LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
Lehr- und Forschungseinheit Medieninformatik
Prof. Dr. Heinrich Hußmann

Diplomarbeit

Unterstützung von Content-Management-Prozessen und Web-Engineering-Methoden durch Web 2.0 - Technologien

Christoph Metz
metzc@ifi.lmu.de

Bearbeitungszeitraum:	01.05.2006 bis 02.11.2006
Betreuer:	Richard Atterer
Verantw. Hochschullehrer:	Prof. Dr. H. Hußmann

Kurzzusammenfassung

In der vorliegenden Arbeit wird untersucht, wie die Entwicklung und Verwaltung von Webangeboten durch browserbasierte Werkzeuge unterstützt werden kann.

Dazu werden zunächst die im Zuge von Web 2.0 aufgekommenen neuen Möglichkeiten analysiert und auf ihre Auswirkung auf Webangebote im Allgemeinen hin sowie auf ihre Relevanz für ein browserbasiertes Entwicklungswerkzeug hin überprüft.

Anschließend folgt ein Überblick über die Ziele und Methoden des Web-Engineering. Ergänzend werden dabei verschiedene Studien zur Arbeitsweise von Webentwicklern sowie eine im Rahmen dieser Arbeit durchgeführte Umfrage zu browserbasierten Entwicklungswerkzeugen für Webangebote ausgewertet.

Schließlich wird ein Konzept für ein browserbasiertes Entwicklungswerkzeug namens W2MS vorgestellt und anhand eines Prototypen getestet. Eine optimale Ausnutzung technischer Möglichkeiten sowie die Integration von Web-Engineering-Ansätzen stehen dabei ebenso im Mittelpunkt wie eine Orientierung an den Gewohnheiten und Wünschen von Entwicklern im semi- bis nicht-professionellen Bereich.

Abstract

This thesis examines how the development and management of websites can be assisted through browser-based tools.

New possibilities emerged with the evolution of Web 2.0. These new possibilities are first analyzed and then examined with regard to their effect on websites in general and with regard to their relevance for a browser-based development tool.

This is followed by an overview of the goals and methods of web engineering. The overview includes a discussion of several studies looking at the way how web developers work as well as the interpretation of a survey about browser-based development tools for websites that was conducted as a part of this thesis.

Eventually, a concept for a browser-based development tool called W2MS is presented and a W2MS prototype is tested. W2MS focuses on the optimal use of technical possibilities, the integration of web engineering principles and the consideration of the habits and needs of semi- and non-professional developers.

Aufgabenstellung

Thema: Unterstützung von Content-Management-Prozessen und Web-Engineering-Methoden durch Web 2.0 - Technologien

Das Ziel der Diplomarbeit ist es einerseits, neue Möglichkeiten, die durch Web 2.0 und AJAX entstanden sind, auf Content-Management-Systeme anzuwenden, um die Benutzerfreundlichkeit im Webentwicklungsprozess zu verbessern; andererseits soll Augenmerk darauf gelegt werden, die Entwicklung von benutzerfreundlichen Webseiten zu unterstützen, bzw. sicherzustellen.

Es soll ein Konzept für ein CMS entwickelt werden, dass diese Ziele in der Planung und dem Layout von Webseiten, in der Content-Einpflegung und -Verwaltung, in Workflows, sowie in User- und automatischen Tests umsetzt. Aufsetzend auf eine Grund-Implementierung des CMS sollen einige innovative Aspekte des Konzepts implementiert werden.

Im Vorfeld soll ein Überblick über vorhandene CMS, sowie Webentwicklungsprozesse und -ansätze gegeben werden. Dabei sollen Probleme identifiziert, sowie neue Möglichkeiten skizziert werden, um diese dann in Bezug auf das Ziel der Diplomarbeit anzuwenden.

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig angefertigt, alle Zitate als solche kenntlich gemacht sowie alle benutzten Quellen und Hilfsmittel angegeben habe.

München, 2. November 2006

. . . . . .

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Das Web ist heute aus den meisten Lebensbereichen nicht mehr wegzudenken. Es ist das Informationsmedium Nummer 1 und schickt sich in letzter Zeit sogar an, dem Fernseher als Unterhaltungsmedium Konkurrenz zu machen. Immer mehr Personen treten dabei selbst als Betreiber von Webangeboten auf. So wird es bis zum Ende des Jahres 2006 wahrscheinlich über 100 Millionen Webangebote geben [NETCRAFT2006]. Es liegt nahe, dass das Web als Ganzes unter diesen Umständen in der Gefahr steht, deutlich an Qualität einzubüßen. Nielsen schrieb schon 1999:

> „Unless the vast majority of Web sites are improved considerably, we will suffer a usability meltdown of the Web no later than the Year 2000“ [NIELSEN1999, Seite 66].

Diese Prophezeiung hat sich in *dieser* Deutlichkeit nicht erfüllt, das von Nielsen thematisierte Problem ist heute aber mehr denn je vorhanden. In der Forschung gibt es verschiedene Bemühungen, um diesem „meltdown“ entgegenzuwirken, doch geschieht dies weit ab von dem einfachen „Webmaster“, der möglichst einfach und schnell ein eigenes Webangebot erstellen will. Es besteht also ein Bedarf an Werkzeugen, die die einfache Entwicklung und Verwaltung von Webangeboten unterstützen, dabei aber ein Mindestmaß an Qualität sicherstellen. Letztlich ist das Web die Summe aller Webangebote. Je mehr schlecht umgesetzte Webangebote es gibt, desto mehr wird die Benutzbarkeit des Web als Ganzem darunter leiden. Wenn das Web parallel dazu immer wichtiger für das tägliche Leben wird, so bedeutet das ein ernstes Problem.

