Als Beispiel wird, der in Webapplikationen fast immer vorhandene, Scrollbalken angeführt: wenn die Scrollrichtung gewechseln werden soll, muss zuerst eine grobmotorische Bewegung an das andere Ende des Balken durchgeführt werden, um den dort positionierten kleinen Knopf feinmotorisch betätigen zu können. Dies erfordert Konzentration und lenkt von der eigentlich durchgeführten Tätigkeit (einen Text lesen, Daten eingeben etc.) ab.

Das Wahrnehmungs- und Erkenntnisvermögen. Die Bedienung einer Applikation erfolgt über den Austausch von sensorischen Daten. Die User verarbeiten die, unter anderem, über Bildschirm und Lautsprecher gelieferten audiovisuellen Reize und senden mittels Tastatur und Maus Steuersignale zurück. Die Interpretation der Daten sowie die darauf basierenden Entscheidungen sind von User zu User unterschiedlich. Ein Beispiel für das Anpassen des Informationsflusses an das Wahrnehmungs- und Erkenntnisvermögen der User ist die Verwendung von speziellen Visualisierungstechniken. Durch eine derartige Aufbereitung von großen und multidimensionalen Datenmengen sind die User (bzw. ihr Wahrnehmungs- und Erkenntnisapparat) imstande, diese deutlich schneller zu analysieren und bewerten.

Die Persönlichkeit. Das Userverhalten variiert infolge der individuellen Persönlichkeit des Menschen. Vorsichtige, zurückhaltende Charaktere profitieren gerne von einer umfassenden Hilfefunktion, während intuitive, spontane Typen eine selbsterklärende Bedienung bevorzugen. Die Klassifizierung der Persönlichkeitsmerkmale einer speziellen Usergruppe ist allerdings schwierig und umstritten. Nennenswerte Methoden sind hier der Myers-Briggs Type Indicator (MBTI) [Kei98] oder der Big Five Test [HH95], die beide der Persönlichkeitspsychologie entstammen. Für die EntwicklerInnen lässt sich jedoch festhalten, dass eine Untersuchung des Userverhaltens Ergebnisse bringen kann, die in Zusammenhang mit bestimmten Persönlichkeitsmerkmalen stehen. Beispielsweise sortieren manche User ihre E-Mails gerne in Ordner, während andere ordnerlos arbeiten, aber die E-Mails gefiltert anzeigen.

Kulturelle Besonderheiten. Der Lebens- und Arbeitsalltag der User ist geprägt durch ihre kulturelle Identität. Multikulturelle Applikationen - im speziellen Webapplikationen - stehen damit vor der Herausforderung auf die kulturell bedingten individuellen Unterschiede der User Rücksicht zu nehmen. Ein oftgenanntes Beispiel ist die Leserichtung (in Europa von links nach rechts, in China und Japan umgekehrt), welche die Wahrnehmung von Inhalten beeinflusst. Dies stellt jedoch nur die Spitze des Eisberges dar: in [MAC+00] werden Kriterien vorgestellt, mit denen die Eigenarten einer Reihe von Kulturen analysiert sowie deren Einfluss auf die Handhabung von Webapplikationen bewertet werden.

Die Betrachtung und Rücksichtnahme dieser Faktoren ist vor allem bei jenen Komponenten der Applikation wichtig, die in direktem Kontakt mit den Usern stehen. Das Speichern von Informationen in eine Datenbank ist im Normalfall unabhängig von den Persönlichkeitsmerkmalen der User. Im Gegensatz dazu stehen jedoch die Benutzeroberfläche - das Interface - und das Applikationsverhalten - die Interaktion -, die infolge ihrer direkten Kommunikation mit den Usern auf diese bestmöglich eingehen sollen. Das Ziel der EntwicklerInnen, das Versichern eines hohen Maßes an Usability, hängt von einem optimalen Design dieser beiden Komponenten ab. Interaktion und Interfaces bestimmen dabei nicht nur das Aussehen und das Verhalten einer Applikation, sondern erzeugen bei den Usern ein komplexes Anwendungserlebnis. Ist dieses positiv, beweist das den Erfolg der Entwicklung.

3 Interface & Interaktion

Der englische Begriff Interface lässt sich generell mit Schnittstelle in den deutschen Sprachraum übersetzen. Im Kontext der Kommunikation zwischen User und Applikation bzw. der Schnittstelle zwischen Mensch und Maschine, wird dieser Begriff dann zu User Interface erweitert - übersetzt spricht man von der Benutzeroberfläche. Der im Internet verfügbare Thesaurus der Universität Princeton [MF07] definiert User Interface als

> „S: (n) interface, user interface ((computer science) a program that controls a display for the user (usually on a computer monitor) and that allows the user to interact with the system)“

Es ist also jener Teil einer Applikation, der für die Anzeige der unterschiedlichen grafischen Elemente verantwortlich ist, über welche die Kommunikation mit den Usern stattfindet. Als solches liegt das User Interface an der Oberfläche der Applikation und erlaubt die Benutzung derselben - daher der deutsche Begriff Benutzeroberfläche. Aus der Definition geht ebenfalls hervor, dass jede Interaktion zwischen den Usern und der Applikation über diese Komponente erfolgt.

Die Gestaltung des Interface gliedert Cooper [CR03] in Graphical Design und Visual Interface Design:

Eines der Ziele des *Graphical Designers* ist ein Interface, das die User auf einer visuell ästhetischen Ebene anspricht. Darunter fallen Elemente wie Schriftgröße und -art, Farben, Logos und vieles mehr. Darüber hinaus versucht er, die Handhabung der Benutzeroberfläche in die vorhandenen visuellen Objekte zu kodieren, um so eine intuitive Verwendung zu ermöglichen. Zum Beispiel vermittelt die grafische Repräsentation eines Knopfes, der sich in seiner Höhe von der des Hintergrundes abhebt, die intuitiv erfassbare Anweisung, diesen zu drücken.

Der *Visual Interface Designer* setzt sich hingegen mehr mit den dynamischen Aspekten des Interface auseinander. Was passiert, wenn der eben erwähnte Knopf gedrückt wird? Wie wird die damit stattfindende Interaktion am besten visualisiert? Wie können die User die Reaktion der Applikation (auf den Knopfdruck) bestmöglich erkennen? Er sorgt damit dafür, dass die visuelle Struktur des Interface die Interaktion - also das wechselseitige Verhalten - zwischen User und Applikation bestmöglich repräsentiert. Anders ausgedrückt: der Visual Interface Designer gestaltet jenen Teil der Schnittstelle, über den die Interaktion zwischen den Usern und der Applikation stattfindet.

Das Interesse dieser Arbeit gilt, wie der Titel schon ausdrückt, dem Interaktionsdesign und nicht dem Interfacedesign. Aus dem, eben vorgestellten, Aufgabenbereich des Visual Interface Designer geht allerdings hervor, dass die Gestaltung des Interface eng mit der Interaktion einer Applikation zusammenhängt. Daher beinhaltet der Begriff Interaktionsdesign im weiteren auch immer das Design der relevanten Komponenten des Interface.

3.1 Interaktion

Die Handhabung der Applikation durch die User ist motiviert von der Erfüllung einer Aufgabe, wie z.B. das Schreiben einer E-Mail. Bis zu diesem definierten Ziel befinden sich User und Applikation in einem interaktiven Prozess. Crawford erklärt diese Interaktion in [Cra02, S.5] als

> „interaction: a cyclic process in which two actors alternately listen, think and speak.“

Die Interaktion zwischen BenutzerInnen und Applikation entsteht also durch die wechselseitige Kommunikation und ist ein permanenter Dialog mit Feedback und Kontrollmöglichkeiten. Die Kommunikation findet dabei über alle vorhandenen und verwendbaren Kanäle statt, wie z.B. Bildschirm, Lautsprecher, Tastatur, Maus. Der Dialog besteht aus der wechselseitigen Abfolge von Aktion und Reaktion der HandlungspartnerInnen, die ein bestimmtes Ziel verfolgen. Das Feedback während des Dialoges erlaubt den Usern, stetig neue Entscheidungen zu treffen, mit denen sie gegebenenfalls, über die vorhandenen Kontrollmöglichkeiten, den Ablauf der Applikation beeinflussen.

Durch das bedachte und userzentrierte Design dieser Interaktion entsteht eine hohe Usability, da das Verhalten der Applikation an die Bedürfnisse und Anforderungen der User angepaßt wird. In [CR03, S.xxxix] wird Interaction Design definiert als

> „Interaction Design is the definition and design of the behavior of artifacts, environments, and systems, as well as the formal elements that communicate that behavior."

Das Verhalten der Applikation sowie die Teile des Interface, die dieses Verhalten an die User kommunizieren, stehen im Zentrum des Interaktionsdesign. Die DesignerInnen versuchen die Interaktion vorherzusehen und zu planen, um die Applikation bestmöglich auf den Dialog mit den Usern vorzubereiten. Während des Dialoges bzw. der Interaktion sorgt das geplante Verhalten der Applikation dafür, dass die User die erhaltenen Informationen bestmöglich wahrnehmen und interpretieren können. Anhand der oben erwähnten Aufgabe - das Schreiben einer E-Mail - lässt sich dies illustrieren: wenn jeder Buchstabe nicht sofort nach dem entsprechenden Tastendruck, sondern zufällig auftaucht, kann die Aufgabe kaum durchgeführt werden. Die Informationen werden zwar seitens der Applikation vollständig verarbeitet und wiedergegeben (möglicherweise auch ästhetisch ansprechend), allerdings ohne eine für die User nachvollziehbare logische Struktur. Das Verhalten der Applikation wäre somit unzumutbar.

Die Anwendungsmöglichkeiten moderner Computersysteme sind so vielfältig, wie es die Vorstellung des Menschen, der sie bedient, zulässt. Kombiniert man dies mit der Fülle an menschlichen Eigenschaften, wie in Kapitel 2.3 illustriert, wird deutlich, dass das Design der Interaktion, welche zwischen User und Applikation stattfindet, eine komplexe und anspruchsvolle Aufgabe darstellt. In der Praxis entstehen bei der Erfüllung dieser Aufgabe neue Erkenntnisse, die dokumentiert und publiziert werden. Dieses aus der Empirie kommende Wissen liegt z.B. in Form von Design Tipps wie in [Joh03] vor. Johnson beschreibt in diesem Werk 60 Probleme beim Design von Webseiten und Webapplikationen sowie dazugehörige Lösungen. Dieses sehr praxisnahe Wissen ist stark an den Kontext, in diesem Fall die entsprechende Website bzw. Webapplikation, gebunden. Das erleichtert zwar das Verständnis und gegebenfalls die Umsetzung, es schränkt jedoch die Wiederverwendbarkeit auf ähnliche Problemsituationen ein.

In den Ingenieurswissenschaften versucht man daher Lösungen für Probleme so weit zu abstrahieren, dass sie möglichst generell wiederverwendbar sind. Im Software Engi-

neering werden so genannte Entwurfsmuster [GHJ95], im englischen Design Patterns, erstellt, die auf viele ähnliche Problemstellungen anwendbar sind. Bei der Entwicklung von Webapplikationen arbeitet man z.B. nach dem Model-View-Controller Pattern [BMRS96]. Im Interaktionsdesign kann ebenfalls auf derartige Entwurfsmuster zurückgegriffen werden, allerdings mit gewissen Einschränkungen:

Michel Beaudouin-Lafon kritisiert in [BL04] das Fehlen einer soliden und allgemein anerkannten theoretischen Basis, als die Ursache für die schleppende Entwicklung von Innovationen im HCI Bereich. Ähnlich dazu erklärt Shneiderman in [SP04], unter Verweis auf [Car03], dass es in der HCI Forschung derzeit eine Vielzahl von unterschiedlichen Theorien gibt, die geprüft, bearbeitet, korrigiert und auch wieder verworfen werden. Diese Problematik lässt sich auf das relative junge Alter - ungefähr 25 Jahre - des Wissenschaftszweiges sowie der Komplexität des Forschungsbereiches zurückführen.

Die Verwendung von Entwurfsmustern wird also erschwert, da keine allgemeine theoretische Basis deren Wissen formalisiert, verbindet und erklärt. Beaudouin-Lafon definiert daher den Begriff des Interaktionsmodells, das verschiedene Entwurfsmuster unterschiedlicher Abstraktionsgrade - er spricht von Interaktionstechniken - kombiniert. Die Definition eines solchen Interaktionsmodells enthält Eigenschaften, welche ein Formalisieren des darin enthaltenen Wissens erlauben. Dadurch können die enthaltenen Interaktionstechniken in Relation gesetzt und verglichen werden. Im folgenden Kapitel 3.2 wird die Arbeit von Beaudouin-Lafon erläutert und der Begriff des Interaktionsmodells erklärt.

Damit stehen die InteraktionsdesignerInnen noch vor der Herausforderung, aus der Fülle an vorhandenem Wissen im HCI Bereich jene Entwurfsmuster auszuwählen, welche aufgrund ihrer Eigenschaften die Anforderungen der zu entwickelnden Applikation bestmöglich erfüllen. Dazu wird auf das Wissen in den Arbeiten von Cooper und Shneiderman zurückgegriffen. Kapitel 3.3 und 3.4 beschreiben deren Vorgehensweise bei der Kategorisierung und Klassifizierung von Entwurfsmustern für Interaktionsdesign.

3.2 Beaudouin-Lafon: Paradigms, Models

Beaudouin-Lafon identifiziert in [BL04] 2 Ebenen, auf denen Interaktion analysiert und gestaltet werden kann. *Interaction Paradigms* arbeiten auf einer höheren Abstraktionsebene und wollen das „Phänomen“ der Interaktion zwischen User und Computer konzep-

tionalisieren. *Interaction Models* beschäftigen sich hingegen mit praxisnahen Entscheidungen über den konkreten Ablauf der Interaktion.

Die *Interaction Paradigms* teilt er anhand des vorherrschenden Leitgedanken in 3 Kategorien, die auch im Fokus unterschiedlicher Forschungsrichtungen liegen, ein:

Computer-as-Tool betrachtet den Computer als hochentwickeltes Werkzeug, mit dem der Mensch seine Fähigkeiten erweitern kann - ähnlich der Erfindung des Rades, um schwere Lasten besser transportieren zu können. Klassische Interaktionsformen wie WIMP (Window, Icon, Menu, Pointer) und Direct Manipulation (Drag&Drop) fallen in diese Kategorie. Das Feld der Human Computer Interaction beschäftigt sich mit diesem Bereich.

Computer-as-Partner verfolgt die Verwirklichung einer anthropomorphen Kommunikation zwischen Mensch und Computer, wie z.B. bei der Spracherkennung oder in agentenbasierten Systemen. Diese Kategorie liegt im Forschungsbereich der künstlichen Intelligenz.

Computer-as-Medium sieht den Computer rein als Übertragungsmedium für zwischenmenschliche Kommunikation, wie z.B. bei E-Mail, Chat oder Videotelefonie. Der Forschungszweig Computer Supported Cooperative Work (CSCW) vertritt diesen Ansatz.

Beaudouin-Lafon ist der Meinung, dass diese 3 Paradigmen auf den beiden Eigenschaften, die den Mensch von anderen Lebensformen abgrenzen, aufbauen: die Fähigkeit zur Herstellung und Verwendung von Artefakten, wie in Computer-as-Tool und Computer-as-Medium verwendet, sowie die Fähigkeit zur Kommunikation mit Sprache, wie in Computer-as-Partner und Computer-as-Medium verwendet. Das für ihn als HCI Forscher relevante und interessante Paradigma ist Computer-as-Tool, in dem er neue Interaktionsformen schaffen will, die den Menschen mit seinen speziellen Eigenschaften bestmöglich unterstützen und verbessern.

In [BL00, S.446] definiert Beaudouin-Lafon den Begriff *Interaction Model* als

> „An interaction model is a a set of principles, rules and properties that guide the design of an interface. It describes how to combine interaction techniques in a meaningful and consistent way and defines the „look and feel“ of the interaction from the user's perspective.“

Ein Interaktionsmodell enthält demnach das praktisch anwendbare Wissen, das die In-

teraktionsdesignerInnen während der Entwicklung einer Applikation benötigen, um Entscheidungen zu treffen. Verschiedene Interaktionstechniken werden kombiniert, um den Usern ein optimales Anwendungserlebnis bieten zu können.

Die InteraktionsdesignerInnen können unterschiedliche Modelle anhand von 3 Eigenschaften evaluieren:

1. *Descriptive Power*: die Aussagekraft. Wie gut kann es existierende Benutzerschnittstellen beschreiben?
2. *Evaluative Power*: die Evaluierungsstärke. Wie gut hilft es bei der Auswahl von Designalternativen?
3. *Generative Power*: die Gestaltungsstärke. Wie gut hilft es neue Designs zu erstellen?

Der Abstraktionsgrad und die Zusammensetzung eines konkreten Interaktionsmodells sind variabel, sie wirken sich allerdings auf diese 3 Eigenschaften aus. Modelle mit hohem Abstraktionsgrad, wie z.B. Direct Manipulation [Shn97], haben eine hohe Aussagekraft und leiden unter niederiger Evaluierungs- und Gestaltungsstärke. Praxisnahe Modelle wie z.B. die Apple Human Interface Guidelines [Con06a] verlieren an Aussagekraft und Evaluierungsstärke, weisen aber eine hohe Gestaltungsstärke auf. Ein optimales Interaction Model zeichnet sich demnach durch die Balance zwischen den 3 Eigenschaften aus.

Veranschaulichen lässt sich dieses Konzept an einem von Beaudouin-Lafon erstellten Interaktionsmodell - der Instrumental Interaction [BL00]. Das Modell beschreibt, auf der Grundlage von Direct Manipulation, die Gestaltung von Instrumenten als Mediatoren zwischen User und Computer. Der Scrollbalken ist z.B. ein Instrument, mit dem die User den Bildschirminhalt bewegen können (es vermittelt damit wie ein Mediator die Interaktion). Diese Definition erlaubt das Beschreiben einer großen Menge an Interfaces und gibt dem Modell eine hohe Aussagekraft. Es evaluiert Instrumente anhand dreier Eigenschaften - der „degree of indirection“, „degree of integration“ und „degree of conformance“ - und sorgt damit für die nötige Evaluierungsstärke. Schließlich erhält das Modell durch die Betrachtung von Interfaces als Instrumente Gestaltungsstärke - es wird z.B. die Verwendung von Dialogfenstern, die per Definition keine Instrumente sind, vermieden.

3.3 Cooper: Principles, Patterns

Cooper schreibt in [CR03] über *Interaction Design Principles*, *Interaction Design Patterns* und *Design Imperatives*. Mit den *Design Imperatives* will Cooper dem Leser 4 generelle Richtlinien mitgeben, die in jedem Design auffindbar sein sollen:

1. *Ethical [considerate, helpful]*: das Wohlergehen der User berücksichtigen.
2. *Purposeful [useful, usable]*: die Ziele, Wünsche und Anforderungen der User verwirklichen.
3. *Pragmatic [viable, feasible]*: die technischen und systemischen Anforderungen berücksichtigen.
4. *Elegant [efficient, artful, affective]*: die Gebrauchstauglichkeit optimieren.

Interaction Design Principles sind für Cooper generell anwendbare Richtlininien, die den EntwicklerInnen helfen, Ziele, Wünsche und Einschränkungen der User in formale und strukturelle Anforderungen an das Interaktionsdesign umzusetzen. Sie sollen damit den Arbeitsaufwand für die DesignerInnen reduzieren und die Usability für die User optimieren. Zielbereiche der Verbesserung der Usability sind das bessere Verständnis der Applikationslogik, die visuelle Aufbereitung der Informationen, die Erlernbarkeit und die physische Bedienbarkeit.

Interaction Design Principles werden nach jener Ebene der Applikationsentwicklung, in der sie anwendbar sind, kategorisiert:

- *Conceptual-level*: was ist der Zweck der Applikation, wer verwendet sie und warum?
- *Interaction-level*: wie verhält sich die Applikation?
- *Interface-level*: wie ist die Benutzeroberfläche der Applikation?

In [CR03] finden sich Richtlinien für alle 3 Ebenen, der Fokus liegt jedoch auf der Interaktion. Cooper grenzt den Abstraktionsgrad der Interaction Design Principles durch den Vergleich mit Style Guides ab. Style Guides sind, nach seiner Definition, spezifische und detaillierte Anweisungen für das Design einer Applikation, wie z.B. die Anzahl der Schaltflächen in einem Dialogfenster. Interaction Design Principles sind dagegen abstrakter und adressieren komplexere Probleme.

Nach Cooper formalisieren *Interaction Design Patterns* das Wissen aus Entscheidungen, die während des Design Prozesses getroffen werden, zu allgemein anwendbaren Lösungsmustern. Diese Muster können dann bei zukünftigen, ähnlichen Problemstellungen wiederverwendet werden. Das reduziert nicht nur den Aufwand, sondern erlaubt auch den geplanten Einsatz von Interaktionstechniken, deren Qualität hinsichtlich der Usability schon bekannt ist. Das Erzeugen von NutzerInnenzufriedenheit - Cooper spricht vom „well-being" der User - muss auf jeden Fall das Ergebnis der Verwendung von Interaction Design Patterns sein. In [CR03, S.93] definiert er dies als den Unterschied zu Patterns aus dem Software Engineering („this human element" meint „well-being"):

> „It is this human element that differentiates interaction design patterns from engineering design patterns, whose sole concern is efficient reuse of code."

Wie schon bei den Principles teilt er auch die Patterns nach der Applikationsebene, in der sie angewendet werden, ein:

- *Postural*: diese Muster arbeiten auf dem Conceptual-level und beschäftigen sich mit dem Gesamteindruck, den die Applikation auf die User machen will. Dazu gehören z.B. Eindrücke wie unscheinbar, fordernd, grell, sanft etc.
- *Structural*: diese Muster beschäftigen sich mit der Struktur der Kommunikation zwischen User und Applikation. Das beeinhaltet sowohl den Informationsfluss ansich als auch die visuelle Aufbereitung in unterschiedliche Ansichten, Dialogfelder etc.
- *Behavioral*: diese Muster beschäftigen sich mit dem Verhalten der Applikation, das bestmöglich auf die Anforderungen der User abgestimmt sein soll. Je klarer die Arbeitsweise der Applikation sichtbar ist, desto leichter können die User sie verwenden.

3.4 Shneiderman: Guidelines, Principles, Theories

Shneiderman systematisiert in [SP04] das vorhandene Wissen nach dem Grad der Abstraktion in *Guidelines*, *Principles* und *Theories*.

Guidelines sind spezifische und praxisorientierte Richtlininen, die das Design der Applikation konsistent und für alle DesignerInnen verständlich festhalten. Sie sind das direkte

Ergebnis aus den Erfahrungen von schon abgeschlossenen, erfolgreichen Entwicklungsprozessen. Dieser enge Bezug zur Praxis ist sowohl Nach- als auch Vorteil: weicht die Problemsituation leicht von jener, aus der die Guideline entstanden ist, ab, ist die Anwendung schwierig, unmöglich oder das Ergebnis unvorhersehbar. Ist der Kontext jedoch ident, kann das Expertenwissen rasch und sicher eingesetzt werden. Guidelines helfen z.B. beim Design der Navigation, der Anzeige oder der Dateneingabe.

Principles formalisieren das praktische Wissen der Guidelines, um es aus dem spezifischen Kontext herauszulösen und allgemein anwendbar zu machen. Damit sind sie universeller einsetzbar und langlebiger, erfordern jedoch ein erhöhtes Maß an Aufwand für die Verwendung. Als Beispiel bringt Shneiderman das Principle „recognizing user diversity“ [SP04, S.66], also die Anforderung an die DesignerInnen, sich mit den speziellen Eigenschaften der User zu beschäftigen. Die Erfüllung dieses Principle in einem konkreten Design führt zuerst zu einem praktischen Mehraufwand, allerdings mit der Aussicht auf eine verbesserte, erfolgreichere Applikation. Zu den Principles gehören z.B. Interaktionsformen wie Direct Manipulation und Menu Selection.

Ausgehend von den Principles werden *Theorien* verfasst, die den höchsten Abstraktionsgrad innehaben. Shneiderman definiert die Anforderungen an eine Theorie mit [SP04, S.86]

> „However, a good theory should at least be understandable, produce similar conclusions for all who use it, and help to solve specific practical problems.“

Sie soll also verständlich, reproduzierbar und praktisch anwendbar sein. Als Eigenschaften einer Theorie beschreibt Shneiderman [SP04, S.82] „descriptive“, „explanatory“ oder „predictive“, also beschreibend, erklärend oder vorhersagend. Beschreibende und erklärende Theorien helfen ein gemeinsames und konsistentes Verständnis für die Terminologie, also die verwendeten Elemente und ihre Relation zueinander, einer Wissensbasis zu schaffen. Vorhersagende Theorien erlauben Evaluation und Vergleich von alternativen Designs vor ihrer tatsächlichen Umsetzung.

Er weist darauf hin, dass im HCI Bereich derzeit eine Vielzahl von unterschiedlichen Theorien existieren, die versuchen, sich bei den AnwenderInnen (ForscherInnen, EntwicklerInnen etc.) durchzusetzen. Aufgrund der Größe und Komplexität des Wissensbereiches ist es für ihn fraglich, ob es jemals eine einheitliche, solide und allgemein anerkannte Theorie geben wird.

4 Entwicklung des Interaktionsmodells

Anhand der in den vorherigen Kapiteln beschriebenen Arbeiten von Shneiderman und Cooper wird deutlich, dass der Abstraktionsgrad einer Interaktionstechnik darüber entscheidet, ob und wie gut sie wiederverwendet werden kann. Das in diesem Kapitel entwickelte Interaktionsmodell soll sowohl praktisch umsetzbar als auch wiederverwendbar sein. Daher entscheidet sich der Autor für Interaktionstechniken von mittlerem und hohem Abstraktionsgrad: Principles (nach Shneiderman) und Interaction Patterns (nach Cooper). Diese werden innerhalb eines Interaktionsmodells kombiniert und anhand dessen Eigenschaften - Aussagekraft, Evaluierungsstärke, Gestaltungsstärke - bewertet.

Das Internet wurde ursprünglich als dezentrale Wissensdatenbank entworfen: User sollen jederzeit und von überall auf untereinander verknüpfte Dokumente zugreifen können. Diese Dokumente heißen im englischen Web Pages, zu deutsch Webseiten, und sind über Links miteinander verknüpft. Nach Cooper [CR03] ist die Verwendung des Internet als Wissensdatenbank informationszentriert: die User wollen hauptsächlich Webseiten suchen, diese anzeigen und Links verfolgen. Mehrere Webseiten werden daher in einer, so genannten, Website organisiert, z.B. in einer Hierarchie und durch ein Navigationsmodell miteinander verknüpft. Suchmaschienen bieten die Möglichkeit das Navigationsmodell zu umgehen, indem Webseiten mit spezifischen Informationen direkt gefunden und angesteuert werden können. Bei der Entwicklung einer Website liegen die primären Anforderungen demzufolge in den Bereichen Information (die Inhalte der Webseiten) und Organisation (die Struktur der Information). Die Navigation ist das einzige, mäßig interaktive Element.

Webapplikationen weisen hingegen, infolge ihrer transaktionsbasierten Arbeitsweise und der dynamischen Inhalte, eine hohe Interaktivität auf. Da das Verhalten einer Webapplikation zur Laufzeit von den Usern dynamisch verändert werden kann, spricht man von

der transaktionsbasierten Arbeitsweise - jede Transaktion erzeugt eine Veränderung im Programmablauf. Mit dieser Veränderbarkeit ergeben sich in weiterer Folge dynamische Inhalte, die je nach Nutzungskontext unterschiedlich behandelt werden müssen. Das Verhalten, die Interaktion mit den Usern, steht daher im Vordergrund bei der Entwicklung von Webapplikationen. Je besser die User die Arbeitsweise und Informationen - das Verhalten - der Webapplikation verstehen, desto höher ist die Gebrauchstauglichkeit - die Usability.

Nach Beaudouin-Lafon kann ein Interaktionsmodell mit hoher Aussagekraft eine signifikante Bandbreite von Interfaces bzw. Interaktionselementen erfassen und beschreiben. Eine Methode dafür ist das Verwenden einer Taxonomie, die bei Shneiderman [SP04, S.85] wie folgt definiert ist:

> „A taxonomy imposes order by classifying a complex set of phenomena into understandable categories."

Die Taxonomie ordnet also komplexe Objekte anhand von bestimmten, definierten Kriterien in wohlunterscheidbare Kategorien. Diese Einteilung soll gut verständlich und nachvollziehbar sein. Da das Ziel dieser Arbeit ein Interaktionsmodell für Webapplikationen ist, benötigt es ausreichende Aussagekraft, um typische Interaktionselemente in diesem Bereich beschreiben zu können.

Es gibt jedoch noch keine fundierte und allgemein anerkannte Taxonomie, die interaktive Elemente von Webapplikationen beschreibt. Daher hat der Autor dieser Arbeit die Inhalte von [SJK04], [SP04], [CR03] und [KPRR03] studiert und auf Gemeinsamkeiten geprüft. Es stellt sich heraus, dass jede dieser Arbeiten Webapplikationen in spezifische Komponenten aufteilt, die sinngemäß äquivalent sind, allerdings unterschiedlich bezeichnet werden. Innerhalb dieser Schnittmenge gibt es wiederum 5 Komponenten, die dem Interaktionsdesign per Definition aus Kapitel 3.1 zugeordnet werden können. Das entwickelte Interaktionsmodell für Webapplikationen verwendet daher eine Taxonomie, bestehend aus den im weiteren beschriebenen 5 Komponenten.

Per Definition nach Beaudouin-Lafon besitzt ein Interaktionsmodell auch Evaluierungs- und Gestaltungsstärke (*Evaluative Power* und *Generative Power*). Mit den in den 5 Kapiteln vorgestellten Interaktionstechniken wird aufgrund des gewählten Abstraktionsgrades beides erreicht: sie bieten sowohl die Möglichkeit, Vor- und Nachteile unterschiedlicher Designs zu vergleichen, als auch die konkrete Gestaltung neuer Designs.

4.1 Visual Interface

Der Begriff Visual Interface Design wurde schon in Kapitel 3 kurz beschrieben, um den Aufgabenbereich der InteraktionsdesignerInnen abzugrenzen. Nach Cooper verfügt das menschliche Gehirn über hochentwickelte Mechanismen, um visuelle Muster rapide zu erkennen und zu verarbeiten. Beispielsweise können Gesichtsmerkmale innerhalb von Sekundenbruchteilen ausgewertet und mit einem Namen verknüpft werden. Das Ziel des Visual Interface ist ähnlich: Applikationslogik und -verhalten sollen für die User rasch und leicht verständlich sein. Dazu müssen die relevanten Elemente der Benutzeroberfläche in ihrer Struktur und Form an die Wahrnehmung des Menschen angepasst werden. Ist die Anpassung erfolgreich, kann das menschliche Gehirn die wahrgenommenen Muster ebenso rasch und effizient verarbeiten, wie die Gesichtsmerkmale aus dem obigen Beispiel.

Bereiche, auf die sich das Visual Interface Design bezieht, sind das Seitenlayout, die Anordnung (Organisation) der Elemente darin und die klar struktierte Kommunikation des Applikationszustandes. Die Information in diesen Bereichen ist nicht textbasiert, i.e. geht es um die korrekte Anordung von Schaltflächen, Symbolen, Grafiken, etc. In XHTML Seiten wird dies unter anderem mit Input Tags, Image Tags, Blockelementen (Div Tags), Tabellen oder speziell implementierten JavaScript Widgets umgesetzt (Abb. 4.1).

In [SP04, S.501] erklärt Shneiderman das Konzept der „layout appropriateness“, um herauszufinden, ob die Anordnung der interaktiven Elemente die User in der Handhabung optimal unterstützt. Dabei wird der Weg, den die User mit dem Mauszeiger zurücklegen müssen, um typische Anwendungsfälle durchzuführen, betrachtet. Ist dieser Zeigeweg nicht minimal und erfordert häufiges Springen zwischen weit entfernten Bereichen der Benutzeroberfläche, leidet die Usability.

Cooper erläutert in [CR03] eine Reihe von Interaction Patterns, deren Anwendung die Usability des Visual Interface positiv beeinflusst:

Vermeiden von „Visual Noise“. Die grafischen Elemente des Visual Interface sollen auf das Notwendigste reduziert werden, um die Kommunikation mit den Usern zu erleichtern. Ein Überfluss an Schaltflächen, Symbolen, Grafiken etc., mit denen die Applikation ihre Logik kommuniziert, erhöht das Risiko, dass die User etwas übersehen oder ausblenden.

Kontrast & Gruppen. Durch wohlüberlegtes Kontrastieren und Gruppieren von grafischen, interaktiven Elementen entstehen visuelle Muster, die die User effizient erkennen und leicht im Gedächtnis behalten können. Durch Kontrast, also spezifische Unterscheidungsmerkmale, sollen sich nichtinteraktive Elemente der Benutzeroberfläche, wie z.B. Logos von manipulierbaren, interaktiven, Elementen wie z.B. Buttons, abheben. Cooper unterscheidet 4 Arten von Kontrast: Dimensional Contrast gibt ein dreidimensionales Aussehen, welches gleichzeitig intuitiv die Handhabung vermittelt (z.B. bei einem Button). Tonal Contrast hebt Elemente durch Veränderungen ihrer farblichen Erscheinung (Farbton, Sättigung, Helligkeit) von anderen ab. Spatial Contrast verändert die zweidimensionale räumliche Anordnung, z.B. von links nach rechts, und äußere Form, z.B. eckig oder rund, von grafischen Elementen. Layering bezieht sich ebenfalls auf die räumliche Anordung, allerdings in der dritten Dimension - durch Verwendung der z-Achse - und erlaubt damit das Überlappen, Hin- und Wegschieben von grafischen Elementen. Durch Kontrastierung entsteht in den meisten Fällen ein gleichzeitiges Gruppieren: diese Gruppen unterscheiden interaktive Elemente anhand ihrer Funktionen.

Konsistentes Ausrichten von Elementen einer Gruppe. Innerhalb einer Gruppe sollen gleiche interaktiven Elemente dieselbe Ausrichtung, z.B. linksbündig, besitzen. Die Gruppen selbst sollen in Gitternetzstrukturen eingebettet werden, geordnet nach ihrer Wichtigkeit. Dies vermittelt den Usern eine aufgeräumte Benutzeroberfläche, die sie besser verstehen können.

Visualisieren von Applikationsverhalten. Ergänzend zu der textuellen Beschreibung soll das Applikationsverhalten visualisiert werden, um das Verstehen seitens der User zu erleichtern.

Verwenden von idiomatischen Interfaces. Der im Internet verfügbare Thesaurus der Universität Princeton [MF07] definiert Idiom mit

> „S: (n) idiom, idiomatic expression, phrasal idiom, set phrase, phrase (an expression whose meanings cannot be inferred from the meanings of the words that make it up)“

Die Bedeutung eines Idiomes ergibt sich also nicht aus den Bestandteilen an sich, sondern erst aus einer speziellen kulturellen Verwendung. Die Bedeutung ist damit nicht intuitiv oder implizit erfassbar, sondern muss erst erlernt werden. Nach Cooper ist der Großteil der heute verwendeteten User Interfaces idiomatisch, wie z.B. Links in Webapplikatio-

Abbildung 4.1: Teile des Visual Interface auf Netvibes.com

nen. Die Verwendung eines Links ist nicht intuitiv oder implizit erfassbar (außer die Bezeichnung des Links lautet „Klick mich mit der Maus an“), sondern muss erst erlernt werden. Ein gutes Idiom zeichnet sich dadurch aus, dass es, einmal erlernt, sofort im Gedächtnis bleibt und wiederverwendet werden kann.

4.2 Content Organization

Ebenso wie die leicht verständliche Kommunikation der Applikationslogik durch grafische Platzhalter, muß die Darstellung der textbasierten Information an die Wahrnehmung der User angepasst werden. Oft ist es nicht möglich, den Applikationszustand in Symbole oder Schaltflächen zu kodieren, dann bleibt die Ausgabe von Text. Die Anzeige von textbasierten Elementen soll klar strukturiert und organisiert sein. Das Lesen und Verarbeiten von Textinformation ist zwar langsam und erfordert gezielte Aufmerksamkeit, allerdings kann durch entsprechende Platzierung die Wahrnehmung der User gezielt (auf den Text) gelenkt werden. In XHTML Seiten werden Textinformationen in Tabellen, Inlineelementen (z.B. Span Tags) und Blockelementen (Div Tags) organisiert (Abb. 4.2).