1.2 Ansatz und Ziel dieser Arbeit

Der Titel dieser Arbeit kann in drei Komponenten aufgeteilt werden, die exemplarisch für Ansatz und Ziel dieser Arbeit stehen:

„Unterstützung ..“

Werkzeuge sind dazu da, dass sie den Benutzer dabei unterstützen zu *seinem* Ziel zu kommen. Wenn sie ihn jedoch dabei aufhalten, dann haben sie *ihr* eigentliches Ziel verfehlt. Auch in der noch relativ neuen Disziplin des Web-Engineering gibt es schon eine Vielzahl von Werkzeugen und Ansätzen, die jedoch von ihrer Zielgruppe – den Webentwicklern – nur sehr zurückhaltend wahrgenommen und angewendet werden. Es wird an dieser Stelle vermutet, dass der Grund hierfür darin liegt, dass diese Werkzeuge und Ansätze den Entwickler nicht als *Benutzer* im Auge haben. Dementsprechend stehen dabei nicht die eigentlichen Ziele und Wünsche des Entwicklers im Mittelpunkt, sondern übergeordnete Ziele, die für den Entwickler jedoch keine unmittelbare Relevanz haben. Ungeachtet dessen hätte es jedoch durchaus positive Auswirkungen, wenn Webentwickler dazu gebracht würden, bestimmte Ansätze und Methoden des Web-Engineering umzusetzen – sowohl für die Entwickler selbst, als auch für die Besucher ihrer Webangebote. In dieser Arbeit soll es daher um ein Werkzeug gehen, dass den Entwickler als *Benutzer* unterstützt, indem es Ansätze und Methoden des Web-Engineering mit *seinen Zielen* verbindet. Das dabei erstellte Werkzeug „W2MS“ wird in Kapitel 4 vorgestellt.

„.. von Content-Management-Prozessen und Web-Engineering-Methoden ..“

Um dem Webentwickler ein nützliches Werkzeug zur Verfügung stellen zu können, müssen zu-

nächst die Anforderungen und Möglichkeiten identifiziert werden, die im Bereich der Entwicklung von Webangeboten bestehen. Außerdem sollten, wie oben beschrieben, die tatsächlichen Ziele und Wünsche von Entwicklern untersucht werden. In Kapitel 3 wird daher dieser Themenkomplex näher beleuchtet. Web-Engineering-Methoden und -Ansätze aus der Forschung werden dabei genauso behandelt wie die tatsächlichen Arbeitsweisen von Entwicklern in der Praxis. In diesem Zusammenhang wurde im Rahmen dieser Arbeit auch eine Umfrage durchgeführt, die in diesem Kapitel ebenso wiedergegeben wird.

„... durch Web 2.0 – Technologien“

Das Web befindet sich in einer ständigen Bewegung. Die technischen Möglichkeiten entwickeln sich weiter und mit ihnen auch die Art der Benutzer sowie die Art der Benutz*ung*. Web 2.0 ist also nicht nur mit einer Technologie gleichzusetzen. Ein Anliegen dieser Arbeit ist es jedoch, im Zuge von Web 2.0 aufgekommene technische Ansätze zielführend zu nutzen. Dabei sollen nicht-technische Aspekte von Web 2.0 allerdings nicht außer Acht gelassen werden. Ein Werkzeug, das eine Unterstützung für die Entwicklung von Webangeboten sein soll, muss auch die aktuellen Gegebenheiten des Web berücksichtigen und gleichermaßen optimal nutzen. In Kapitel 2 wird daher der aktuelle Stand des Web von verschiedenen Seiten beleuchtet.

Es wird im Allgemeinen versucht, deutsche Begriffe ihren englischen Pendants vorzuziehen. Da man es jedoch innerhalb des Themenkomplexes dieser Arbeit hauptsächlich mit der englischen Sprache zu tun hat, ist dies nicht immer leicht. An manchen Stellen werden die englischen Begriffe beibehalten, da die deutschen Übersetzungen dort sehr unüblich sind. *Accessibility* wird aus diesem Grund z.B. nicht mit „Zugänglichkeit“ ersetzt. Es wurde versucht, diese Entscheidung jeweils sorgfältig abzuwägen. Es sei an dieser Stelle noch auf das Glossar verwiesen, dass einige für diese Arbeit wichtige Begriffsklärungen enthält.

2 Web 2.0 – neue Möglichkeiten, neue Anforderungen

2.1 Das Web wird 18 – ein kurzer Lebenslauf

In der frühen Zeit des Internets war von der Struktur, wie wir sie heute in Form des Web kennen, noch nicht viel zu sehen. Informationen waren oft nur über einen bestimmten Provider oder innerhalb einer Mailbox zugänglich. Die Vernetzung war also immer auf ein abgeschlossenes Netz beschränkt, in das sich eine beschränkte Anzahl von Nutzern, meist durch Einwahl mit einem Modem, angeschlossen hatten.

Die Idee, Dokumente weltweit verfügbar zu machen, entstand dann 1989 am CERN, als Tim Berners-Lee das Konzept von vernetzten Hypertexten vorstellte, die auf sogenannten Webservern vorgehalten werden. Diese Hypertext-Dokumente konnten über das Programm *www* von jedem Internetzugang aus abgerufen werden, allerdings funktionierte das ganze nur in einer Textkonsole. Einen Durchbruch erlebte die Idee erst mit dem graphischen Browser *Mosaic*, den ein Student 1993 für das *National Center for Supercomputing Applications* (NCSA) erstellte (vgl. [GRIBBLE2006]).

Einhergehend mit der massenhaften Zunahme privater Internetzugänge ergaben sich nun vorher nie gekannte Möglichkeiten – Informationen konnten nun von prinzipiell jedem einer prinzipiell unbegrenzten Menge von Leuten zugänglich gemacht werden. Es scheint nicht verwunderlich, dass die Anzahl der über das Web zugänglichen Dokumente sehr schnell in unüberschaubare Größenordnungen anstieg. 1991 wurde der erste Web-Server der Öffentlichkeit vorgestellt, 1992 gab es schon 50, 1999 waren 720.000 öffentlich zugängliche Webangebote bekannt, heute (2006) sind es ungefähr 100 Millionen und es werden pro Monat ungefähr fünf Millionen mehr (vgl. [NETCRAFT2006]). Laut Netcraft sind die jüngsten Zuwächse zu einem Großteil auf die Explosion von Blogging-Diensten (MySpace, Microsoft Live Spaces, Blogger) zurückzuführen. Dies weist darauf hin, dass der einfache Internetsurfer immer mehr auch zum Produzenten mit einem eigenen Webangebot wird.

2.2 Web 2.0 – eine Definition

Die Web-Gemeinschaft hat immer schon ihre *Buzzwords* gebraucht, um bestimmte Konzepte, Techniken oder Anwendungen auf einer breiten Ebene zu etablieren. *Web 2.0* stellt da keine Ausnahme dar und ist momentan wahrscheinlich das gewichtigste – wenn auch schon lange nicht mehr das neueste – Buzzword des Web. Da es der Natur eines Buzzwords entspricht, wenig spezifische Bedeutung zu enthalten, lässt es sich fast beliebig für jeden gerade gewünschten Zweck instrumentalisieren. Dementsprechend existieren eine Menge Definitionen, die nicht selten in einer Art Glaubenskrieg umkämpft werden. Obgleich es nicht Anliegen dieser Arbeit ist, diesen Kampf aufzunehmen, muss an dieser Stelle doch definiert werden, wie *Web 2.0* im Geltungsbereich dieser Arbeit zu verstehen ist.

Der Begriff Web 2.0 wurde ursprünglich im Jahr 2004 von O'Reilly während eines Brainstormings erfunden, um etwas zu beschreiben, was dort als „Wendepunkt für das Web“ wahrgenommen wurden. Später wurde eine gleichnamige Konferenz ins Leben gerufen, die sich fortan jährlich mit diesem Thema befasste. Der Verzicht auf eine klare und vor allem *kurze* Definition war bewußt und ist charakteristisch für das Konzept an sich – so geht die Referenzdefinition von Tim O'Reilly über mehrere Seiten [OREILLY2005]. Später ließ sich O'Reilly in seinem Blog jedoch zu einer Ein-Satz-Definition hinreißen, die als hier als Grundlage für die weiteren Überlegungen herangezogen werden soll:

> „Web 2.0 is the network as platform, spanning all connected devices; Web 2.0 applications are those that make the most of the intrinsic advantages of that platform: delivering software as a continually-updated service that gets better the more people use it, consuming and remixing data from multiple sources, including individual users, while providing their own data and services in a form that allows remixing by others, creating network effects through an 'architecture of participation,' and going beyond the page metaphor of Web 1.0 to deliver rich user experiences.” [OREILLY2005b]

Folgende Punkte sind hier bemerkenswert:

- Das Web wird als Plattform für Anwendungen und Dienste verstanden.
- Die Möglichkeiten dieser Plattform werden möglichst voll ausgenutzt.
- Der Benutzer wird als Teilnehmer, nicht nur als Konsument, verstanden.
- Webangebote werden nicht für sich isoliert, sondern als Teil des Netzwerks gesehen.
- Inhalte werden als eigenständige *Objekte* verstanden; die *Seite* ist nicht mehr grundlegende Dateneinheit, sondern nur ein möglicher Zugriffspunkt auf ein bestimmtes Objekt.

Es lassen sich drei verschiedene Dimensionen ausmachen, die hier von Bedeutung sind: eine *technische*, eine *strukturelle* und eine *soziale*. Diese werden im Folgenden kurz erläutert. Die Erläuterungen dienen zunächst nur als Überblick und haben keinen Anspruch auf Allgemeingültigkeit. Im weiteren Verlauf dieses Kapitels werden dann einzelne Aspekte – sofern relevant - im Detail behandelt und auf die Fragestellungen dieser Arbeit angewandt.

2.2.1 Die technische Dimension

Die technische Dimension von Web 2.0 umfasst das aktuell im Web technisch Machbare. Das können durchaus Dinge sein, die zwar schon lange theoretisch verfügbar sind, aber bisher nur von wenigen umgesetzt worden sind. Das prominenteste Beispiel dafür ist *AJAX*, das an sich nichts Neues ist, aber durch die Schaffung des Begriffs in das Bewusstsein einer breiten Entwickleröffentlichkeit gelangt ist.

Web 2.0 bedeutet in dieser Dimension, dass zur Umsetzung einer Anforderung die beste aktuell zur Verfügung stehende technische Möglichkeit verwendet wird.

2.2.2 Die strukturelle Dimension

Die strukturelle Dimension ergibt sich zu einem gewissen Grad aus der technischen, zumindest sind Veränderungen und Entwicklungen hier meist auf neue, bzw. neu entdeckte technische Möglichkeiten zurückzuführen. So kann man sich durch die Verwendung von AJAX zu einem gewissen Grad von der „Seiten-Metapher“ [OREILLY2005b] losmachen, die in den Ursprüngen des Web und dem technischen Aufbau von HTTP begründet ist. Die „Seite“ ist nun nicht mehr das zentrale Inhaltselement, sondern nur noch ein Container für bestimmte Objekte. Darüberhinaus sind Interaktionen des Benutzers mit dem Server mit AJAX nicht mehr zwingend mit dem Neuladen oder Wechsel der Seite verbunden.

Die „Objekt-Orientierung“ und die direktere Client/Server-Kommunikation lassen schon von sich aus Assoziationen mit der Software-Welt aufkommen. Und in der Tat verstehen sich aktuelle Webangebote nicht selten als „Anwendung“, zumindest aber als „Dienst“. Darauf aufbauend - und im Zusammenspiel mit Techniken wie RSS, SOAP, RPC - fügen sich Webangebote außerdem in ein großes verteiltes System ein, in dem Daten (Objekte) ausgetauscht oder

Funktionalität angeboten wird.