Textelemente werden, ebenso wie grafische Elemente, von den Usern als visuelle Struktur erkannt, allerdings wird für das Erkennen und Lesen der Bedeutung mehr Zeit benötigt. Daher ist es sinnvoll, Objekte grafisch zu repräsentieren, mit einer detailierten

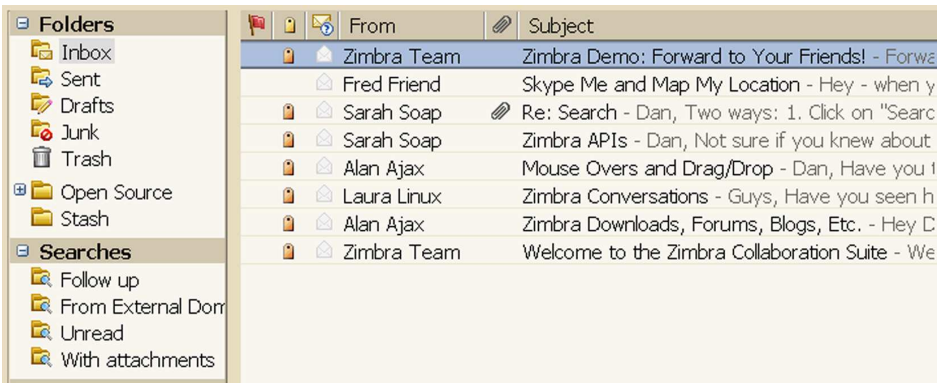

Abbildung 4.2: Textbasierte Information auf Zimbra.com

textuellen Beschreibung. Cooper spricht von „visually show what, textually show which“ [CR03, S.242]. Da vor allem des Lesen von Text langsam und anstrengend ist, soll die Ausdrucksweise kurz und klar sein. Außerdem wird von der Verwendung von Abkürzungen, ausgenommen der standardisierten, abgeraten. In grafischer Hinsicht soll sich der Text deutlich vom Hintergrund abheben und eine Schriftgröße von minimal 11 Punkt besitzen. Shneiderman betont vor allem das effiziente Anordnen und Gruppieren von Textelementen auf der Benutzeroberfläche. Er verweist auf die Studie [IH02], derzufolge die User eine spaltenorientierte Anordnung bevorzugen. Außerdem sollen serifenlose Fonts sowie farblich markierten Überschriften verwendet werden.

4.3 Navigation

Die User verwenden eine Webapplikation durch Interaktion mit unterschiedlichen Webseiten. Auf diesen Seiten ist die gesamte Funktionalität und entsprechende Informationen abgebildet. Die Navigation durch diese Menge von Seiten ist ein kritisches Element, das die Usability stark beeinflusst. Die User müssen effizient durch die Webapplikation navigieren können, während sie dabei die Orientierung behalten - also was gerade passiert und wie anderen Seiten angesteuert werden können. Die Visualisierung der Navigationsinformationen erfolgt z.B. mittels Menüs, Breadcrumbs [Hud04] oder Signposts [CR03]. Innerhalb dieser Visualisierungen verweisen Links auf die URL - die eindeutige Adresse innerhalb des Internet - der entsprechenden XHTML Seite (Abb. 4.3).

Shneiderman verweist in [SP04] auf die Interaction Principles aus [SJK04], um die Usa-

bility der Navigation zu verbessern:

In Webapplikationen verteilt sich die Interaktion mit den Usern häufig auf mehrere, nacheinander angezeigte Webseiten. Innerhalb einer derartigen Sequenz sollen die User immer erkennen können, wie weit sie schon gekommen sind und was noch auf sie zukommt. Dieses Feedback soll auf der Benutzeroberfläche immer gut sichtbar und leicht verständlich sein. In Webapplikationen müssen zudem die externen Navigationsfunktionen des Browsers beachtet werden: die User können, über entsprechende Browserknöpfe, in der Seitenhistorie vor und zurück gehen und Lesezeichen setzen. Diese Navigationsmöglichkeiten sollen erhalten bleiben, was beispielsweise bei der Verwendung von Popup Fenstern nicht der Fall ist.

Die Navigationselemente sollen nach ihrer Funktion gruppiert und entsprechend ihrer Wichtigkeit grafisch aufbereitet werden. Diese Gruppen sollen, wie auch schon in Kapitel 4.1 erläutert, applikationsweit konsistent sein. In empirischen Studien hat sich gezeigt, dass die User eine Positionierung der Navigationsgruppen auf der linke Seite der Benutzeroberfläche bevorzugen. Ist die Navigation hierarchisch abgebildet, soll für informationszentrierte Bereiche ein verschachteltes Menü verwendet werden. Transaktionale Bereiche hingegen sollen mit nicht-hierarchischen, simultan einstellbaren Navigationselementen ansteuerbar sein.

Ein beliebtes Idiom in Webapplikationen ist die Verwendung von Karteikartenreitern zur Navigation: diese sollen eine klar verständliche Beschriftung haben, am oberen Ende der Webseiten positioniert werden und realistisch visualisiert werden (i.e. sind aktive Karteikarten im Vordergrund und überlappen die Inaktiven). Durch Setzen des „title“ Attributes für beliebige XHTML Elemente sollen Tooltips eingeblendet werden. Von der Verwendung von Breadcrumbs zur Navigation wird abgeraten, da empirische Studien gezeigt haben, dass diese nicht von allen Usern effektiv verwendet werden.

Cooper betont, dass die Notwendigkeit einer Navigation für die User primär eine Belastung darstellt. Daher ist der einfachste Weg die Navigation zu verbessern, die Anzahl der ansteuerbaren Seiten zu reduzieren. Als permanente Orientierungsmöglichkeit sollen Signposts verwendet werden - eine primäre applikationsweite Navigationsgruppe, wie z.B. Karteikartenreiter. Zwischen der Repräsentierung eines Navigationselementes und seiner Funktion soll eine klare Verknüpfung erkennbar sein: wenn beispielsweise eine Karteikarte namens Kontakt angeklickt wird, soll auch tatsächlich auf eine Webseite mit dieser

Abbildung 4.3: Die primäre Navigation auf Amazon.de

Information navigiert werden. Wichtige und häufig genutzte Funktionen sollen an die Oberfläche der Navigation, während Expertenfunktionen in das Innere platziert werden können. Generell sollen zu tiefe und komplizierte Navigationshierarchien vermieden werden, z.B. durch eine Beschränkung auf 2 bis 3 Ebenen.

4.4 Direct Manipulation Objects

Nach Shneiderman [Shn97] besitzen Direct Manipulation Objects 3 Charakteristika:

1. *visuell repräsentiert und manipulierbar*: das Objekt wird durch ein grafisches Element visualisiert und kann über dasselbe Element manipuliert werden. Beispielsweise wird ein Dokument mit dem Idiom einer Papierrolle repräsentiert.
2. *manipulierbar durch eine physische Aktion*: das Objekt wird durch die physische Interaktion mit einem Eingabegerät manipuliert. Beispielsweise wird das Dokument gelöscht, indem die Papierrolle mit der Maus angeklickt wird.
3. *sofort sichtbare Auswirkungen*: die Auswirkungen einer Manipulation des Objektes werden sofort visualisiert. Beispielsweise verschwindet die Papierrolle sofort nach dem Löschen.

Nachdem aktuelle Browser die Standards Javascript und DOM standardkonform umsetzen, können Direct Manipulation Objects in Webapplikationen eingesetzt werden. In XHTML Seiten werden grafische Elemente durch Einsatz von JavaScript Programmcode zu Direct Manipulation Objects (Abb. 4.4).

Shneiderman erklärt in [SP04] weiters folgende Vorteile von Direct Manipulation:

- Sichtbarkeit von Objekten und zugehörigen Aktionen.
- Rapide, reversible, schrittweise Aktionen.
- Ersetzen von Textkommandos mit einer Zeigeaktion auf das Objekt.

Durch diese ist die Interaktionstechnik für AnfängerInnen rasch und leicht erlernbar, während ExpertInnen dauerhaft effizient arbeiten können. Das permanente visuelle Feedback zeigt den Usern unmittelbar, was ihre Aktionen bewirken. Dies resultiert in einem starken Kontrollgefühl, welches wiederum das Selbstvertrauen und damit das Anwendungserlebnis im Allgemeinen positiv beeinflusst. Nach [CR03, S.264] formulieren dies die Apple Styleguides mit „Users want to feel that they are in charge of the computer's activities". Durch die Sichtbarkeit und leichte Reversierbarkeit der Aktionen haben die User zudem weniger Angst davor, Fehler zu machen.

Direct Manipulation hat jedoch auch einige Nachteile [SP04]: das visuelle Manipulieren von Objekten benötigt fast immer sehr viel Anzeigefläche, was dazu führen kann, dass der Bildschirminhalt mit Hilfe der Scrollbar bewegt werden muss. Sollten dabei wichtige Informationen verloren gehen, rät Shneiderman auf die visuelle Repräsentation zu verzichten und eine dichtere, beispielsweise textbasierte, Darstellung zu wählen [SP04]. Das Erlernen der Bedienung ist anfangs zwar schnell und einfach, allerdings nicht intuitiv - die User benötigen also jemanden, der die Interaktionstechnik vorzeigt bzw. beibringt. Die Basis für Direct Manipulation ist die Auswahl von geeigneten Metaphern und/oder Idiomen, um die Objekte visuell zu repräsentieren. Ist diese Darstellung schlecht gewählt, wird die Usability stark in Mitleidenschaft gezogen.

Cooper teilt Direct Manipulation in 2 Unterkategorien [CR03]: Program Manipulation konzentriert sich auf die Steuerung der Applikation und des Interface, und soll für alle User erlernbar und meisterbar sein. Content Manipulation hingegen wird für das direkte Erstellen, Bearbeiten und Bewegen von Daten verwendet. Letzteres kann zwar von allen Usern durchgeführt werden, allerdings ist das Ergebnis abhängig von den individuellen Fähigkeiten wie z.B. beim direkten Zeichnen mit der Maus in einem Zeichenprogramm. In dieser Arbeit liegt der Fokus auf Program Manipulation mit der Maus, welche wiederum von Cooper in Teilbereiche [CR03] gegliedert wird. Für das hier entwickelte Interaktionsmodell sind die beiden Bereiche Selection und Drag&Drop von Interesse:

Selection bezeichnet die Auswahl von Objekten durch Hinzeigen und Anklicken mit der Maus. Für die weitere Manipulation wird in Applikationen immer von einer Reihung Objekt - Verb ausgegangen: zuerst werden die Objekte gewählt und dann die entsprechend anzuwendende Aktion. Mehrfache Selektion von Objekten ist möglich durch Aufziehen eines Rahmens um mehrere, zusammengehörige Objekte, oder durch schrittweises Klicken auf Objekte mit gedrückter Strg-Taste (die sich auf der Tastatur befindet). Norma-

Abbildung 4.4: Direct Manipulation auf Netvibes.com

lerweise (bei nicht gedrückter Strg-Taste) deselektiert eine neue Auswahl ein eventuell bereits selektiertes Objekt. Ausgewählte Objekte sollen sich visuell abheben: die Farbe des Objektes sowie die des darin enthaltenen Textes ändern sich nach einem Schema, das applikationsweit konsistent ist. Wenn eine derartige farbliche Markierung nicht ausreicht, können selektierte Objekte auch mit zusätzlichen visuellen Merkmalen wie z.B. einem Rahmen versehen werden.

Drag&Drop ist ein Idiom für das Verschieben von Objekten auf der Benutzeroberfläche. Dabei wird das Objekt selektiert, bei gedrückter Maustaste über den Bildschirm bewegt und mit Loslassen der Maustaste abgesetzt. Die Bewegung ist gleichbedeutend mit einer Transformation, die auf das selektierte Objekt angewendet wird. Dabei repräsentiert der Zielbereich auf der Benutzeroberfläche, also der Bereich in dem das Objekt abgesetzt wird, die Funktion. Beispielsweise wird ein Objekt, das auf dem Papierkorb abgesetzt wird, gelöscht. Wichtig bei diesem Vorgang ist, dass jeder Schritt entsprechend des Applikationsverhaltens visualisiert wird: das Bewegen eines Objektes wird optimalerweise durch ein Anhaften desselben an dem Mauszeiger repräsentiert. Wird der Mauszeiger mit dem Objekt über einen passenden Zielbereich bewegt, ändert dieser seine Darstellung. Nach Loslassen des Objektes muss das Ergebnis der ausgeführten Funktion ebenfalls visualisiert werden, beispielsweise das Verschwinden des zu löschenden Objektes im Papierkorb.

4.5 Controls

In [CR03, S.337] sind Controls definiert als:

> „Controls are manipulable, self-contained screen objects that allow communication betweeen users and software."

Controls sind also all jene Objekte auf der Benutzeroberfläche, die eigenständig manipulierbar sind und der Kommunikation zwischen User und Webapplikation dienen. Man bezeichnet sie auch als Widgets, Gadgets oder Gizmos. Controls erlauben den Usern mit der Applikation zu interagieren und ihren Ablauf zu steuern. Cooper unterscheidet in [CR03] folgende Arten:

Imperative Controls werden verwendet, um eine Funktion der Applikation auszulösen. Dazu gehören Buttons (Knöpfe) und Butcons (halb Button, halb Icon, z.B. in der Toolbar des Browsers). Siehe Abbildungen 4.5b und 4.5c.

Selection Controls werden verwendet, um Menüoptionen oder Daten auszuwählen. Dazu gehören Checkboxen (Auswahlfelder mit Mehrfachselektion), Butcons mit Zustandsinformationen (z.B. gedrückter Knopf mit Icon), Flip-Flop Buttons (Knöpfe mit Zustand, durch Beschriftung visualisiert), Radio Buttons (Auswahlfelder mit Einfachselektion), Combutcons (Butcons mit Zustand und Drop-Down Menü), Picklists (Auswahllisten, Drop-Down Menüs), Comboboxes (Kombination zwischen Eingabefeld und Auswahlliste) und Tree Controls (hierarchische Baumlisten). Siehe Abbildung 4.5a.

Entry Controls werden verwendet, um Daten einzugeben. Dazu gehören Textedit Fields (Texteingabefelder), Spinners (Schalter), Gauges (Messleisten), Sliders (Schieber), Knobs (Drehknöpfe). Die Dateneingabe kann mit oder ohne Eingabebeschränkung ausgeführt sein. Siehe Abbildung 4.5d.

Display Controls werden verwendet, um visuelle Elemente der Applikation zu manipulieren. Dazu gehören Text Controls (Beschriftungen, Textanzeigen, Überschriften), Scrollbars, Splitters (Begrenzungen) und Drawers (ein- und ausblendbare Oberflächen).

Nach Cooper soll die Handhabung einer *Imperative Control*, wie z.B. das Drücken eines Button, durch eine entsprechend grafische Repräsentation intuitiv erfassbar sein. Welche Applikationsfunktion damit ausgelöst wird, ergibt sich aus dem Applikations-

zustand, der den Usern wohlbekannt sein muss. Falls ein derartiges Steuerungselement seine Handhabung und die daraus resultierende Applikationsfunktion nicht ausreichend kommunizieren kann, wie z.B. ein schlecht gewähltes Icon auf einem Butcon, werden die User es nicht verwenden. Sowohl Buttons als auch Butcons sind als XHTML Elemente vorhanden.

In XHTML sind nur folgende *Selection Controls* per W3 Standard definiert [W3C02]: Checkboxes, Radio Buttons und Picklists. Andere Steuerungselemente müssen erst mit JavaScript implementiert werden und werden daher aufgrund der angestrebten praktischen Umsetzbarkeit des Interaktionsmodells nicht behandelt. Nach Cooper brauchen Checkboxes immer einen eindeutig assoziierten Text, der kurz und verständlich (am besten ein Wort) die Auswahl beschreibt. Die Handhabung der Checkbox - Klicken für Aktiviern und erneutes Klicken für Deaktivieren - ist ein wohlbekanntes Idiom, das schnell erlernbar ist. Der Radio Button wird ähnlich gehandhabt, allerdings ausschließlich in Gruppen verwendet, in denen nur eine Einfachauswahl möglich ist. Gruppen von Radio Buttons erlauben zwar die simultane Darstellung aller Auswahloptionen, verbrauchen dabei aber sehr viel Platz. Daher empfiehlt Cooper ihre Verwendung nur in Bereichen der Applikation, in denen die Anzeige aller Auswahlmöglichkeiten unbedingt notwendig ist. Picklists erlauben die Auswahl einer endlichen Menge von Textelementen, die wiederum ein Kommando, ein Objekt oder ein Attribut repräsentieren. Falls die Höhe der Picklist nicht alle darin enthaltenen Elemente aufnehmen kann, wird ein horizontaler Scrollbalken angezeigt, mit dem die unsichtbaren Elemente in den sichtbaren Bereich bewegt werden können. Ein Element in einer Picklist wird durch Mausklick selektiert und deselektiert. Ist es eine Picklist mit Einfachauswahl, wird bei Selektion eines neuen Elements ein bereits selektiertes Element deselektiert. In einer Picklist mit Mehrfachauswahl können mehrere Elemente durch Anklicken mit gedrückter Strg Taste ausgewählt werden. Cooper rät von der Verwendung der Mehrfachselektion bei Picklists mit Scrollbars ab: entweder werden schon selektierte aber unsichtbare Elemente vergessen oder sie müssen erst gesucht werden, um sie zu deselektieren.

In XHTML ist nur eine *Entry Control* per W3 Standard definiert [W3C02]: das Texteingabefeld. Dessen Größe und Zeilenanzahl ist variabel - sowohl einfeld als auch mehrzeilige Varianten sind möglich. Eine Beschränkung der Eingabemöglichkeiten ist nur durch die Größe des Feldes möglich. Ansonsten können die User jedes beliebige Zeichen setzen. Damit ist die Applikation gezwungen die Eingabe der User zu validieren, bevor sie verarbeitet wird. Dies ist prinzipiell ein Dilemma: die User können hinweislos Daten eingeben,

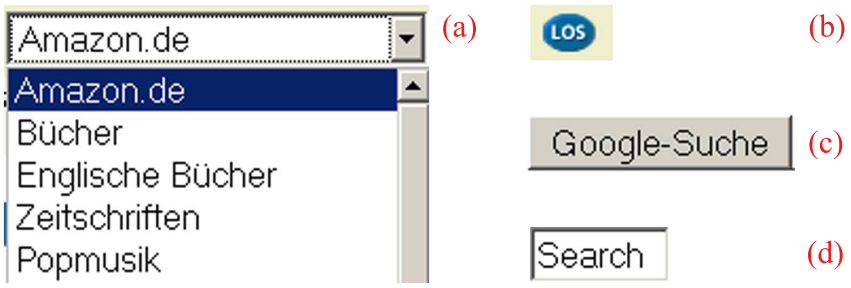

Abbildung 4.5: Auswahl an Controls

die sich später als ungültig erweisen und erneut eingegeben werden müssen. Das erzeugt unweigerlich Unzufriedenheit mit der Handhabung. Durch entsprechendes Design des Eingabefeldes und des Validierungsmechanismus, kann dies jedoch vermieden werden. Cooper unterscheidet 2 Arten von Validierung [CR03]: aktive Validierung erfolgt während der Eingabe und lässt nur bestimmte Zeichen z.B. Zahlen zu. Passive Validierung wartet bis die Eingabe beendet wurde und prüft dann im Hintergrund die Gültigkeit. Cooper empfiehlt die Verwendung der passiven Variante mit einer Wartezeit von 400 Millisekunden, in denen keine Useraktivität feststellbar ist. Beide Validierungsmethoden müssen ein ansprechendes visuelles Feedback liefern, aufgrund dessen die User die Gültigkeit oder Ungültigkeit der Daten erkennen können. Beispielsweise kann das Eingabefeld farblich umrandet werden, um den Zustand - ungeprüft, gültig oder ungültig - zu signalisieren. Die erwartete Eingabe kann seitens der Applikation mit Clue Boxes kommuniziert werden. In XHTML ist dies durch Verwendung eines Attributes für alle Elemente möglich.