Web 2.0 bedeutet hier also, dass Webangebote Dienste auf Objekten innerhalb eines großen verteilten Systems (dem Web) zur Verfügung stellen.

2.2.3 Die soziale Dimension

Die soziale Dimension ergibt sich schließlich zu großen Teilen aus den Veränderungen in der strukturellen Dimension. Die Benutzer beginnen das Web als System von verknüpften Diensten zu verstehen, in dem Identitäten und Objekte unabhängig von „Seiten" existieren. Der Web 2.0 - Benutzer ist in diesem System mit einer virtuellen Identität präsent, indem er überall persönliche Daten in Form von Blog-Einträgen, Videos und Kommentaren, Ratings oder Tags hinterlässt. Durch jeden interagierenden Benutzer wird das Gesamtnetz verändert. Durch die technischen Möglichkeiten und strukturellen Gegebenheiten ist es dem normalen Benutzer zudem ermöglicht, selbst ein Webangebot zu unterhalten.

Im Sinne von Web 2.0 wird das Web als eine Plattform verstanden, auf der alle Akteure als aktive Teilnehmer auftreten. Das Web ist zu *dem* weltweiten sozialen Netz geworden ist, dass Auswirkungen weit über die technische Domäne hinaus hat. Als soziales Netz ist es außerdem gesellschaftlichen Problemen, Prozessen und Herausforderungen ausgesetzt. Dies ist vielleicht der gewichtigste Punkt von allen.

2.2.4 Auf den Punkt gebracht ...

Ausgehend von diesen Beobachtungen, wird *Web 2.0* nun wie folgt definiert:

> *Web 2.0* steht symbolhaft für alle Webangebote, die dem aktuellen Entwicklungsstand des Web - in technischer, struktureller und sozialer Dimension – in sinnvoller Weise entsprechen.

Der Zusatz „in sinnvoller Weise" soll darauf hindeuten, dass die gedankenlose Verwendung alles *Neuen* natürlich nicht Ziel des Ganzen ist. Es geht eher darum, dass alle zur Verfügung stehenden Möglichkeiten zur Realisierung der Anforderungen eines bestimmten Webangebots in Erwägung gezogen werden und dann die in dem speziellen Fall optimalste Wahl getroffen wird. Für das Webangebot einer Zeitung würde es wahrscheinlich keinen Sinn machen, den Inhalt von den Besuchern generieren zu lassen (obwohl es solche Nachrichtenseiten durchaus gibt, siehe z.B. www.newsvine.com) – aber es könnte z.B. ein intelligentes soziales Bewertungssystem integriert werden, das die Benutzungsqualität des Angebots entscheidend erhöhen würde.

Die oben genannte Definition mag trivial erscheinen, unterscheidet sich gerade deswegen jedoch von manchen langen Ausführungen zu diesem Thema, die eine klar definierte Art von Webanwendungen skizzieren und damit sehr einseitig abgrenzen. Das Web ist ein sich in der Entwicklung befindendes Medium – der Entwicklungsstatus ergibt sich dabei ständig aus der „Benutzungsrealität" aller Teilnehmer. Wenn dabei eine neue Technik, ein neues Konzept oder eine neue Benutzungspraxis aufkommt und sich bewährt, wird das zum Teil des Web – und muss nicht extra in eine neue „Web 2.0 Definition" integriert werden. Andererseits müssen schon lange etablierte Konzepte nicht zwangsläufig als „Web 1.0" beschimpft werden, wenn sie in einem bestimmten Bereich heute immer noch die beste Alternative darstellen. Diese Sicht auf Web 2.0 unterstützt auch der Erfinder des Web, Tim Berners-Lee. In einem Interview äußerte er kürzlich, dass Web 2.0 im Prinzip nichts anderes ist als das, was das Web (1.0) schon immer war – nämlich: „... a collaborative space where people can interact" [BERNERSLEE2006]. Er führt weiter aus, dass das, was „Web 2.0" genannt wird, in Wirklichkeit nur in der konsequenten Verwendung der Standards besteht, die in den letzten Jahr(zehnt)en entwickelt worden sind. In

der Tat hatte Berners-Lee schon 1996 in seinem Artikel „The World Wide Web: Past, Present and Future“ das Web als einen Raum für Interaktionen zwischen Leuten beschrieben und dabei viele Konzepte vorgestellt, die heute auch so in herkömmlichen Web 2.0 Definitionen zu finden sind [BERNERSLEE1996].

Entgegen der Definition von O'Reilly wird Web 2.0 in dieser Arbeit also nicht als eine neue oder andere Art von Webangeboten verstanden. Das würde keinen Sinn machen, denn es gibt viele Klassen von Webangeboten, die nicht der O'Reilly-Definition entsprechen, aber trotzdem sinnvoll sind und bleiben. Es gibt keinen Grund diese als „Web 1.0“ zu deklassieren. Daraus folgt, dass hier - entgegen der „2.0“-Assoziation - kein Paradigmen-Wechsel im Web vorliegt. Vielmehr beschreibt Web 2.0 neue, zu „Web 1.0“ komplementäre, Prinzipien und Techniken, deren sich ein Webangebot ganz oder auch nur teilweise bedienen kann.

Für die Disziplin des Web-Engineering bedeutet das einfach, stets mit dem Status-Quo des Web auf Augenhöhe zu bleiben. Diese Sichtweise macht im Hinblick auf Web-entwicklungswerkzeuge am meisten Sinn, da diese dem Entwickler helfen sollen, seine Vorstellungen auf die bestmögliche Weise umzusetzen, ohne ihm dabei ein Konzept (ob neu oder alt) aufzuzwängen.

Im Hinblick auf das Alter des Web bietet sich hier, wie in den Worten des Titels von Abschnitt 2.1 („Das Web wird 18“) angedeutet, die Analogie zum „Erwachsenwerden“ an. Nimmt man 1989 als Gründungsjahr, wird das Web im Jahr 2007 18 Jahre alt – die „Volljährigkeit“ ist also bald erreicht, das „Jugenddasein“ aber noch nicht überwunden. Um die Analogie noch ein wenig weiter zu treiben: Das Web ist schon ein gutes Stück gereift, hat sich kontinuierlich verändert, hat seine ersten Erfahrungen gemacht und Fähigkeiten entwickelt, wird aber hoffentlich nicht auf diesem Stand stehenbleiben und vielleicht auch schlechte Angewohnheiten wieder aufgeben.

Im Kontext dieser Arbeit ist es natürlich von Interesse, die Konsequenzen von Web 2.0 für Webentwicklungswerkzeuge zu untersuchen. Relevante Aspekte von Web 2.0 werden im Folgenden im Detail behandelt. Die Frage, inwieweit aktuelle Werkzeuge überhaupt schon bei Web 2.0 „angekommen“ sind und wie entsprechende Werkzeuge aussehen könnten oder sollten wird dann in späteren Kapiteln beleuchtet.

2.3 AJAX

Der Browser als Benutzerschnittstelle zum Web und das HTTP-Protokoll als Kommunikationsbasis sind zunächst erstmal auf die Anzeige für entfernt gelagerte Dokumente hin ausgelegt und bieten daher keine sehr geeignete Umgebung für komplizierte Benutzerschnittstellen. Stark vereinfacht ausgedrückt funktioniert das Web so, dass der Anwender nichts anderes machen kann, als jeweils immer ein Dokument anzufordern, dass ihm dann der Browser über HTTP vom Server lädt und anzeigt. Für interaktive Webanwendungen sind das natürlich denkbar ungünstige Gegebenheiten. Herkömmliche Webangebote sind um das Prinzip herum aufgebaut, dass bei jeder Interaktion des Anwenders (die eine Kommunikation zum Server erfordert) eine neue Seite vom Server angefordert wird, der die Interaktion dann auswertet und die aktualisierte Oberfläche als Antwort komplett zurückschickt (Abbildung 2.1, *classic web application model*).

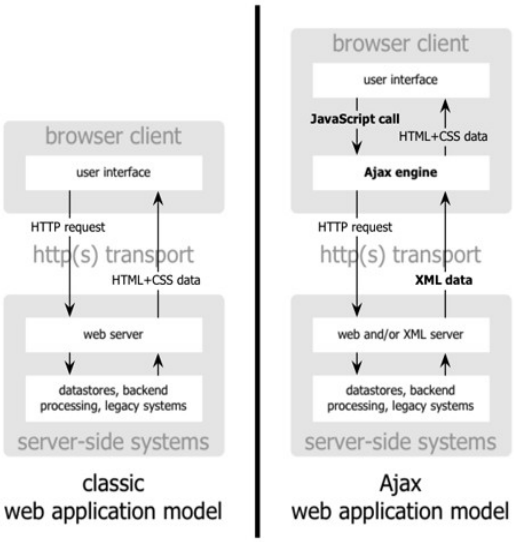


Abbildung 2.1: classic and Ajax web application model, Quelle: [GARRETT2005]

Durch ein JavaScript-Objekt namens *XMLHttpRequest* kann dieses Neuladen der Browserseite allerdings umgangen werden. Werden über dieses Objekt Anfragen zum Server geschickt, erfolgt die Kommunikation asynchron im Hintergrund. Sobald die Antwort des Servers eingetroffen ist, wird sie dann an eine Callback-Funktion übergeben, die diese dann auswerten und ggf. die Oberfläche anpassen kann. Ein Neuaufbau der Browserseite findet dabei nicht statt.

Folgendes Code-Beispiel (JavaScript) zeigt exemplarisch die Funktionsweise von XMLHttp-Request:

```
xmlhttp = new XMLHttpRequest();
xmlhttp.open("POST", "test.php",true);
xmlhttp.onreadystatechange=function() {
      if (xmlhttp.readyState==4) {
            alert(xmlhttp.responseText)
      }
}
xmlhttp.send('mydata=9324');
```

Der entscheidende Punkt liegt hierbei in dem Event „onreadystatechange", das an das XML-HttpRequest-Objekt gebunden wird: sobald der Server die Antwort schickt, wird die Callback-Funktion ausgeführt - die Kommunikation findet also asynchron statt und die Oberfläche auf dem Client läuft davon unberührt weiter. Es muss bei einer Interaktion also nicht mehr die ganze Seite/Oberfläche neu übertragen werden, sondern nur die wirklich relevanten Nutzdaten. Die Anpassung der Oberfläche muss dann über JavaScript beim Client erfolgen (vgl. Abbildung 2.1).

Die Existenz dieser Möglichkeit war jedoch lange relativ unbekannt - bis Jesse James Garrett Anfang 2005 den Artikel „Ajax: A New Approach to Web Applications" im Web veröffentlichte [GARRETT2005]. In dem Artikel erklärt er die grundlegende Funktionsweise einer Webanwendung mit asynchroner Kommunikation und fasst das Ganze in dem Begriff *AJAX* zusammen. *AJAX* soll für „Asynchronous JavaScript and XML" stehen und wird heute als Synonym für das oben beschriebene Prinzip verwendet.

2.3.1 Flüssige Interaktionen

Durch den Neuaufbau der kompletten Seite bei dem klassischen Kommunikationsmodell entsteht für den Benutzer eine Wartezeit, je nach Verbindung und Arbeitsaufwand für den Server. In dieser Zeit sieht der Benutzer nur eine weiße Seite und kann demnach auch nicht weiter mit der Oberfläche interagieren. Selbst wenn diese Wartezeit minimal ist, entsteht ein Flicker-Effekt, der für den Benutzer störend ist und das Benutzungsgefühl deutlich verschlechtert.