Display Controls werden in XHTML Seiten hauptsächlich dazu verwendet, um Teile der Benutzeroberfläche abzugrenzen, ein- oder auszublenden. Dazu eignen sich Iframe Tags sowie Div Tags und Span Tags. Die Manipulationsmöglichkeiten dieser Elemente - verschieben, einklappen, ausklappen - sollen durch ihre visuelle Repräsentation intuitiv erfassbar sein. Ausgelöste Änderungen an der Darstellung der Benutzeroberfläche sollen, seitens der User, nachvollziehbar und reversierbar sein. Innerhalb von ausblendbaren Bereichen sollen sich keine wichtigen interaktiven Komponenten der Applikation befinden, die die User übersehen könnten. Scrollbars sind ein Mechanismus, der seitens des Browsers zur Verfügung gestellt wird, und auf den die angezeigten Webseite wenig Einfluss hat. Über JavaScript Funktionen können jedoch wichtige Bereiche einer Webseite mitscrollen und permanent sichtbar sein.

5 Implementierung des Interaktionsmodells

Das in Kapitel 4 entwickelte Interaktionsmodell soll mit dem aktuellen Stand der Technik im Rahmen einer Webapplikation umgesetzt werden. Der Entwicklungsprozess ist das Thema dieses Kapitels - von der Modellierung bis zur fertigen Webapplikation.

5.1 Modellierung

Der Aufgabenbereich der zu implementierenden Webapplikation findet sich im Projektmanagement. Speziell im universitären Bereich ist eine genaue Zeiterfassung der Arbeitsstunden - bei Lehrveranstaltungen mit praktischem Charakter - seitens der Studenten unabdingbar: die LehrveranstaltungsleiterInnen benötigen eine möglichst genaue Aufstellung der Arbeitsleistung für die Beurteilung der Studierenden und Evaluation der Lehrveranstaltung. Üblicherweise wird eine derartige Aufwandserfassung durch das individuelle Ausfüllen von digitalen Dokumenten - so genannten Stundenlisten - realisiert. Dieses System unterliegt jedoch gewissen Einschränkungen: die digitalen Stundenlisten liegen entweder auf dem privaten PC oder im Speicherbereich der Studierenden auf einem der Universitätsserver. Wird die Arbeit an einem anderen Ort durchgeführt (z.B. eine Besprechung in einem Kaffeehaus), kann der Aufwand nicht sofort eingetragen werden. Dies führt in der Regel zu vernachlässigten Stundenlisten, die bei Abschluss der Lehrveranstaltung nachträglich und damit ungenau aktualisiert werden. Die Informationen, die hierbei verloren gehen, fallen sowohl zu Lasten der Studierenden als auch der LehrveranstaltungsleiterInnen: Erstere haben häufig einen erhöhten Arbeitsaufwand, der jedoch in den Stundenlisten nicht aufscheint, während Zweitere die Angemessenheit der Inhalte der Lehrveranstaltung an den geplanten Arbeitsumfang nicht evaluieren können.

Die skizzierte Situation soll durch die Verwendung der Webapplikation „Zeitspur“ [1] verbessert werden. Die User, also die Studierenden, können von jedem PC mit Internet-Zugang ihren Arbeitsaufwand festhalten. Dabei werden die angefallenen Arbeitszeiten nicht nachträglich eingetragen, sondern gleich während der Arbeit mitgeschnitten. Die Anwendung der Zeitspur kann in metaphorischer Anlehnung an einen Kassettenrecorder erklärt werden: die User wählen eine Aufgabe eines Projektes aus und legen diese in die Zeitspur ein. Über die Controls der Applikation kann die Aufnahme der Arbeitszeit für die eingelegte Aufgabe gestartet, pausiert und gestoppt werden. Nach Beenden der Aufnahme kann die eingelegte Aufgabe außerdem ausgeworfen werden. Ein Auswerfen oder Stoppen der Aufnahme führt immer zu einem Speichervorgang der mitgeschnittenen Arbeitszeit in Minuten, wobei eine Arbeitszeit von geringer als einer Minute verworfen wird. Das Pausieren der Aufnahme kann eingesetzt werden, wenn zwischenzeitlich etwas anderes erledigt werden muss, z.B. die Beantwortung eines Telefonanrufes. Das Aufnehmen von Arbeitsaufwand ist in Abbildung 5.1 als Anwendungsfalldiagramm sowie in Abbildung 5.2 als Zustandsdiagramm illustriert. In Abbildung 5.2 inkludiert das Erreichen des Endzustandes ein Speichern der Arbeitszeit, falls diese größer als eine Minute ist.

Für das Anlegen und Modifizieren von Projekten und Aufgaben gibt es eine spezielle Usergruppe namens Admin. Diese können Projekte anlegen, bearbeiten und löschen. Zu jedem Projekt gibt es eine Liste mit Aufgaben, die sie wiederum anlegen, bearbeiten und löschen können. Die aufgenommenen Arbeitsstunden zu jeder Aufgabe sind ebenfalls in einer Listenansicht zugänglich. Die Anwendungsfälle für die Usergruppe Admin sind in Abbildung 5.3 als Anwendungsfalldiagramm illustriert. Abbildung 5.4 gibt einen Überblick über alle beschriebenen Anwendungsfälle und zeigt die Authentifizierung durch Login / Logout.

Anwendungsfall- und Zustandsdiagramme sind Methoden der Modellierung von Desktopapplikationen. Wie in [KPRR03] erläutert wird, ist dies jedoch häufig unzureichend für die speziellen Charakteristika von Webapplikationen. Daher werden auf der Basis von klassischen Beschreibungssprachen wie UML [Gro07] und ER [Che76] neue angepasste Modellierungsmethoden, wie WebML [CFB+02] und UWE [Koc06] entwickelt. Der Autor dieser Arbeit hat sich bei der weiteren Modellierung der Webapplikation für UWE (UML-based Web Engineering) entschieden, da mit ArgoUWE [KKMZ03] ein frei verfügbares Werkzeug für die Erstellung von Diagrammen existiert.

[1] Die Idee und die Marke „Zeitspur“ stammen von Mag. Horst Tellioglu

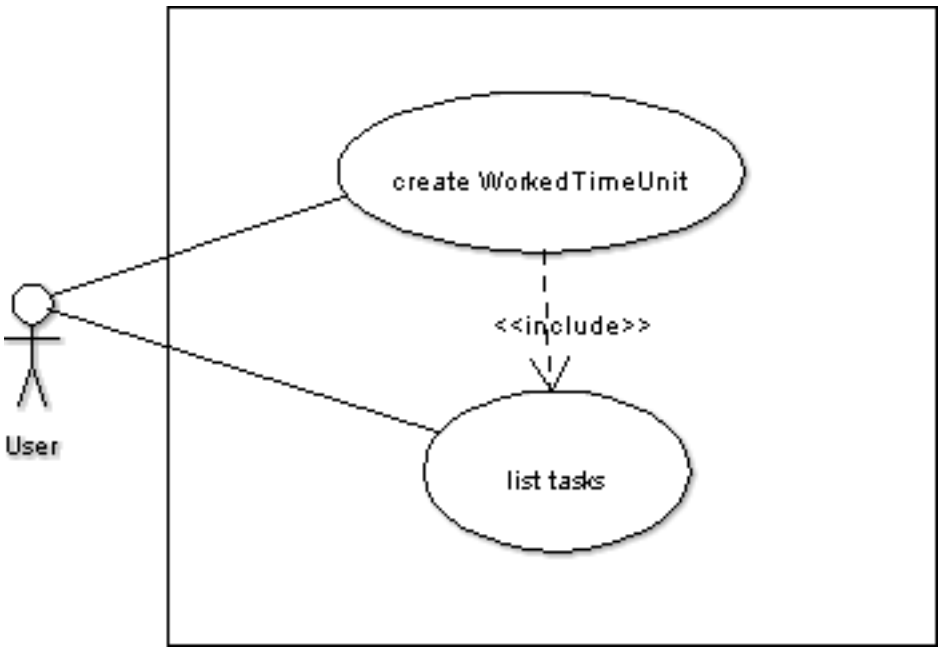

Abbildung 5.1: Anwendungsfalldiagramm für das Aufnehmen von Arbeitsaufwand

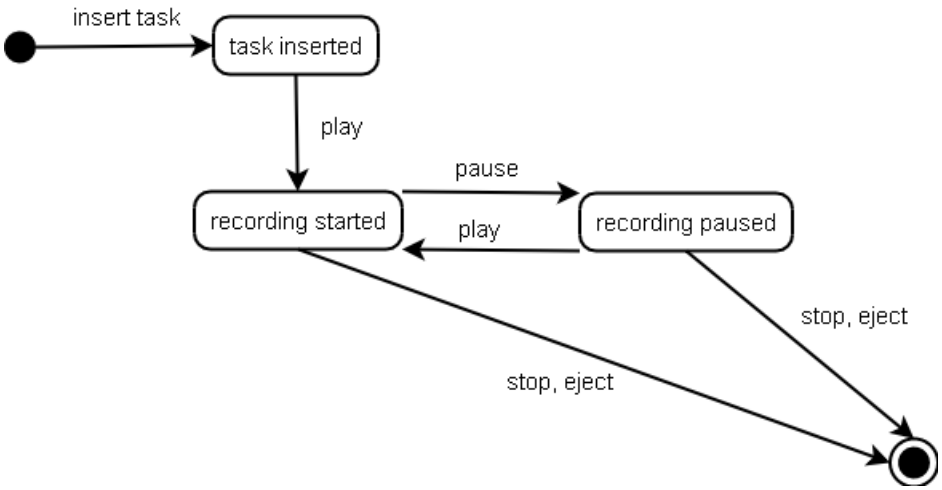

Abbildung 5.2: Zustandsdiagramm für das Aufnehmen von Arbeitsaufwand

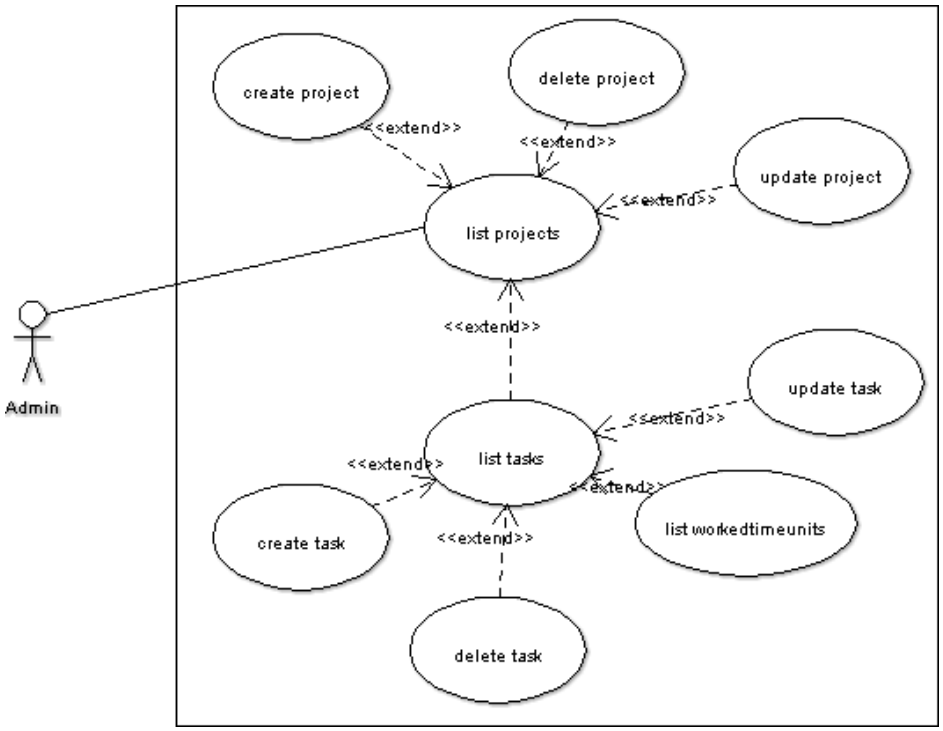

Abbildung 5.3: Anwendungsfalldiagramm für die Usergruppe Admin

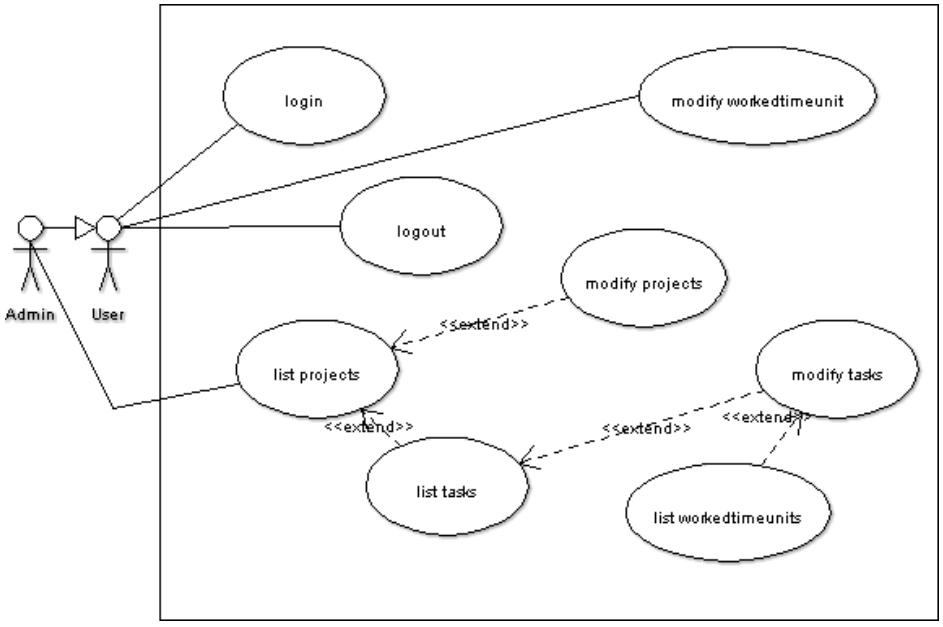

Abbildung 5.4: Übersicht aller Anwendungsfälle

Basierend auf der objektorientierten Modellierungsmethode UML zeigt Abbildung 5.5 ein vereinfachtes Klassendiagramm der Zeitspur. Es gibt Projekte (Project), denen mehrere Aufgaben (Task) zugeordnet sein können. Zu jeder Aufgabe gibt es eine Anzahl von aufgenommenen Arbeitszeiten (WorkedTimeUnit). Die Arbeitszeiten sind den Usern über ihr eindeutiges Login aus dem User Objekt zugeordnet. In Desktopapplikationen wird die Navigation aus den UML Anwendungsfalldiagrammen ersichtlich. Die Nicht-Linarität von Hypertext, wie in Webapplikationen verwendet, erfordert jedoch zusätzliche Konzeptionsmethoden, damit die User nicht kognitiv überlastet werden. UWE berücksichtigt dies mit dem Hypertext-Strukturmodell, welches die Hypertext Verknüpfungen (Links) in Webapplikationen visualisiert. Dafür werden bestehende Objekte aus dem Klassendiagramm durch Menüs, Indexe und Queries verbunden:

- *Menüs* agieren, wie in Kapitel 4.3 beschrieben, als permanente Steuerung durch die Applikation - beispielsweise der Menüpunkt „Verwaltung“ in der Zeitspur.
- *Indexe* sind Auflistungen von Objekten, z.B. eine Liste aller Projekte.
- *Queries* stehen für die Auflistung einer speziellen Teilmenge von Objekten, z.B. aller Arbeitszeiten eines Users für eine Aufgabe.

Für die Webapplikation Zeitspur sind Hypertext-Sichten für die 2 Usergruppen User und Admin notwendig. Abbildung 5.6 zeigt das Hypertext-Strukturmodell aus der Sicht der Mitglieder der Usergruppe Admin. Diese können im Gegensatz zu der Usergruppe User auf alle Bereiche der Applikation zugreifen. Das eingeschränkte Modell für die Usergruppe User wird hier nicht präsentiert.

5.2 Gestaltung des User Interface

Wie in Kapitel 5.1 kurz erläutert ist die Gestaltung des User Interface der Zeitspur von der Metapher eines Kassettenrecorders inspiriert. Das Arbeiten mit Metaphern im Designprozess wird in [Wag05] näher beschrieben. Ein Kassettenrecorder ist ein Gerät, in den Kassetten eingelegt, abgespielt und ausgeworfen werden können. In dem Design wird dieser Abspielvorgang von Musik mit dem Abspielen von Zeit für eine bestimmte Arbeitsaufgabe verknüpft: in der Zeitspur wird die Arbeitszeit von Aufgaben abgespielt und gespeichert. Das Verwenden der Aufnahme Symbolik eines Kassettenrecorders wird absichtlich vermieden, obwohl es ähnlich verwendet werden könnte: Zeit für eine Ar-

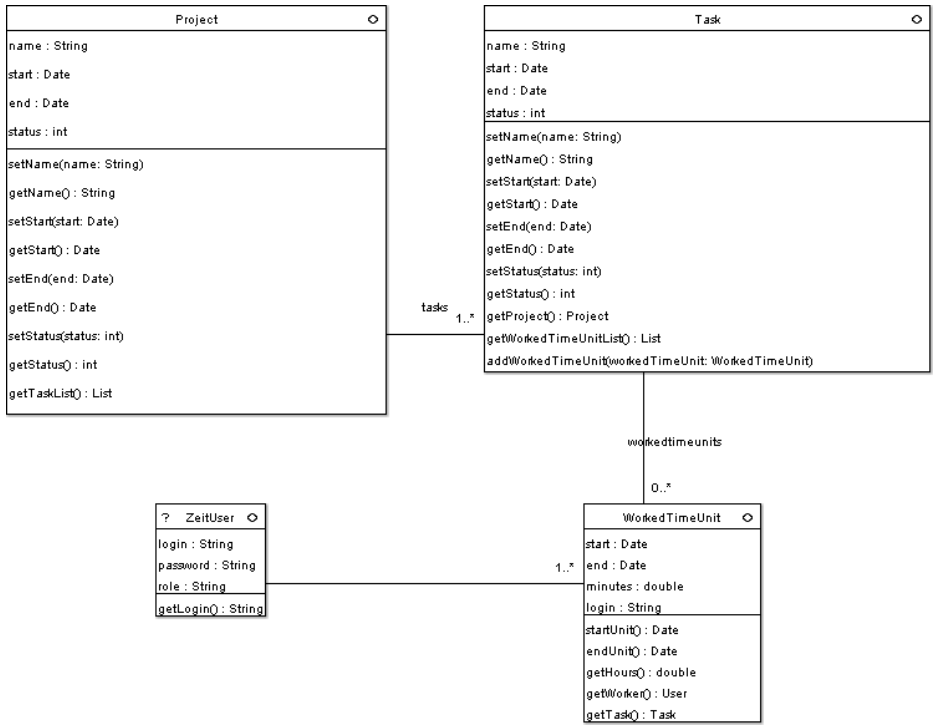

Abbildung 5.5: UML Klassendiagramm der Zeitspur

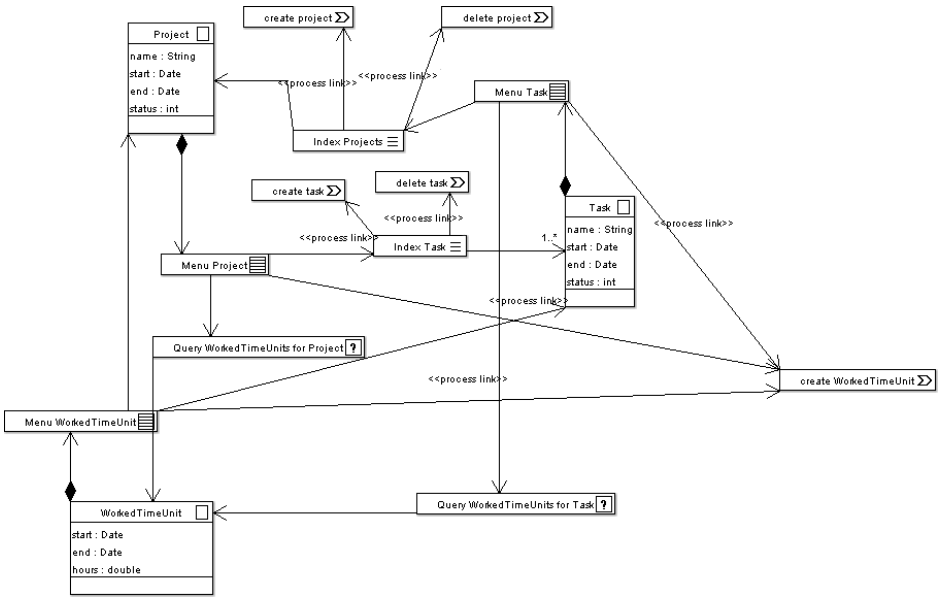

Abbildung 5.6: UWE Hypertext-Strukturmodell für die Usergruppe Admin

beitsaufgabe wird erfasst, wenn der Aufnahme-Button angeklickt wird. Es erscheint dem Autor jedoch unpassend, da mit diesem Vorgang auch das Löschen bzw. Überspielen von Kassetten assoziert wird. Außerdem ist das Drücken der Aufnahmetaste eines Kassettenrecorders mit dem Aufleuchten von Rot in der Anzeige verbunden, was im Kontext einer Applikation jedoch eher einen Ruhe- oder Ausgangszustand anzeigt. Die Zustände Stop, Pause und Ausgeworfen werden daher mit Rot, eine laufende Aufnahme mit Grün visualisiert.