Durch Verwendung von AJAX kann dieses Problem komplett umgangen werden. Anfragen zum Server laufen im Hintergrund und stören nicht den Interaktionsfluss des Benutzers. Dies ist ein sofort spürbarer Komfort- und Geschwindigkeitsgewinn für den Benutzer (vgl. Abbildung 2.2).

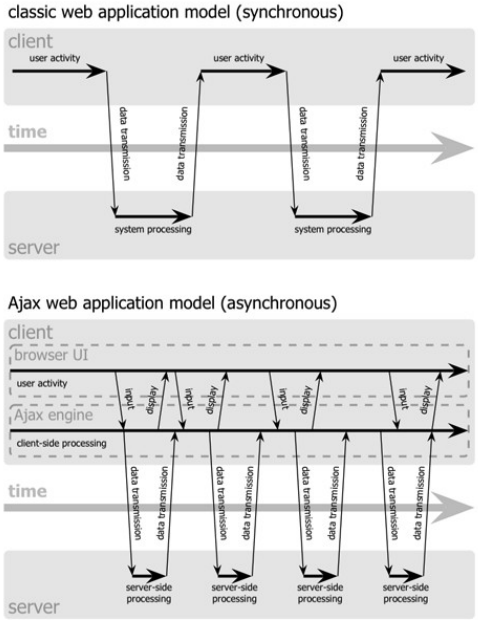


Abbildung 2.2: Synchrone und asynchrone Datenübertragung, Quelle: [WIKIPEDIA2006]

2.3.2 AJAX und REST

HTTP ist ein zustandsloses Protokoll. Das bedeutet, dass jede Anfrage, die über HTTP an einen Server gestellt wird, für sich alleine steht und nicht im Kontext einer längeren Kommunikation. Das bedeutet für eine Webanwendung ohne AJAX, dass bei jeder Verbindung zum Server der Status der Anwendung auf dem Client komplett gelöscht wird, da die Browserseite neu geladen wird. Um die Oberfläche wieder genau so herzustellen wie vor der Verbindung, muss entweder der Client bei einer Anfrage sämtliche Statusinformationen der Anwendung stets mitschicken (auch *kontext-vollständige Anfrage* genannt), oder der Server muss diese Daten serverseitig speichern (auch *Sessions* genannt). Kontext-vollständige Anfragen sind bei komplexeren Anwendungen nicht mehr praktikabel, bzw. nicht ohne weiteres realisierbar. Daher werden in vielen Fällen Sessions benutzt.

Es liegt auf der Hand, dass der komplette Neuaufbau der Oberfläche (das ist im Prinzip ein *Neustart* der Anwendung) bei jeder Verbindung sehr ineffizient ist. Es stellt für den Server einen beträchtlichen Rechenaufwand dar, ständig die gesamte Oberfläche aus den Zustandsdaten dynamisch generieren zu müssen. Außerdem ist die Sicherung bzw. das „Mitschleppen“ des

Anwendungszustandes eine lästige und aufwändige Angelegenheit. Bei komplexen DHTML-Anwendungsoberflächen ist zudem die 1:1 Wiederherstellung der Oberfläche nach einem Neuladen der Browserseite nicht immer möglich.

Die Verwendung von Sessions hat außerdem zur Folge, dass die HTTP-Anfragen nicht mehr idempotent sind, d.h. dass die gleiche Anfrage – je nach Zustand auf dem Server – unterschiedliche Ergebnisse haben kann. Dies ist jedoch eine Grundvoraussetzung für das Caching von Anfragen. Die Verwendung von Sessions schließt also Caching aus – das bedeutet wiederum eine beträchtliche zusätzliche Serverlast, da jede einzelne Anfrage immer wieder neu berechnet werden muss.

Ein Architekturprinzip, das die oben beschriebenen Probleme von dynamischen Webanwendungen verhindern soll, ist REST (Representational State Transfer). Der zentrale Punkt dieses Prinzips ist die Idempotenz von Anfragen, also in der Konsequenz ein Server ohne Client-Sessions. Das bedeutet, dass bei einer REST-Anwendung alle Anfragen zwischengespeichert werden können, was wiederum die Performanz und Skalierbarkeit der Webanwendung sicherstellt. Darüber hinaus kann die Architektur eines Servers ohne Client-Sessions recht einfach gehalten werden. Webanwendungen, die nur mit kontext-vollständigen Anfragen arbeiten, sind demnach REST-konform.

Im Prinzip ist eine REST-Architektur sehr wünschenswert – wenn man dabei nicht komplett auf serverseitige Sessions verzichten müsste. REST steht für Performanz und Skalierbarkeit, Sessions stehen für komplexe, hoch-personalisierte Webanwendungen.

Mit AJAX ist nun eine neue Möglichkeit hinzugekommen, die die Vorteile von beiden Seiten vereint. Da mit AJAX das Neuladen der Browserseite nicht mehr notwendig ist, lassen sich sog. *single-page applications* (im Folgenden SPA) realisieren. Eine SPA kommt während der gesamten Laufzeit der Anwendung ohne Neuladen aus, d.h. sämtliche Verbindungen zum Server laufen asynchron im Hintergrund ab. Der entscheidende Vorteil dabei ist, dass die Seite, d.h. die Programmlogik auf dem Client, ohne Unterbrechung läuft. Bisherige Webanwendungen sind so strukturiert, dass die Anwendungslogik auf dem Server läuft und der Browser nur als „Ausgabegerät“ für die Anwendung auf dem Server fungiert – also als eine Art Monitor. Auf dem Client wird dabei nur vereinzelt Programmcode eingesetzt, im Prinzip gilt es als anzustrebendes Ideal, dass eine Webangebot ohne JavaScript lauffähig sein soll.

AJAX ermöglicht es nun, diese Architektur grundlegend zu ändern. Die Programmlogik kann komplett auf den Client ausgelagert werden – der Server fungiert dabei (überspitzt ausgedrückt) nur noch als Datenbank, die man für das Lesen oder Speichern von Daten kontaktiert. Dadurch kann die Architektur auf dem Server sehr einfach gehalten werden. Vor allem aber erfüllt er damit das REST-Prinzip, da der Client seine Session selbst verwaltet und nur idempotente Anfragen an die „Datenbank“ schickt (vgl. [HIGGINS2006] und Abbildung 2.3). Durch AJAX können also hoch-personalisierte dynamische Webangebote performant und skalierbar realisiert werden, da die meiste Arbeit auf den Client ausgelagert werden kann und der Server zudem durch die REST-Konformität durch Caching in allen Varianten (direkt auf dem Server, in Proxys, im Browsercache) wesentlich entlastet werden kann. Eine REST-konforme SPA verwendet die HTTP-Kommunikation genau in dem ursprünglichen Sinn des Protokolls und nutzt somit seine Stärken voll aus.

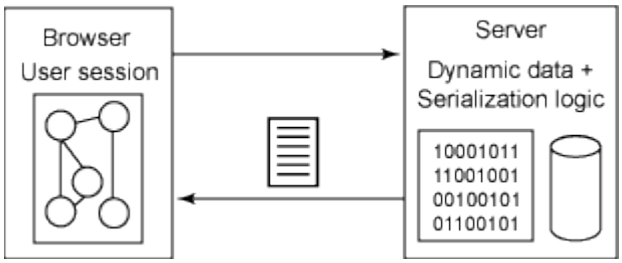


Abbildung 2.3: An immersive Ajax application, Quelle: [HIGGINS2006]

Natürlich gibt es auch hier ein „Aber“. JavaScript ist zwar besser als sein Ruf, aber mit ausgewachsenen Programmiersprachen kann es nicht mithalten. Ein großes Problem ist z.B. das Debugging, das zusammen mit einem schlechten Programmierstil schon mal zu einer unlösbaren Angelegenheit werden kann. Mittlerweile gibt es jedoch schon sehr hilfreiche Werkzeuge in diesem Bereich, z.B. die Firefox-Erweiterung FireBug [HEWITT2006]. Der Debugging-Komfort liegt dabei schon mindestens auf dem Niveau von aktuellen PHP-Debuggern (subjektive Empfindung des Autors), reicht jedoch nicht an den Komfort einer vollwertigen Java-Entwicklungsumgebung heran. Zudem beschränkt sich dieses Werkzeug auf den Browser Firefox, was auf ein weiteres, sehr großes Problem hinweist: die Browserkompatibilität. Die unterschiedliche Funktionsweise der verschiedenen Browser ist eines der größten Hindernisse für die Entwicklung von komplexen Webangeboten (vgl. 3.7.2) und damit natürlich auch im Besondern für SPAs. Abhilfe schaffen hier Frameworks und Toolkits, auf die der Entwickler aufsetzen kann (siehe 3.5.1 und 3.5.2).

Ein Haken bei der Sache ist, dass bei einer SPA der gesamte Quellcode - also die Anwendungslogik - zum Client übertragen werden muss, da Java*Script* nicht vorkompiliert wird, wie etwa ein Java-Applet. Hierzu gibt es zwar Methoden, den Quellcode unlesbar zu machen, allerdings wird dadurch ein Reverse-Engineering nur erschwert, aber nicht vollständig verhindert. Für einige Betreiber könnte das ein unüberwindbares Hindernis bedeuten.

2.3.3 Trennung von Daten und Präsentation

Die Trennung von Daten und Präsentation ist eine Tugend, die natürlich auch unabhängig von AJAX und SPAs zu Recht hochgehalten wird. Und doch ist sie in besonderem Maße charakteristisch für Web 2.0. Durch AJAX kann dieses Prinzip schließlich so erweitert werden, dass die Präsentation auch komplett vom Server getrennt wird. Zwar müssen auf dem Server zwangsläufig Präsentationsinformationen wie Templates und Stylesheets gespeichert werden. Diese sind aber statischer Natur, müssen nur einmal geladen werden und sind fortan im Cache des Client verfügbar. Während der Laufzeit einer SPA werden nur noch relevante Nutzdaten übertragen. Die Anwendungslogik auf dem Client nimmt diese Daten dann entgegen, interpretiert sie und kümmert sich dann um die richtige Präsentation der Daten bzw. Anpassung der Oberfläche (vgl. Abbildung 2.1). Der Server muss die Daten also nicht mit Layout-Informationen verknüpfen oder vermischen. Das hat wiederum zum Vorteil, dass dieselben Daten für verschiedene Zwecke verwendet werden können. Die rechenintensive Befüllung von Templates auf dem Server entfällt also völlig. Ein weiterer offensichtlicher Vorteil ist die Einsparung von Traffic-Volumen. Die Präsentationsdaten werden (übrigens genauso wie die Programmlogik) nur einmal beim Aufruf des Webangebots übertragen.

Es sei hier erwähnt, dass dieses Prinzip in Form von CSS-Sheets schon seit langem etabliert ist. Die Übertragung von Nutzdaten anstatt von HTML-Code (also von einem mit Daten befüllten Template) ist jedoch nur durch die Verwendung von AJAX zu realisieren.

2.3.4 AJAX und die History

Ein Kritikpunkt an AJAX war von Anfang an, dass die Durchführung von Anfragen ohne Neuladen der Browserseite die History des Browsers durcheinanderbringt. Anfragen, die über XMLHttpRequest getätigt werden, werden nicht in die History eingetragen. Demzufolge beziehen sich die Browseraktionen Vor, Zurück und Neuladen immer auf das letzte Neuladen der Seite und nicht etwa auf den letzten XMLHttpRequest. Genauso ist es mit der Bookmark-Funktion, die einen Verweis auf die Anfrage setzt, die zum letzten Neuladen der Seite geführt hat. Zu erwarten wäre eigentlich, dass ein Verweis auf die aktuelle Ansicht gesetzt wird.