Basierend auf dieser Metapher benötigt das Visual Interface folgende Elemente:

- Arbeitsaufgaben, die wie Kassetten in die Zeitspur eingelegt werden können.
- Ein Ladefach, in das die Arbeitsaufgaben eingelegt werden.
- Eine Zeitanzeige für Stunden, Minuten und Sekunden, in der die abgespielte Zeit visualisiert wird.
- Knöpfe für das Abspielen, Pausieren, Stoppen und Auswerfen von Arbeitsaufgaben.
- Eine farbliche Visualisierung das Abspielvorganges.

Das Einlegen von Arbeitsaufgaben in die Zeitspur impliziert schon hier die Verwendung von Direct Manipulation, wie in dem entwickelten Interaktionsmodell beschrieben. Jede Aufgabe ist also als eigenständiges grafisches Element repräsentiert (Abb. 5.7a). Das Ladefach gleicht in seiner Form und visuellen Repräsentierung den Kassetten (Abb. 5.7b) und enthält zusätzlich einen kurzen textbasierten Hilfehinweis zur Handhabung, sofern nichts eingelegt ist. Die Zeitanzeige sowie die Visualisierung des Abspielvorganges sind animierte Objekte: Ersteres enthält eine fortlaufende Zeit ab dem Betätigen des Abspiel Knopfes (Abb. 5.7c). Zweiteres zeigt den laufenden Abspielvorgang mit einem grünen Rechteck, welches sich von links nach rechts bewegt (Abb. 5.7d), und den gestoppten Abspielvorgang durch ein rotes Rechteck. Die Knöpfe für das Starten, Pausieren, Stoppen und Auswerfen der Arbeitsaufgabe sind mit den von Kassettenrecordern bekannten Symbolen versehen (Abb. 5.7e). Eine Picklist ermöglicht die Anzeige von Arbeitsaufgaben durch Auswahl des übergeordneten Projektes (Abb. 5.7f).

Zusätzlich zu diesen Elementen benötigt das User Interface, gemäß dem Interaktionsmodell, auch eine Navigation sowie eine Anzeigefläche für textbasierte Information. Die Navigation (Abb. 5.7h) besteht aus textbasierten Hyperlinks, welche sich durch die im Web üblichen Unterstriche sowie ihre Zeichengröße von anderen Textstellen abheben.

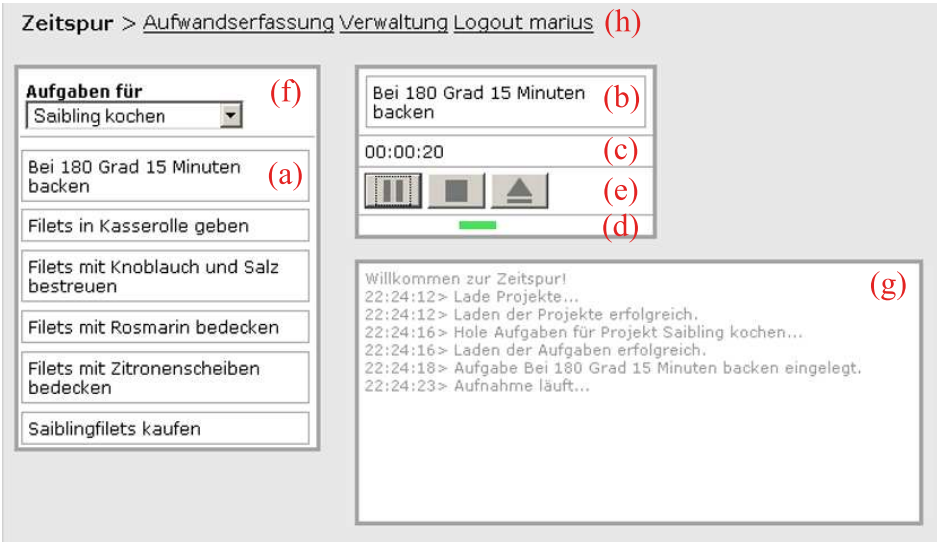

Abbildung 5.7: User Interface der Zeitspur

Die Inhalte in der Anzeigefläche für die textbasierte Kommunikation (Abb. 5.7g) sind in der Schritfgröße etwas kleiner und in der Schriftfarbe schwächer kontrastiert. Dadurch kann der Wahrnehmungsapparat der User die textbasierte Information besser ausblenden, um sich auf die wichtigeren Elemente zu konzentrieren. Für detaillierte Information kann die Aufmerksamkeit auf den Text fokussiert werden, um den Ablauf der Interaktion anhand der Zeitstempel nachzuvollziehen.

5.3 Softwarearchitektur

Die Architektur von Webapplikationen ist nach [KPRR03] auf mindestens 2 Schichten verteilt: Server und Client. In der Praxis bedeutet dies, dass ein Großteil der Applikationslogik auf dem Server angesiedelt ist, der Anfragen von den Clients (bzw. einem Browser auf einem Client Rechner) entgegennimmt, verarbeitet und die Antwort in Form von XHTML Seiten zurücksendet. Dieser Kommunikationsvorgang erfolgt üblicherweise synchron: Der Client auf die Antwort des Servers bevor die User die Applikation erneut bedienen können. Das in Kapitel 5.2 vorgestellte Visual Interface ist mit Methoden der Implementierung von Webapplikationen, die nur auf dieser synchronen Architektur aufbauen, nicht realisierbar. Die Möglichkeit der Verwendung von Interaktionstechniken wie Direct Manipulation wurde allerdings in den letzten 2 Jahren, durch Verwendung von AJAX (Asynchronous JavaScript and XML) ermöglicht. Dabei beschreibt AJAX

weniger eine spezielle Technologie, sondern viel mehr die koordinierte Verwendung von schon vorhandenen Web Technologien [Gar05]:

- Standardisierte Darstellung im Browser mittels XHTML [W3C02] und CSS [W3C06]
- Kommunikation der Daten mit XML [W3C04], XSLT [W3C99] oder JSON [Cro06]
- Asynchrone Kommunikation von Client und Server mittels XMLHttpRequest [McL05]
- Implementieren von Interaktionstechniken im Browser mit JavaScript 1.5 (gleichwertig zu ECMA Script Spezifikation 262 Version 3) [Ass99] und DOM [Con05]

Traditionelle Webapplikationen ohne AJAX sind in ihrer Interaktivität durch die oben skizzierte synchrone Kommunikation zwischen Client und Server eingeschränkt: jede Manipulation der User an der Applikation muss explizit an den Server gesendet werden, damit dieser sie verarbeiten und entsprechend beantworten kann. Dies geschieht immer seitenweise, i.e. müssen die User auf das erneute Laden der kompletten Webseite warten, um das Feedback der Applikation zu erhalten. Der Browser auf der Clientseite agiert also größtenteils passiv und rendert ausschließlich das User Interface. Einerseits verhindert dies den wünschenswerten, flüssigen Dialog zwischen User und Applikation, andererseits können Interaktionstechniken nur seitenweise eingesetzt werden. Da die komplette Applikationslogik, die das Verhalten definiert, auf dem Server liegt, kann das Visual Interface nur nach einem Kommunikationsvorgang mit demselben aktualisiert werden (um den Usern das Verhalten der Webapplikation mitzuteilen). Diese Kommunikation basierend auf der Webseite als kleinsten Einheit macht Interaktionstechniken wie Direct Manipulation unmöglich.

AJAX hingegen überträgt einen Teil der Applikationslogik auf den Client und verbindet diesen über einen asynchronen Kommunikationskanal mit dem Server (Abb. 5.8). Dadurch ensteht in der Benutzeroberfläche auf der Clientseite die Möglichkeit zu granularen Interaktionsformen, die nicht an das Absenden und erneute Laden einer Webseite gebunden sind. Eingaben der User, die die Applikationslogik auf dem Client nicht verarbeiten kann, werden asynchron an den Server kommuniziert. Aufgrund der Asynchronität der Kommunikation erlaubt der Client auch weiterhin Usereingaben, ohne den Interaktionsprozess zu unterbrechen. Sobald die Antwort vom Server eintrifft, führt der Client die erhaltenen Anweisungen aus, beispielsweise wird das Visual Interface aktualisiert.

Mit JavaScript und dem Document Object Model (DOM) können außerdem Elemente des Visual Interface, also die Elemente der XHTML Seite, mit Interaktion versehen wer-

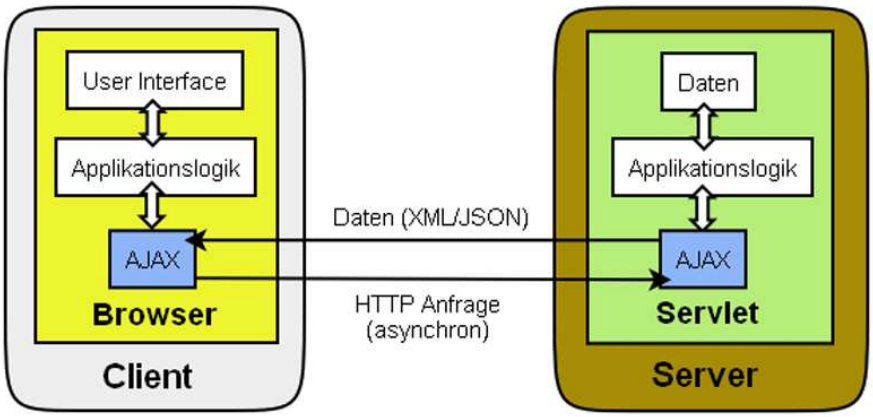

Abbildung 5.8: AJAX Kommunikation zwischen Client und Server

den. Damit wird z.B. der Einsatz von Direct Manipulation möglich. In der Vergangenheit ist dies häufig an der nicht standard konformen und unzureichenden Implementierung dieser beiden Technologien in den Browsern gescheitert: die clientseitige Applikationslogik, welche die Interaktion ausdrückt, musste für jeden Browser, in dem sie funktionieren sollte, angepasst und getestet werden.

Die Architektur der zu entwickelnden Applikation wird mit Hilfe von so genannten Frameworks realisiert. In [KPRR03, S.82] wird der Begriff Framework wie folgt definiert:

> „Ein Softwaresystem, das die Softwarearchitektur und teilweise Implementierungen für eine Familie von Anwendungen bereitstellt. Durch Spezialisierung kann ein Framework in eine konkrete Anwendung überführt werden."

Unterstützt von Frameworks können also immer wiederkehrende Problemstellungen in der Softwareentwicklung, durch die Verwendung von bereits konzipierten und implementierten Programmcode, effizient und ressourcenschonend gelöst werden. Wie in Kapitel 2.1 erläutert, ist die *Verlässlichkeit und Verfügbarkeit von Funktionalität und Daten* der Applikation, eine der Anforderungen für eine an Usability orientierte Entwicklung. Daher wird die Implementierung der Applikationslogik mit Frameworks realisiert, die hinreichend erprobte Patterns zur Verfügung stellen. Die verwendeten Frameworks sollen folgende Anforderungen erfüllen:

1. Abstraktion der Datenbasis mittels Object Relational Management (ORM). Die Datensätze aus der relationalen Datenbank sollen automatisiert als Objekte zur

Verfügung stehen.

2. Transaktionsbasierter Umgang mit Datensätzen. Unvollständige Datensätze also solche, deren Transaktion in der Applikation nicht abgeschlossen wurde, sollen nicht in die Datenbasis abgebildet werden.
3. Template basierte Darstellung des User Interface. Verwenden von XHTML Vorlagen, in die mittels Programmcode Daten eingebunden werden.
4. Authentifizierung von Clients mit Username und Password. Die User der Webapplikation müssen bekannt und authentifiziert sein, um die Zeitspur zu verwenden.
5. Verbinden der Client-Server Applikationslogik über asynchrone Kommunikation. Clients sollen Zugriff auf ausgewählten Programmcode der serverseitigen Applikationslogik haben.
6. Browserübergreifende JavaScript Funktionen für die Manipulation des Visual Interface. Clientseitiger Programmcode, der die Interaktion zwischen Applikation und User steuert, soll auf mehreren gängigen Browsern funktionieren.

Die Auswahl der Frameworks wird auf solche eingeschränkt, die in der Programmiersprache Java verfasst sind: im Gegensatz zu diversen Skriptsprachen, wie PHP oder Ruby, besitzt Java wesentlich bessere Spracheneigenschaften, wie z.B. Typen- und Datensicherheit oder Polymorphismus [Pun06]. Mit dem Apache Tomcat Servlet Container [Fou07] existiert außerdem ein frei verwendbarer Webserver, über den Java in Webapplikationen verwendet werden kann. Von den Java basierten Frameworks werden weiters nur jene gewählt, die aufgrund ihrer Lizenzierung (LGPL, MIT, GPL) für kommerzielle und nicht-kommerzielle Projekte frei verfügbar sind.

Für die Abstraktion der Datenbasis wird das Framework Hibernate [JBo07] gewählt, da es sowohl ORM als auch transaktionsbasiertes Datenmanagement anbietet. Das Spring Framework [Int07] verwaltet die Hibernate Konfiguration, bietet einen Authentifizierungsmechanismus an [Lim07] und erlaubt das Einbinden von Programmcode in XHTML Vorlagen. In Spring wird die für die Zeitspur notwendige serverseitige Applikationslogik in Form von Java Objekten integriert. Hibernate und Spring werden seit Jahren entwickelt und bieten die klassische synchrone Client-Server Architektur für Webapplikationen. Die Unterstützung für AJAX erfordert zusätzlich den Einsatz von jüngeren Projekten.

Bei der Auswahl der AJAX Frameworks werden weitere Kriterien in Erwägung gezogen: da die Geschwindigkeit, mit der die Entwicklung in diesem neuen Bereich der Web-Technologie vorangeht, sehr hoch ist (wie in der Informatik üblich), kommen nur solche Projekte in Frage, die eine angemessene Dokumentation und Weiterentwicklungspotential besitzen. Damit soll versichert werden, dass ein gewähltes Framework in naher Zukunft noch immer betreut und verbessert wird. Hinsichtlich dieser Forderung bietet sich das DWR Framework [Get07] für die asynchrone Verbindung zwischen Client und Server an. Es wird seit 2004 entwickelt und hat einen dokumentierten Projektplan für die Zukunft. Außerdem stellt die Homepage des Projektes ausreichend Dokumentationsmaterial (Bedienungsanleitung, API, Demos, Tutorials, Mailingliste) inklusive einer Integrationsanleitung in das Spring Framework zur Verfügung.

Über DWR können Client und Server zwar asynchron kommunizieren, es bietet jedoch keine Funktionen für die browserübergreifende Manipulation des Visual Interface. In diesem Bereich findet sich das Scriptaculous Framework [Fuc07], das seit 2005 entwickelt und ebenfalls regelmäßig aktualisiert wird. Die Dokumentation auf der Homepage (Bedienungsanleitung, API, Demos, Wiki) erlaubt eine effiziente Verwendung in eigenen Projekten. Es unterstützt sowohl die Verwendung von Direct Manipulation, als auch das abstrakte Manipulieren von DOM Objekten über JavaScript.

In Abbildung 5.9 sind die beschriebenen Frameworks, ihre Komponenten und ihr Zusammenspiel als Softwarearchitekturdiagramm (Architecture Analysis and Design Language [FGH06]) illustriert: die Datenbasis wird innerhalb der Datenbankimplementierung MySQL abgebildet. Hibernate übernimmt die Abbildung der relationalen Daten (aus der Datenbank) auf die objektorientierten Java Entitäten. Die Kommunikation mit der Datenbank wird über Transaktions- und Ressourcenmanagement optimiert. Die derart verfügbar gemachten Objekte werden über Data Access Objects (DAOs) angesprochen. Spring bindet die DAOs als JavaBeans [Mic07a], die über das Bean Management konfiguriert werden, ein. Die Geschäftslogik der Applikation, wie z.B. das Editieren oder Löschen von Datensätzen, wird innerhalb selbst programmierter Business Objects realisiert. Jene Java Objekte, die über Java Servlet Pages [Mic07b] manipuliert werden können, werden als Service Beans freigegeben. Diese Service Interfaces bilden die zentrale Schnittstelle, über welche die Anwendungsmöglichkeiten der Applikation realisiert werden. Mit Java Servlet Pages werden aus Templates XHTML konforme Webseiten generiert, auf denen die User mit der Applikation interagieren. Eine Webseite wird dabei als View abgebildet. Jene Views, auf denen Objekte manipuliert werden, enthalten

Backing Beans - eben die manipulierbaren Objekte. Nach einer Manipulation werden die Daten validiert und entsprechend konvertiert (Validator / Convertor) , damit sie an der Datenbasis wieder in die Datenbank gespeichert werden können.

DWR bildet ausgewählte Java Objekte und Methoden in die Skriptsprache JavaScript ab - es agiert als JavaScript Proxy. Im Browser können derart exportierte Objekte und Methoden asynchron ausgeführt werden. Über Reverse AJAX kann außerdem zur Laufzeit, dynamisch vom Server ausgehend, JavaScript Code auf den Browsern ausgeführt werden. Über Scriptaculous wird das Visual Interface dynamisch und asynchron aktualisiert: wird z.B. das Abspielen einer Aufgabe gestoppt, muss nicht die Seite neu geladen werden, um die Arbeitszeit zu speichern. Dies wird über einen asynchronen Kommunikationskanal zum Server durchgeführt, während der User die Webapplikation flüssig weiterverwendet.

5.4 Applikationslogik

Das Verhalten der Applikation, also die Interaktion mit den Usern, wird von der Applikationslogik bestimmt. Wie in Kapitel 5.3 beschrieben, muss ein Teil dieser Logik auf der Clientseite implementiert werden, um die in dem entwickelten Interaktionsmodell geforderten Techniken umsetzen zu können. DWR ermöglicht die asynchrone Verbindung der Applikationslogik auf Client und Server, durch das Konvertieren und Exportieren von ausgewählten Java Methoden und Objekten. Die Vorgehensweise ist dabei in 2 Schritte gefasst: zuerst werden die gewünschten Exporte in einer XML Datei konfiguriert, danach kann der von DWR generierte JavaScript Code, über einen Link in die XHTML Seiten eingebunden werden.

Die Arbeitsweise von Spring vereinfacht diesen Vorgang noch weiter, indem es die DWR Konfiguration einbindet. Spring baut, wie die meisten Frameworks, auf dem Konzept der Inversion Of Control (IOC), um seine Funktionsweise zu realisieren [Int07]: das Framework stellt einen vordefinierten Kontrollfluss bereit, der an bestimmten Stellen an die implementierten Objekte der EntwicklerInnen abgegeben wird. Nicht das Hauptprogramm bestimmt also die Applikationslogik, sondern die untergeordneten Objekte - daher spricht man von einer Umkehr der Kontrolle. Diese zu verwendenden Objekte und ihr Zusammenspiel werden in einer zentralen XML Datei konfiguriert - in der

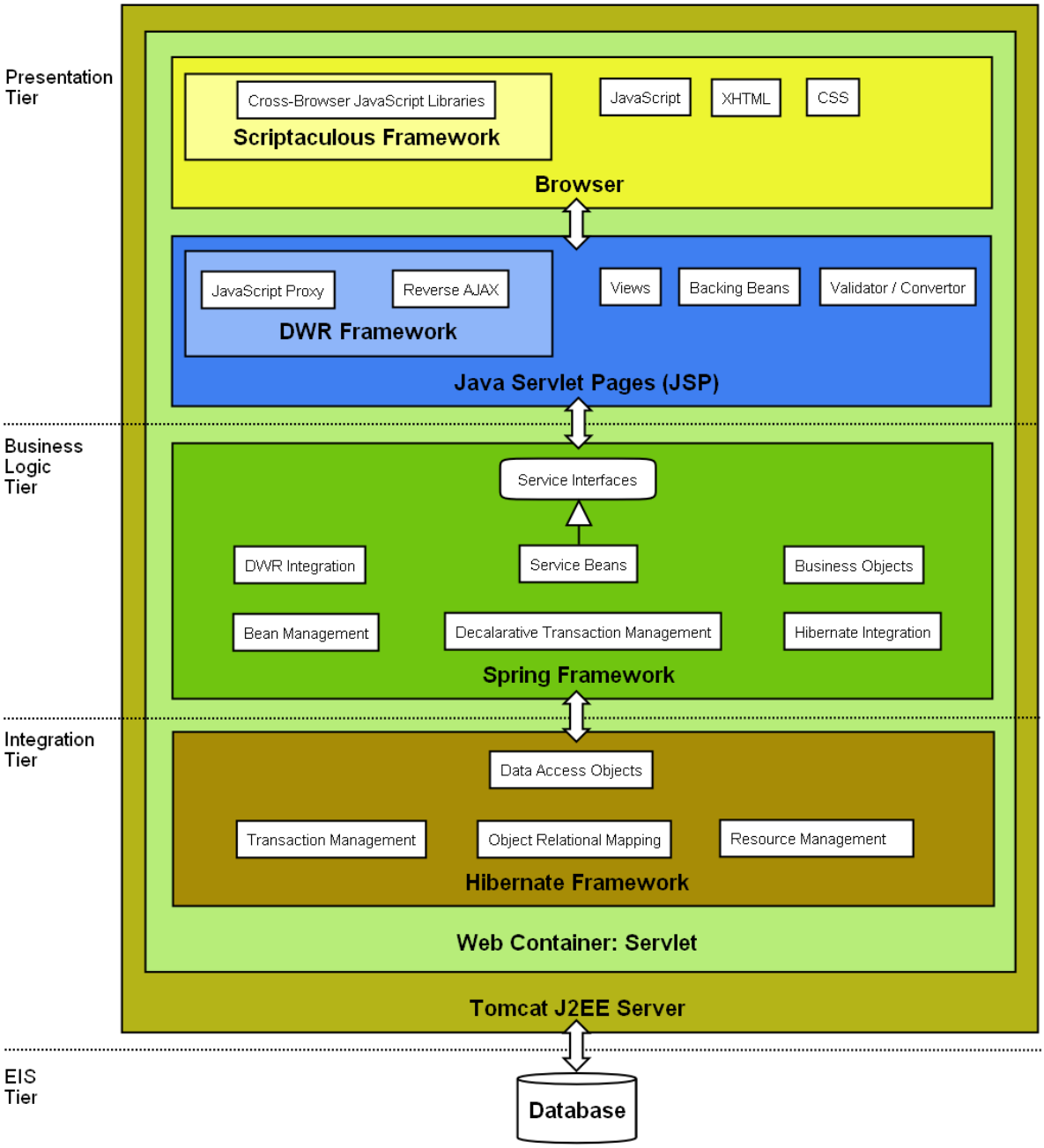

Abbildung 5.9: Softwarearchitektur der Zeitspur

Webapplikation Zeitspur heißt diese beispielsweise „servlet-zeitspur.xml“. In dieser Konfigurationsdatei sind dann auch die DWR Einstellungen integriert:

```
01: <dwr:configuration>
02:    <dwr:convert class="db.Project" type="bean" />
03:    <dwr:convert class="db.Task" type="bean" />
04:    <dwr:convert class="db.WorkedTimeUnit" type="bean" />
05: </dwr:configuration>
06: <bean id="zeitUser" class="db.ZeitUser">
07:    <dwr:remote javascript="zeitUser" />
08: </bean>
```

In Zeilen 1 bis 5 werden Java Objekte angegeben, die als Datentypen in den Client Browsern zur Verfügung stehen sollen. Dies ermöglicht z.B., dass ein clientseitiger Aufruf der Methode `project.getTaskList()` (Abb. 5.5) eine Liste von JavaScript Objekten vom Typ `db.Task` zurückgeben kann. Aufgrund der Verwandtschaft von Java und JavaScript können Java Datentypen wie `Date` oder `int` automatisch konvertiert werden und benötigen keine explizite Zuordnung. Nach dem Erhalt der Objekte kann die Applikationslogik auf dem Client diese normal manipulieren, also z.B. den Wert `Task.name` auslesen. In Zeilen 6 bis 8 werden alle Methoden des Java Objektes vom Typ `db.ZeitUser` in ein JavaScript Objekt mit Namen `zeitUser` exportiert. Nach Einbinden der von DWR generierten JavaScript Codebibliothek, kann z.B. clientseitig die Funktion `zeitUser.getLogin()` (Abb. 5.5) aufgerufen werden, um den Login String der aktuell authentifizierten Sitzung zu erhalten.

Neben der asynchronen Anbindung an den Server muss das Visual Interface auf dem Client die konzipierten Interaktionstechniken umsetzen können. Im clientseitigen Browser wird das User Interface einer Webapplikation in Form von XHTML Seiten angezeigt: diese enthalten eine von der Spezifikation definierte Menge an grafischen Elementen (z.B. Bilder, Text etc.). Das Document Object Model beschreibt wie diese grafischen Elemente von den Browsern als programmierbare Objekte interpretiert werden sollen. Die Anzeige einer XHTML Seite läuft dann folgendermaßen ab: der Browser des Clients empfängt eine Seite mit XHTML Code, parst diesen und erstellt darauf basierende programmierbare Objekte. Diese werden gerendert und schließlich auf der Anzeige des Client Rechners angezeigt. Mit der standardkonformen Verfügbarkeit dieser Objekte in gängigen Browsern kann clientseitiger JavaScript Code diese DOM Objekte und damit

alle Elemente des User Interface dynamisch manipulieren.

Trotz der offenkundigen Bemühungen der Hersteller, die Vorgaben der Standards des W3C zu erfüllen, gibt es häufig Inkompatibilitäten zwischen unterschiedlichen Browser-Implementierungen: clientseitiger Programmcode, der in einem Browser funktioniert, wird von einem anderen Modell fehlerhaft oder gar nicht verarbeitet. Häufig ist dies auf kleinere Abweichungen zurückzuführen, die vorab erkannt und zur Laufzeit vermieden werden können. Scriptaculous kapselt daher jene Funktionen, die Zugriff auf die DOM Objekte im Browser erlauben, und erlaubt deren Verwendung auf einem erhöhten Abstraktionsniveau. Damit können die EntwicklerInnen weitgehend browserunabhängigen Programmcode entwickeln. Die Scritpaculous Biblioteken werden durch Links im XHTML Code der Webseiten eingebunden.

Da die Interaktion ohne ein erneutes Laden der Webseite auskommen soll, muss die clientseitige Applikationslogik in der Lage sein, Elemente des User Interface dynamisch zu erstellen, zu bearbeiten und zu löschen. Beispielsweise wird in der Zeitspur, nach Selektion eines Projektes, nicht die Seite neu geladen, sondern ein asynchroner Kommunikationskanal zum Server geöffnet, über den die Aufgabenliste angefordert wird. Mit der Anforderung wird gleichzeitig eine Funktion verknüpft, die bei Erhalt der Daten auszuführen ist. Diese Funktion kann die erhaltenen Daten - JavaScript Objekte vom Typ `db.Task` - allerdings nicht sofort im User Interface angezeigen, da es sich nicht um XHTML Elemente handelt. Daher erstellt die Applikationslogik auf dem Client, basierend auf den Daten der Aufgaben, DOM Objekte, die in das User Interface eingefügt werden. Etwaig vorhandene grafische Elemente, die Aufgaben eines anderen Projektes repräsentieren, müssen zuvor entfernt werden.

Das Umsetzen der Interaktionstechnik Direct Manipulation erfolgt durch Verwendung der in Scriptaculous enthaltenen Effekt Klassen. Diese bieten eine solide Grundlage, auf der Elemente des Visual Interface mit Interaktion versehen werden können. Die Umsetzung von Drag&Drop in der Zeitspur arbeitet mit den Aufgaben und dem Ladefach: jene DOM Objekte, welche die Aufgaben repräsentieren, werden der Scriptaculous Klasse `Draggable` übergeben. Dadurch können diese mit der Maus aufgenommen, bewegt und abgelegt werden. Das DOM Objekt des Ladefachs wird über `Droppables.add()` zur Zielablage der Aufgaben gemacht.

Die restliche Interaktion rund um die Abspielkontrolle der Zeitspur ist mit eigenständi-

gem JavaScript Code implementiert:

- das Messen der Zeit
- die Aktualisierung der textbasierten und grafischen Applikationsinformationen
- das Speichern der Daten
- die Funktionen der Controls

Die Zeitmessung erfolgt durch Verwendung des `Date` Objektes, welches in JavaScript nativ verfügbar ist. Dieses repräsentiert einen Zeitpunkt, durch Messen der seit 01.01.1970 vergangenen Millisekunden. Bei Starten des Abspielvorganges wird der Startzeitpunkt gespeichert. Über die Browserfunktion `window.setTimeout()` wird der Browser angewiesen, sekündlich eine Funktion aufzurufen, welche die seit dem Start vergangenen Sekunden speichert und die fortlaufende Zeitanzeige in der Abspielkontrolle aktualisiert.

Die textbasierten Informationen in dem Bereich unterhalb der Abspielkontrolle werden über die Funktion `info()` erstellt. Diese generiert einen aktuellen Zeitstempel, kombiniert diesen mit dem Text und fügt beides in das XHTML Element ein. Die Informationen aus diesem Bereich sind vor allem deshalb wichtig, weil hier das erfolgreiche oder nicht-erfolgreiche Ausführen von asynchronen Kommunikationsvorgängen, wie z.B. das Speichern von Arbeitsaufwand, angezeigt wird. Grafische Informationen, wie der Wechsel der Visualisierung zwischen Abspielen und Pause, sind durch Austausch der anzuzeigenden Bilder realisiert: dies ist ein XHTML durch ein Ändern der URL des Bild Elementes möglich.

Das Speichern von Daten wird von den mit DWR exportierten Funktionen übernommen. Da diese Funktionen asynchron ablaufen, erwarten sie als letzten Übergabeparameter eine so genannte Callback Funktion: wenn der Kommunikationsvorgang mit dem Server abgeschlossen ist, wird diese Funktion mit den erhaltenen Daten aufgerufen und muss sich um deren Verarbeitung oder eventuell aufgetretene Fehler kümmern. Beispielsweise wird bei der Initialisierung der Zeitspur die Projektliste durch Aufrufen von `projectDAO.getProjectList(renderProjectSelect)` generiert. Die Funktion `renderProjectSelect()` ist also für die finale Verarbeitung der erhaltenen Daten zuständig.

6 Evaluierung der technischen Umsetzung

Die vom Autor vertretene Ansicht, dass durch die Verwendung von aktuellen Webtechnologien die Usability von Webapplikationen gesteigert werden kann, wird in diesem Kapitel auf ihre Richtigkeit geprüft. Das in Kapitel 4 entwickelte Interaktionsmodell enthält Interaktionstechniken, welche nach dem aktuellen Stand der HCI Forschung die Usability von Applikationen steigern. Dies wurde in der entsprechenden Literatur, wie [SP04] und [CR03], bereits ausführlich diskutiert und empirisch begründet. Dabei darf allerdings nicht vergessen werden, dass die vorgestellten Interaktionstechniken für Desktopapplikationen entworfen wurden, die auf gänzlich andere Technologien und Gebrauchsparadigmen zurückgreifen.

Das Web und seine Applikationen haben, wie in Kapitel 1 beschrieben, eine eigenständige Entwicklung parallel zu der von Desktopapplikationen erlebt. Dies beinflusst einerseits den Umgang der User, andererseits die verwendbaren Technologien. Die Erwartungshaltung der User an Webapplikationen unterscheidet sich oft von der an Desktopapplikationen: auf einer Webseite wird z.B. automatisch nach Hyperlinks und Navigationsmöglichkeiten gesucht. Hinzu kommt, dass die Charakteristika der User, wie in Kapitel 2.3 erläutert, aufgrund der globalen und dezentralen Verfügbarkeit stark heterogen sind. Gerade aufgrund dieser extremen Unterschiede innerhalb der Usergruppe muss die Bedienung - die Interaktion zwischen den Usern und der Applikation - so intuitiv und einfach wie möglich sein.

Die Webtechnologien für eine Realisierung von Interaktionstechniken, welche die Usability positiv beeinflussen, schränken die EntwicklerInnen prinzipiell ein: Desktopapplikationen bauen auf dem Betriebssystem auf und erhalten damit fast unbeschränkten Zugriff auf die vorhandenen Ressourcen des Rechners - eventuell sogar so weit, dass dieser durch fehlerhaften Programmcode abstürzt. Zusätzlich besitzen sie keine 2-schichtige

Client-Server Architektur, sondern greifen immer ohne Kommunikationsverzögerung auf die vollständige Applikationslogik zurück. Webapplikationen hingegen werden komplett innerhalb der Grenzen eines Browser-Programmes ausgeführt, welches den Zugriff auf die Systemressourcen einschränkt oder sogar unterbindet.

Dies lässt sich an dem Beispiel der Interaktionstechnik Direct Manipulation veranschaulichen: damit ein grafisches Objekt per Mausklick aufgenommen und bewegt werden kann, muss die Applikationslogik darauf achten, ob ein derart manipulierbares Objekt angeklickt wird. In Webapplikationen funktioniert dies nicht direkt über die Bibliotheken des Betriebsssystems, sondern nur über Verwendung der Funktionen, die der Browser zur Verfügung stellt. Die Verarbeitungskette wird also um eine Ebene erweitert. Verbietet ein Browser diesen Zugriff, ist eine Realisierung von Drag&Drop unmöglich.

Die 2-schichtige Architektur von Webapplikationen erfordert außerdem ein höheres Maß an Planung bei der Entwicklung der Applikationslogik. Funktionen, welche die Interaktion auf der Clientseite implementieren, müssen ausgelagert und mit Kommunikationskanälen zum Server ausgestattet werden. Dadurch sind die EntwicklerInnen auch gezwungen, in unterschiedlichen Umgebungen zu arbeiten: einerseits in jener Programmiersprache, die für die serverseitige Logik verwendet wird, andererseits in der Skriptsprache JavaScript, mit welcher die Funktionen innerhalb der Browser implementiert werden.

Nach Veranschaulichung der Einschränkungen und Erschwernisse bei der Entwicklung von Webapplikationen stellte sich natürlich die Frage nach den Vorteilen: mit der Platform Internet besitzen diese automatisch eine Reichweite, die Desktopapplikationen erst durch Marketing und/oder Vertriebsstrategien aufbauen müssen. Kommerzielle Webapplikationen können einfach und schnell ausprobiert sowie bei Gefallen erworben werden (meistens ebenfalls online). Im Desktopbereich sind die User, wenn dasselbe Szenario betrachtet wird, zuerst gezwungen, eine Testversion zu beschaffen, diese dann zu installieren und eventuell die komplette Version in einem Geschäft zu bestellen.

Frei verfügbare Webapplikationen profitieren gerade von eben dieser Unabhängigkeit an die Installation auf dem Client Rechner: der benötigt nur einen installierten Web Browser, der die Webapplikation lädt und ausführt. Dies entlastet einerseits viele User, die mit der Installation, Wartung und Deinstallation von Software möglichst in Ruhe gelassen werden wollen, andererseits steigert es die Verfügbarkeit enorm. Wenn Studierende z.B.

die Zeitspur verwenden wollen, müssen Sie sich nur registrieren (um die Zugangsdaten zu erhalten) und danach die entsprechende URL ansurfen.

Die eben skizzierte direkte Verfügbarkeit von Webapplikationen liefert nun auch die Erklärung für die harten Einschränkungen seitens der Browser. Abseits von technisch bedingten Ursachen finden sich diese in sicherheitsrelevanten Aspekten. Die User können in ihrem Browser jede beliebige Webseite oder Webapplikation öffnen, auch solche, die schädlichen Programmcode enthalten. Das kann, z.B. dazu missbraucht werden, um persönliche Informationen (Kreditkartennummern, Kennwörter) oder Dokumente, die auf dem Rechner gespeichert sind, zu stehlen. Damit dies nicht passieren kann, muss der Zugriff einer Webapplikation auf den clientseitigen Rechner durch den Browser gefiltert werden.

Im weiteren erfolgt die Evaluierung der technischen Umsetzung auf 2 Ebenen: der Interaktion mit dem User sowie der Arbeit mit der verwendeten Softwarearchitektur. In Ersterer wird die Umsetzung der Interaktionstechniken auf ihren Einklang mit den aufgestellten Richtlinien des Interaktionsmodells geprüft. Zweitere beschäftigt sich mit der Frage, wie gut sich die verwendete Architektur in den Entwicklungsprozess einbinden lässt.

6.1 Interaktionstechniken

Das *Visual Interface*, ist, wie in Kapitel 4.1 definiert, überall dort zu finden, wo die Applikation ihren Zustand über nicht-textbasierte Elemente kommuniziert. In der Zeitspur sind daher die Abspielkontrolle (ein JavaScript Widget), der Bereich mit den visuellen Repräsentationen der Aufgaben sowie das Seitenlayout dem Visual Interface zuzuordnen (Abb. 6.1). Die [SP04, S.501] „layout appropriateness“, also die angemessene Platzierung der Kontrollelemente, geht aus Abbildung 6.2 hervor: jeder rote Kreis markiert eine Station im Zeigeweg mit der Maus, die nötig ist, um das Abspielen von Zeit für eine Aufgabe zu starten. Die Verbindungen zwischen den Kreisen, also der Zeigeweg, den die User zurücklegen müssen, ist minimal. Etwaige weitere Funktionen, die mit der Maus angesteuert werden, wie der Pause oder Stop Butcon, befinden sich in direkter Nähe zur letzten Position des Mauszeigers.

Die Schaltflächen wurden auf das minimal notwendigste reduziert, um eine kognitive

Überlastung der User, infolge zu viel visueller Information, zu vermeiden. Durch Verwendung von unterschiedlichen Arten der Kontrastierung, wird den Usern der Umgang mit den interaktiven Elementen erleichtert. Die beiden primären grafischen Elemente, die Box mit den Aufgaben sowie die Abspielkontrolle, heben sich durch ihre Hintergrundfarbe und den Rahmen deutlich vom Hintergrund der restlichen Webseite ab. Die Verwendung der Butcons - durch Anklicken - ist durch den dimensionalen Kontrast, also das räumliche Abheben vom Hintergrund, intuitiv erfassbar. Die gemeinsame äußere Form der Aufgaben und des Ladefachs kennzeichnen diese als zusammengehörig. Die, den beiden Boxen innewohnenden, Elemente sind konsistent linksbündig ausgerichtet und in Gitternetzstrukturen eingebettet. Wichtige Elemente befinden sich weiter oben, wie z.B. die Picklist für die Projekte.

Das Applikationsverhalten wird über die Grafik in der Abspielkontrollbox visualisiert (Vergleiche Abb. 6.1 und Abb. 6.2). Ein durchgängiges rotes Rechteck kennzeichnet, dass der Abspielvorgang gestoppt wurde, während ein sich bewegendes, kleineres, grünes Rechteck das Abspielen symbolisiert. Das Aufnehmen und Ziehen von Aufgaben in das Ladefach ist, wie bei vielen Techniken, die auf Direct Manipulation basieren, idiomatisch. Einmal erklärt, können die User diesen Vorgang leicht im Gedächtnis behalten und wiederverwenden.

Problematisch ist die konsistente und browserübergreifende Darstellung des Visual Interface: 2 unterschiedliche Browser (Mozilla Firefox 2 und Microsoft Internet Explorer 7) zeigen kleiner Unterschiede in der Anzeige der grafischen Elemente. Die Butcons der Abspielkontrolle werden im IE7 ohne, in FF2 mit Abstand zu umliegenden Elementen dargestellt. Außerdem wird die Box mit den textbasierten Informationen nicht an derselben Stelle des User Interface plaziert. Dies verletzt klar die geforderte konsistente Ausrichtung von grafischen Elementen. Ein Design, in dem die Box mit den Textinformationen exakt unterhalb der Abspielkontrolle positioniert ist, ist daher für beide Browser aufgrund unterschiedlicher Implementierungen des Anzeigerendering nicht möglich. Der Autor entscheidet sich für die korrekte Ausrichtung in FF2.

Content Organization, also die Anzeige von textbasierte Information, gibt es in der Zeitspur in allen 3 primären Elemente des User Interface. In der Aufgabenbox ist jede Aufgabe als Rechteck symbolisiert, das eine Kurzbeschreibung enthält. Dies folgt dem Leitsatz von Cooper [CR03, S.242]: „visually show what, textually show which“. In der Abspielkontrolle wird die verstrichene Zeit ebenfalls textbasiert in dem Format

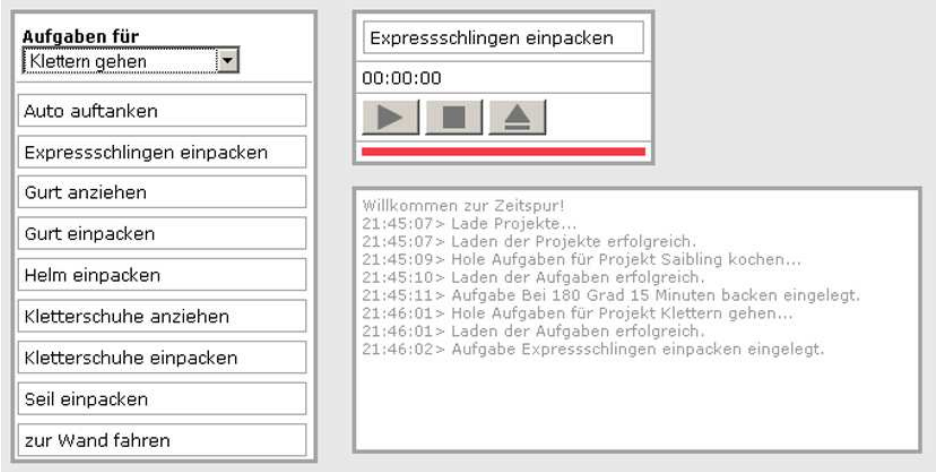

Abbildung 6.1: Visual Interface der Zeitspur

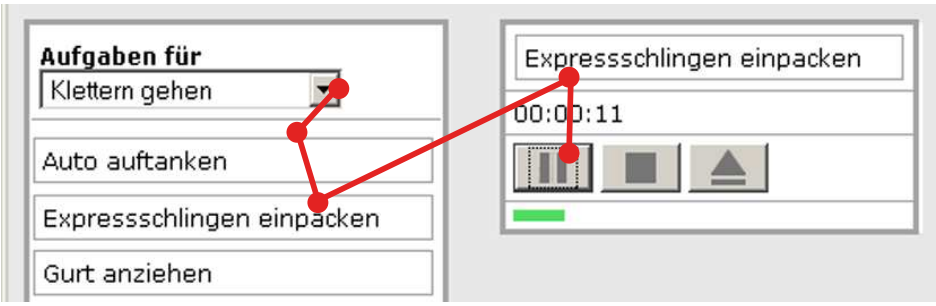

Abbildung 6.2: Layout Appropriateness des Visual Interface

Abbildung 6.3: Content Organization in der Zeitspur

HH:MM:SS angezeigt. Das Verwenden dieses weitverbreiteten Formates, in dem auch einstellige Ziffern mit einer führenden 0 angezeigt werden, erleichtert die Wahrnehmung seitens der User. Für längere Nachrichten wird ein spezieller Bereich des User Interface verwendet (Abb. 6.3).

Dieser Bereich enthält alle Meldungen der Zeitspur seit Start der Webapplikation. Jede Meldung befindet sich in einer Zeile und ist mit einem vorangestellten Zeitstempel versehen. Die verwendete Schriftart (etwas kleiner) sowie die Schriftfarbe (etwas blasser) senken den Kontrast, damit der Wahrnehmungsapparat der User die angezeigten Informationen (bei Desinteresse) leicht ausblenden kann. Eine Ausnahme bilden Fehlermeldungen: diese machen durch Verwendung roter Farbe auf sich aufmerksam. Mit 11 Punkt hat der Text in diesem Bereich die, nach Cooper, minimal zu verwendende Schriftgrösse. Die Informationen sind per Zeitstempel von oben (älter) nach unten (jünger) sortiert, wobei ältere Meldungen angescrollt werden müssen.

Die *Navigation* der Zeitspur ist auf wenige Seiten begrenzt: die Usergruppe User hat Zugriff auf die Aufwandserfassung sowie auf die Logout Seite, während die Usergruppe Admin zusätzlich den Verwaltungsbereich ansteuern darf (Abb. 6.4). Der Link in den Verwaltungsbereich wird also nur für die User der Gruppe Admin eingeblendet. Da die Navigation mit einer Ebene auskommt, besteht kein Bedarf für verschachtelte Menüs oder ähnliches. Die Navigationsleiste wurde als Signpost entworfen, enthält Hyperlinks, die sich durch ihre Darstellung von nicht-interaktiven Textelementen abheben, und ist applikationsweit sichtbar. Sie ist am oberen und nicht am linken Rand des User Interface positioniert, damit der Platz für die Aufgabenliste der Aufwandserfassung frei bleibt. Die

Abbildung 6.4: Navigation der Zeitspur

Navigationselemente innerhalb der Aufwandserfassung, also z.B. die Projekt Picklist, sind simultan bedien- und einstellbar, wie es das Interaktionsmodell für transaktionale Bereiche fordert.

Direct Manipulation wird in der Zeitspur verwendet, um die Aufgaben aus dem Listenbereich in das Ladefach einzulegen. Die User klicken eine Aufgabe an, halten die Maustaste gedrückt, ziehen sie auf das Ladefach und lassen los. Diese Verwendung erfüllt die 3 Charakteristika von Shneiderman, aus Kapitel 4.4:

1. die Aufgaben sind als Rechtecke mit Beschriftung visuell repräsentiert und manipulierbar,
2. die Manipulation erfolgt durch physische Interaktion, in diesem Fall mit der Maus,
3. die Auswirkungen der Interaktion sind sofort sichtbar, z.B. bewegt sich die Aufgabe über die Anzeige und erscheint im Ladefach.

Die visuelle Repräsentierung der Aufgaben wurde sehr simpel gehalten, um die benötigte Anzeigefläche zu minimieren. Damit soll auch bei Projekten mit einer großen Anzahl an Aufgaben versichert werden, dass nicht zu weit gescrollt werden muss, um die Aufgabe auf die Ladefläche zu ziehen. Das Bewegen von Aufgaben über die sichtbare Anzeigefläche (innerhalb des Browsers) hinaus führt zu einem Mitscrollen des Fensters.

Cooper bezeichnet die in der Zeitspur verwendete Art der Direct Manipulation als Drag&Drop [CR03, S.289]. Dabei erfüllt die Implementierung die während der Interaktion durchzuführenden visuellen Änderungen in der Anzeige: mit dem Aufnehmen einer Aufgabe, wird ein transparentes Abbild derselben an den Mauszeiger angeheftet. Dieses bewegt sich mit dem Mauszeiger über das User Interface. Bewegt sich der Mauszeiger mit dem Objekt über den passenden Zielbereich, das Ladefach, erhält dieser eine gelbe Umrandung, um zu signalisieren, dass hier losgelassen werden kann. Nach Loslassen der Aufgabe erscheint diese im Ladefach, um das Resultat des Drag&Drop Vorganges anzuzeigen. Wird eine Aufgabe nicht im Ladefach, sondern an einer anderen Stelle des User Interface abgelegt, zeigt ein Verschwinden des transparenten Abbildes an, dass eine

Abbildung 6.5: Direct Manipulation in der Zeitspur

Interaktion an dieser Stelle nicht möglich ist.

In der Zeitspur werden 2 Arten von *Controls* verwendet: die Butcons in der Abspielkontrolle (Imperative) und die Picklist zur Projektauswahl (Selection). Die Funktionsweise der Butcons, das Anklicken, ist durch ihre grafische Repräsentation - sie haben einen dimensionalen Kontrast zum Hintergrund - intuitiv erfassbar. Welche Funktion sie auslösen, wird mit dem auf ihrer Oberseite platzierten Symbol ausgedrückt. Wie in Kapitel 5.2 erläutert sind die Symbole von Kassettenrecordern und ähnlichen Geräten bekannt.

Falls die User die Bedeutung der Symbole nicht kennen, wird ihre Funktionsweise über andere im User Interface sichtbare Artefakte erschlossen: bei Betätigen des Start Butcons (Abb. 6.6b oben links) wird das Abspielen der Zeit aus dem wandernden grünen Rechteck und der fortlaufenden Zeitanzeige ersichtlich. Ist das Abspielen gestartet, wird das Start Butcon zum Pause Butcon (Abb. 6.6b unten links). Der Pause Butcon visualisiert das Pausieren der Zeit durch Einblenden des roten Rechteckes bei stehenbleibender Zeitanzeige. Der Stop Butcon (Abb. 6.6b oben Mitte) hingegen setzt die Zeitanzeige auf den Ausgangszustand (00:00:00) zurück. Mit dem Auswerfen (Abb. 6.6b oben rechts) Butcon wird deutlich sichtbar die Aufgabe im Ladefach entfernt. Sollten die eben skizzierten Änderungen des User Interface das Applikationsverhalten bei Betätigen der Imperative Controls nicht ausreichend kommunizieren, kann auch die textbasierte Information (in der Box unterhalb der Abspielkontrolle) gelesen werden.

Die Picklist zur Auswahl eines Projektes, dessen Aufgaben angezeigt werden sollen, wird durch Anklicken des seitlich plazierten Butcons geöffnet (Abb. 6.6a). Die Kombination aus dem kurzen Hinweistext darüber sowie im ersten im Anfangszustand immer sichtbaren Eintrag innerhalb der Picklist verdeutlichen die Verwendung.

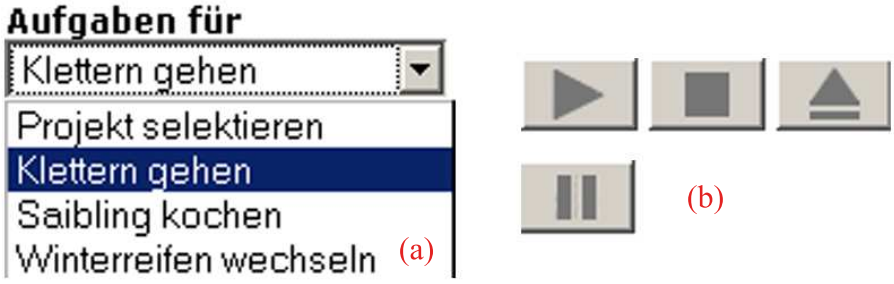

Abbildung 6.6: Controls der Zeitspur

6.2 Entwicklungsprozess

Wie zu Anfang des Kapitels kurz beschrieben ist der Entwicklungsprozess bei Webapplikationen von besonderen Ansprüchen geprägt: Einerseits muss die Modellierung auf die spezielle Eigenschaft der Hypertexttechnologie eingehen, andererseits wird die Applikationslogik getrennt und auf eine 2schichtige Architektur verteilt. Die Verwendung der Beschreibungssprache UML-based Web Engineering (UWE) für die Erstellung des Hypertext-Strukturmodelles hat sich als sinnvoll erwiesen, da die Zusammenhänge zwischen einzelnen Webseiten klar abgebildet werden konnten. Die anderen Vorgänge innerhalb der Zeitspur wurden mit den bekannten Methoden der objektorientierten Entwicklung (Anwendungsfall- und Zustandsdiagrammen) beschrieben.

Die serverseitige Softwarearchitektur stützt sich mit Spring und Hibernate auf reife und bewährte Frameworks für die Entwicklung von Webapplikationen. Durch Verwendung der Programmiersprache Java lassen sich Fehler im Programmcode der Applikationslogik auf ein Minimum reduzieren: Im Gegensatz zu Skriptsprachen wird dieser, bevor er ausführbar ist, vom Compiler auf Fehler geprüft. Das Einbinden der eigenständig realisierten Objekte als Beans wird durch die zentrale Konfiguration von Spring erleichtert. Positiv anzumerken ist auch die semi-automatisierte Einbindung von Hibernate. Unklarheiten und Probleme während der Entwicklung konnten stets mit Hilfe der im Internet verfügbaren Ressourcen (Bedienungsanleitung, API, Forum) erklärt und gelöst werden.

Die Entwicklung auf der Clientseite gestaltet sich auch mit Unterstützung durch DWR und Scriptaculous anspruchsvoller: aktuell gängige Browser zeigen zwar deutliche Fortschritte in der standardgetreuen Umsetzung von Technologien wie JavaScript oder DOM, es kommt jedoch immer wieder zu Inkompatibilitäten, wie z.B. bei der Anzeige des Vi-

sual Interface (siehe Kapitel 6.1). Vom Zugriff auf DOM Objekte ohne die kapselnden Funktionen von Bibliotheken, wie sie Scriptacolous anbietet, wird abgeraten. Versuche während der Entwicklung haben gezeigt, dass dies zu unerwartetem Verhalten seitens der Browser wie z.B. dem Verschwinden der Objekte führen kann.

Das Einbinden der über DWR bereitgestellten asynchronen Kommunikation hat sich als einfach und problemlos erwiesen. Das Framework unterstützt den Entwicklungsprozess hier zusätzlich mit der Verfügbarkeit einer Testumgebung, in der alle exportierten Objekte und Methoden über Aufrufe auf XHTML Seiten ausprobiert werden können. Das Verwenden der von Scriptaculous bereitgestellten Objekte für die Realisierung des Drag&Drop der Aufgaben hat gut funktioniert: die speziellen Anforderungen des Interaktionsmodells konnten durch Einbinden von eigenem Programmcode umgesetzt werden.

Für die Realisierung clientseitiger Applikationslogik wird die Verwendung des Browsers Mozilla Firefox empfohlen, da dieser infolge der integrierten JavaScript Fehlerkonsole und gratis Erweiterungen wie z.B. WebDeveloper den Entwicklungsprozess aktiv unterstützt: fehlerhafter JavaScript Code lässt sich effizient finden und korrigieren.

7 Zusammenfassung & Ausblick

Usability ist in der aktuellen Softwareentwicklung eines der Schlüsselkonzepte für erfolgreiche Applikationen. Das Erforschen und Miteinbeziehen spezifischer Charakteristika der User, wie z.B. kognitiver Fähigkeiten, kultureller Besonderheiten oder persönlicher Eigenschaften in den Entwicklungsprozess resultiert in einer effizienten, effektiven und zufriedenstellenden Handhabung. Aufgrund ihres direkten Kontakts mit den Usern sind Interface und Interaktion zwei Schlüsselkomponenten, deren angemessenes Design die Usability stark (positiv) beeinflusst.

Interaktionsdesign umfasst sowohl die Gestaltung relevanter Teile des User Interface als auch das Konzipieren der Kommunikation zwischen den Usern und der Applikation. Der junge Forschungszweig Human Computer Interaction (HCI) konzentriert sich, seit ungefähr 25 Jahren, auf die Erforschung und Entwicklung von Interaktionsformen, welche die Nutzbarkeit von Computersystemen steigern. Sie brachte Techniken hervor, die in Desktopapplikationen heute allgegenwärtig sind (z.B. Icons, Menüs oder Direct Manipulation) und bei der Entstehung des Internet (um das Jahr 1995) bereits bekannt und verfügbar waren.

Die Usability von Webapplikationen, welche die globale Plattform Internet und ihre Vorteile nutzen wollen, leidet jedoch bisher unter dem harten Konkurrenzkampf zwischen verschiedenen Browserherstellern. Diese vernachlässigen das standardkonforme Implementieren von Webtechnologien und verhindern damit ein Umsetzen von in Desktopapplikationen bereits erfolgreichen Interaktionstechniken. Der Autor untersuchte daher die Fragestellung, ob unter Verwendung aktueller Webtechnologien und Browser-Implementierungen, ein stärker an Usability orientertes Interaktionsdesign in Webapplikationen möglich ist.

Beaudouin-Lafon, Cooper und Shneiderman erklären und beschreiben in ihren Arbeiten das reichhaltige Wissen im Bereich der Human Computer Interaction. Mit dem Begriff

des Interaktionsmodells (nach Beaudouin-Lafon) wird eine theoretische Basis für das Interaktionsdesign in Applikationen herausgearbeitet. Darauf aufbauend werden interaktive Komponenten von Webapplikationen beschrieben und definiert. Das entwickelte Interaktionsmodell kombiniert dann geeignete, bekannte und bewährte Interaktionstechniken für diese Komponenten, um die Usability von Webapplikationen positiv zu beeinflussen.

Basierend auf dem Interaktionsmodell wird die Webapplikation „Zeitspur“ entwickelt, welche die Aufwandserfassung im Anwendungsgebiet Projektmanagement erleichtert. Im Entwicklungsprozess werden Modellierungstechniken verwendet, die ein Abbilden der speziellen Charakteristika von Webapplikationen erlauben. Die Gestaltung des User Interface bedient sich der Inspiration durch Metaphern. Die gefordeten Interaktionstechniken sind nur unter Verwendung aktueller Webtechnologien, wie sie von AJAX (Asynchronous JavaScript and XML) beschrieben wird, umsetzbar. Basierend auf Frameworks wird eine Softwarearchitektur entwickelt, die AJAX unterstützt.

Die Umsetzbarkeit der im Interaktionsmodell geforderten Interaktionstechniken mit aktuellen Webtechnologien wird anhand der Zeitspur evaluiert. Dabei zeigt sich, dass beinahe alle Anforderungen umgesetzt werden konnten: Im geringeren Maße problematisch ist die konsistente und browserübergreifende Positionierung von grafischen Elementen. Das erfolgreiche Umsetzen des Interaktionsmodells bedeutet nach den Erfahrungen, die bei der Verwendung der darin enthaltenen Techniken in Desktopapplikationen gemacht wurden, eine positive Beeinflussung der Usability. Aufgrund der gewählten Softwarearchitektur, die auf einer Kombination von Frameworks basiert, lässt sich das entwickelte Interaktionsmodell zudem effektiv in den Entwicklungsprozess integrieren und wiederverwenden.

Der Autor vertritt die Meinung, dass die Verbreitung von Webapplikationen aufgrund ihrer großen Reichweite und einfachen Verfügbarkeit in Zukunft weiter zunehmen wird. Im Sinne einer userzentrierten Entwicklung ist daher eine Neuorientierung an Interaktionstechniken, welche die Usability steigern, unumgänglich. Derzeit greift man hier auf jene zurück, welche bereits in Desktopapplikationen bekannt und erfolgreich sind. Diese Entwicklung ist naheliegend, nachdem sie dem Web jahrelang vorenthalten wurde. Zeigt sich hier eine gute Umsetzbarkeit und breite Akzeptanz, ebnet das möglicherweise auch den Weg für zukünftige und neuartige Interaktionsformen, die speziell an das Internet angepasst sind.

Problematisch für die DesignerInnen sind hierbei die Einschränkungen seitens der Browser, welche die technischen Möglichkeiten und damit auch das kreative Potential an sich beeinflussen: Ein nach allen Seiten offener Designprozess muss über die Grenzen des Browserfensters hinausgehen. Zeigt sich, dass ohne Rücksicht auf die Browser noch bessere Interaktionsformen geschaffen werden können, muss ein Umdenken hin zu eigenständig internetbasierten Applikationen oder anderen Browser-Implementierungen stattfinden.

Auch die Handhabung von Desktopapplikationen steht aktuell vor einem Umbruch, der mit dem Fortschreiten der technischen Entwicklung immer näher kommt: Konzepte wie Pervasive Computing oder Tangible Interfaces werden langfristig genauso selbstverständlich werden, wie heutzutage die Verwendung der Maus. Dies wird sich zwangsläufig ebenfalls auf die Interaktionsformen der Browser und Webapplikationen auswirken. Es kann also als sicher erachtet werden, dass ein stetiger, kreativer und ingenieurswissenschaftlicher Prozess notwendig ist, um eine an Usability orientierte Interaktion zwischen User und Applikation zu erhalten.

Literaturverzeichnis

[Ass99] European Computer Manufacturers Association. Standard ecma-262, ecmascript language specification [online]. 1999 [cited 20061202]. Available from: <http://www.ecma-international.org/publications/standards/Ecma-262.htm>. Definition of the ECMAScript scripting language.

[BL98] Time Berners-Lee. The world wide web: A very short personal history [online]. 1998 [cited 20061130]. Available from: <http://www.w3.org/People/Berners-Lee/ShortHistory.html>.

[BL00] Michel Beaudouin-Lafon. Instrumental interaction: an interaction model for designing post-wimp user interfaces. In *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 446–453, New York, NY, USA, 2000. ACM Press.

[BL04] Michel Beaudouin-Lafon. Designing interaction, not interfaces. In *AVI '04: Proceedings of the working conference on Advanced visual interfaces*, pages 15–22, New York, NY, USA, 2004. ACM Press.

[BMRS96] Frank Buschmann, Regine Meunier, Hans Rohnert, and Peter Sommerlad. *A System of Patterns. Pattern-Oriented Software Architecture.: 1.* Wiley and Sons, July 1996. Available from: <http://www.amazon.de/System-Patterns-Pattern-Oriented-Software-Architecture/dp/0471958697/ref=pd_bxgy__text_b/303-8002154-4485040>.

[Boe76] B. W. Boehm. Software engineering. In *IEEE Transactions on Computers*, pages 1226–1241, 1976.

[Bor03] John Borland. Browser wars: High price, huge rewards [online]. 2003 [cited 20061202]. Available from: <http://news.zdnet.com/2100-3513_22-996866.html>. Article on zdnet.com explaining the fight between Microsoft and Netscape for browser market shares.

[Car03] John Carroll. *HCI Models, Theories, and Frameworks. Toward a Multidisciplinary Science.* Elsevier Books, May 2003. Available from: <http://www.amazon.de/Models-Theories-Frameworks-Multidisciplinary-Science/dp/1558608087>.

[CFB+02] Stefano Ceri, Piero Fraternali, Aldo Bongio, Marco Brambilla, Sara Comai, and Maristella Matera. *Designing Data-Intensive Web Applications.* Morgan Kaufmann, December 2002. Available from: <http://www.amazon.com/Designing-Data-Intensive-Applications-Kaufmann-Management/dp/1558608435>.

[Che76] Peter Pin-Shan Chen. The entity-relationship model: toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976.

[Con05] World Wide Web Consortium. Document object model (dom) [online]. 2005 [cited 20061202]. Available from: <http://www.w3.org/DOM/>. Definition of the Document Object Model.

[Con06a] Apple Developer Connection. Introduction to apple human interface guidelines [online]. 2006 [cited 20070125]. Available from: <http://developer.apple.com/documentation/UserExperience/Conceptual/OSXHIGuidelines/XHIGIntro/chapter_1_section_1.html>. Apple Human Interface Guidelines.

[Con06b] World Wide Web Consortium. World wide web consortium [online]. 2006 [cited 20061202]. Available from: <http://www.w3.org>. Homepage of the World Wide Web Consortium.

[CR03] Alan Cooper and Robert M. Reimann. *About Face 2.0: The Essentials of Interaction Design.* Wiley, March 2003. Available from: <http://www.amazon.co.uk/exec/obidos/ASIN/0764526413/citeulike-21>.

[Cra02] Chris Crawford. *The Art of Interactive Design: A Euphonious and Illuminating Guide to Building Successful Software.* No Starch Press, December 2002. Available from: <http://www.amazon.com/Art-Interactive-Design-Euphonious-Illuminating/dp/1886411840>.

[Cro06] D. Crockfdord. Javascript object notation (json) memo [online]. 2006 [cited 20070413]. Available from: <http://www.ietf.org/internet-drafts/draft-crockford-jsonorg-json-04.txt>. Memo of JavaScript Object Notation (JSON) Standard.

[Ell05] John Elliot. The ibm 5271 (aka 3270 pc) [online]. 2005 [cited 20061122]. Available from: <http://www.seasip.info/VintagePC/5271.html>. Description and Images of an IBM 5271 computer.

[FGH06] Peter H. Feiler, David P. Gluch, and John J. Hudak. The architecture analysis & design language (aadl): An introduction [online]. 2006 [cited 20070411]. Available from: <http://www.sei.cmu.edu/pub/documents/06.reports/pdf/06tn011.pdf>. Introduction to SAE's AADL Language.

[Fou07] The Apache Software Foundation. Apache tomcat [online]. 2007 [cited 20070416]. Available from: <http://tomcat.apache.org/>. Homepage of the Apache Tomcat Project.

[Fuc07] Thomas Fuchs. script.aculo.us: it's about the user interface, baby! [online]. 2007 [cited 20070411]. Available from: <http://script.aculo.us/>. Homepage of the scriptaculous Framework Project.

[Gar05] Jesse James Garrett. Ajax: A new approach to web applications [online]. 2005 [cited 20070416]. Available from: <http://www.adaptivepath.com/publications/essays/archives/000385.php>. Jesse James Garrett's Article on AJAX.

[Get07] Getahead. Direct web remoting: Dwr is easy ajax for java [online]. 2007 [cited 20070411]. Available from: <http://getahead.org/dwr>. Homepage of the DWR Framework Project.

[GG01] Martin Gaedke and Guntram Gräf. Development and evolution of web-applications using the webcomposition process model. In *Web Engineering, Software Engineering and Web Application Development*, pages 58–76, London, UK, 2001. Springer-Verlag.

[GHJ95] Erich Gamma, Richard Helm, and Ralph E. Johnson. *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison-Wesley Longman, March 1995. Available from: <http://www.amazon.de/Patterns-Elements-Reusable-Object-Oriented-Software/dp/0201633612>.

[Gro07] Object Management Group. Uml resource page [online]. 2007 [cited 20070414]. Available from: <http://www.uml.org/>. Specification of the UML Language.

[HH95] Pierce J. Howard and Jane M. Howard. An introduction to the five-factor

model of personality [online]. 1995 [cited 20061228]. Available from: <http://www.centacs.com/quickstart.htm>. The Big Five Personality Test.

[Hud04] William Hudson. Breadcrumb navigation: there's more to hansel and gretel than meets the eye. *interactions*, 11(5):79–80, 2004.

[IH02] Melody Y. Ivory and Marti A. Hearst. Statistical profiles of highly-rated web sites. In *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 367–374, New York, NY, USA, 2002. ACM Press.

[Int07] Interface21. Spring framework: the leading full-stack java/jee application framework [online]. 2007 [cited 20070411]. Available from: <http://www.springframework.org/>. Homepage of the Spring Framework Project.

[JBo07] JBoss. Hibernate: Relational persistence for java and .net [online]. 2007 [cited 20070411]. Available from: <http://www.hibernate.org/>. Homepage of the Hibernate Framework Project.

[Joh03] Jeff Johnson. *Web Bloopers. 60 Common Web Design Mistakes, and How to Avoid Them.* Morgan Kaufmann Publishers Inc, Mai 2003. Available from: <http://www.amazon.de/Bloopers-Common-Design-Mistakes-Avoid/dp/1558608400/sr=1-3/qid=1163857690/ref=sr_1_3/303-6156075-0689001?ie=UTF8&s=books-intl-de>.

[Kei98] David Keirsey. *Please Understand Me II: Temperament, Character, Intelligence.* Prometheus Nemesis Book Company, May 1998. Available from: <http://www.amazon.com/Please-Understand-Temperament-Character-Intelligence/dp/1885705026>.

[KKMZ03] A. Knapp, N. Koch, F. Moser, and G. Zhang. Argouwe: A case tool for web applications. *First Int. Workshop on Engineering Methods to Support Information Systems Evolution (EMSISE 2003)*, September 2003.

[Koc06] Nora Koch. Transformation techniques in the model-driven development process of uwe. In *ICWE '06: Workshop proceedings of the sixth international conference on Web engineering*, page 3, New York, NY, USA, 2006. ACM Press.

[KPRR03] Gerti Kappel, Birgit Pröll, Siegfried Reich, and Werner Retschitzegger, editors. *Web Engineering: Systematische Entwicklung von Web-Anwendungen.* dpunkt, 2003.

[Lim07] Acegi Technology Pty Limited. Acegi security [online]. 2007 [cited 20070416]. Available from: <http://acegisecurity.org/downloads.html>. Homepage of the Spring Acegi Subproject.

[MAC+00] Aaron Marcus, Nuray Aykin, Apala Lahiri Chavan, Donald L. Day, Emilie West Gould, Pia Honold, and Masaaki Kurosu. Cross-cultural user-interface design: what? so what? now what? In *CHI '00: CHI '00 extended abstracts on Human factors in computing systems*, pages 299–299, New York, NY, USA, 2000. ACM Press. Available from: <http://www.amanda.com/resources/hfweb2000/hfweb00.marcus.html> [cited 20061228].

[McL05] Drew McLellan. Very dynamic web interfaces [online]. 2005 [cited 20070413]. Available from: <http://www.xml.com/pub/a/2005/02/09/xml-http-request.html>. Article on XmlHttpRequest Technology.

[MF07] George Miller and Christiane Fellbaum. Wordnet: a lexical database for the english language [online]. 2007 [cited 20070118]. Available from: <http://wordnet.princeton.edu/>. Wordnet lexical database search service from Princeton University.

[MHC+96] Brad Myers, Jim Hollan, Isabel Cruz, Steve Bryson, Dick Bulterman, Tiziana Catarci, Wayne Citrin, Ephraim Glinert, Jonathan Grudin, and Yannis Ioannidis. Strategic directions in human-computer interaction. *ACM Comput. Surv.*, 28(4):794–809, 1996.

[Mic07a] Sun Microsystems. Javabeans spec [online]. 2007 [cited 20070411]. Available from: <http://java.sun.com/products/javabeans/docs/spec.html>. Homepage of the JavaBeans Specification.

[Mic07b] Sun Microsystems. Javaserver pages technology [online]. 2007 [cited 20070411]. Available from: <http://java.sun.com/products/jsp/index.jsp>. Homepage of the JavaServer Pages Technology.

[Mus07] PC Museum. Xerox alto [online]. 2007 [cited 20070417]. Available from: <http://members.fortunecity.com/pcmuseum/alto.html>. Description of the Xerox Alto.

[Pun06] Franz Puntigam. *Skriptum zur Objektorientierte Programmierung.* Technische Universität Wien, Institut für Computersprachen, October 2006. Available from: <http://www.complang.tuwien.ac.at/franz/objektorientiert/skript06-buch.pdf>.

[Shn97] Ben Shneiderman. Direct manipulation for comprehensible, predictable and controllable user interfaces. In *IUI '97: Proceedings of the 2nd international conference on Intelligent user interfaces*, pages 33–39, New York, NY, USA, 1997. ACM Press.

[SJK04] Janice R. Nall Sanjay J. Koyani, Robert W. Bailey. *Research-Based Web Design & Usability Guidelines.* Computer Psychology, August 2004. Available from: <http://www.amazon.com/Research-Based-Web-Design-Usability-Guidelines/dp/0974996904>.

[Som82] Ian Sommerville. *Software Engineering.* Addison-Wesley Pub. Co. London, 1982. Available from: <http://www.amazon.de/Software-Engineering-Ian-Sommerville/dp/3827370019>.

[SP04] Ben Shneiderman and Catherine Plaisant. *Designing the User Interface.* Addison-Wesley Longman, Mai 2004. Available from: <http://www.amazon.de/Designing-User-Interface-Ben-Shneiderman/dp/0321269780/sr=8-1/qid=1162727263/ref=sr_1_1/303-8002154-4485040?ie=UTF8&s=books-intl-de>.

[W3C99] W3C. Xsl transformations (xslt) version 1.0 [online]. 1999 [cited 20070413]. Available from: <http://www.w3.org/TR/xslt>. W3C Recommendation of XSL Transformations (XSLT) Version 1.0 Standard.

[W3C02] W3C. Xhtml 1.0 the extensible hypertext markup language (second edition) [online]. 2002 [cited 20070413]. Available from: <http://www.w3.org/TR/xhtml1/>. W3C Recommendation of XHTML 1.0 Standard.

[W3C04] W3C. Extensible markup language (xml) 1.0 (third edition) [online]. 2004 [cited 20070413]. Available from: <http://www.w3.org/TR/2004/REC-xml-20040204/>. W3C Recommendation of Extensible Markup Language (XML) 1.0 Standard.

[W3C06] W3C. Cascading style sheets, level 2 revision 1 [online]. 2006 [cited 20070413]. Available from: <http://www.w3.org/TR/CSS21/>. W3C Working Draft of CSS Standard.

[Wag05] Ina Wagner. *Skriptum zu Experimentelle Gestaltung von Multimedia-Anwendungen und Präsentationsstrategien.* Technische Universität Wien, Institut für Gestaltungs- und Wirkungsforschung, 2005. Available from: <http://www.media.tuwien.ac.at/i.wagner/lehre.php>.