Bevor auf diese Punkte eingegangen, wird sollen hierzu jedoch einige relativierende Fakten genannt werden. Das History-Problem gibt es eigentlich schon so lange, wie es interaktive Webangebote gibt, nur ist es nicht so deutlich aufgetreten. Hat man z.B. auf einer Webseite ein POST-Formular abgeschickt, so führte die Zurück-Aktion zwar zur vorherigen Ansicht, machte die getätigte Aktion aber nicht rückgängig. Die Vorwärts-Aktion würde daraufhin Daten nochmal übertragen, oder die Aktion verweigern – in den meisten Fällen würde man hier also entweder einen Fehler machen (das Formular ein zweites Mal abschicken) oder nicht mehr weiterkommen. Der Browser hat zwar gute Gründe, sich so zu verhalten, aber aus der Sicht der Webanwendungen kann nicht die Rede davon sein, dass die Browser-History jemals das geleistet hätte, was man bei einer interaktiven Webanwendung erwarten würde. Ähnlich ist es auch mit der Bookmark-Funktion. Wie schon oben erläutert, sind Webanwendungen, die Sessions verwenden, nicht REST-konform, d.h. die Anfragen an den Server sind nicht idempotent. Ein Bookmark wird also auch hier nicht einen Verweis auf die aktuelle Ansicht abspeichern. Noch ein Punkt, der hier zwar erwähnt, aber nicht vertieft werden soll: Die Verwendung von Frames bedingt ähnliche Probleme, wie die an AJAX kritisierten. Bedenkt man, dass Frames oft dann verwendet werden, um ähnliche Effekte wie durch AJAX zu erzielen, wird klar, dass die richtige Funktionsweise der Browser-History ein grundsätzliches Architekturproblem bei Webanwendungen ist und nicht bloß ein Nischenproblem von AJAX. Im Gegenteil – mit ein wenig Aufwand ist mit AJAX sogar eine wirklich sinnvolle und für den Benutzer intuitive Verwendung der History möglich.

Die Lösung besteht darin, dass man sich anhand verschiedener Tricks Kontrolle über die History des Browser verschafft und dann bei jeder History-Aktion durch die Programmlogik entscheidet, was genau zu tun ist. Es gibt verschiedene Frameworks, die einem die Tricks abnehmen (vgl. 3.5.1) – der Entwickler muss sich dann „nur“ noch darum kümmern, dass seine Anwendung bei jeder History-Aktion etwas sinnvolles macht. Alle Frameworks machen sich im Prinzip zwei Tricks zunutze:

- Ein verstecktes IFrame, dass nicht dynamisch erzeugt wurde, trägt seine History-Vorgänge in die History der Hauptseite ein. In dem IFrame können also fingierte Aktionen durchgeführt werden, um Einfluss auf die History der Hauptseite zu nehmen, ohne dass Änderungen an der Oberfläche bemerkbar sind.

- Sprunglinks (Anchors), haben Einfluss auf die History. Da Sprunglinks konzeptionell dafür da sind, *innerhalb* einer Seite zu verlinken, ohne ein Neuladen hervorzurufen, eignen sie sich hervorragend für History-Operationen in SPAs.

Das Bookmark-Problem kann durch letztere Möglichkeit auch elegant gelöst werden. Sobald sich die Ansicht in einer Webanwendung ändert, muss die Programmlogik dafür sorgen, dass die Seite zu einem bestimmten Sprunglink springt. Der Name des Sprunglinks wird dabei mit einem „#“ an die URL in der Adressleiste des Browsers angehängt. Setzt man nun einen Bookmark, wird der Name des Sprunglinks mitgespeichert. Beim Öffnen des Bookmarks muss die Programmlogik dann natürlich dafür sorgen, dass der Name des Sprunglinks wieder in die richtige Ansicht umgesetzt wird.

Hat man in der Webanwendung über oben genannte Tricks, bzw. mit Hilfe entsprechender Frameworks, die History „unter Kontrolle“ gebracht, bleibt immer noch eine große Herausforderung – denn automatisch passiert jetzt nichts mehr. Eigentlich ist die History eine automatische Funktion des Browsers, um die man sich nicht kümmern muss. Übernimmt man die Kontrolle, hat man jedoch genau diese Automatik deaktiviert – und muss sich nun um alles selber kümmern. Mit einem intelligenten Aufbau kann man dieses Problems aber durchaus Herr werden (siehe hierzu Kapitel). Auf jeden Fall hat man nun alle Möglichkeiten, um das für die jeweilige Anwendung sinnvolle Verhalten zu erreichen. Eine Vorwärts-Aktion muss nun nicht mehr zwingend das Formular neu übermitteln, eine Rückwärts-Aktion kann nun die vorher getätigte Aktion wirklich rückgängig machen – der Entwickler muss es ihr nur beibringen.

2.3.5 AJAX und Accessibility

Im Kontext von AJAX verbirgt sich hinter Accessibility das Problem, dass in einigen Browsern oder z.B. auch in Screenreadern JavaScript deaktiviert oder nicht verfügbar ist. In wenigen Fällen ist zwar JavaScript vorhanden und aktiviert, aber XMLHttpRequest wird nicht unterstützt. Will man diese Nutzer nicht ausschließen, muss ein Alternativzugang eingerichtet werden, der zumindest eine Minimalfunktionalität ohne JavaScript bzw. AJAX ermöglicht. Dies ist für einige Betreiber ein entscheidendes Hindernis, um AJAX in ihren Webanwendungen zu nutzen.

Auf das Thema Accessibility wird im Verlauf dieser Arbeit noch an verschiedenen Stellen eingegangen.

2.3.6 Fazit

Zusammenfassend kann man sagen, dass die *intelligente* Verwendung von AJAX sehr viel schnellere Web-Oberflächen mit einer von Desktop-Anwendungen bekannten Nutzungserfahrung (siehe Abbildung 2.2), sowie eine performante, skalierbare und einfach strukturierte Architektur ermöglicht.

2.4 Comet

Comet ist eine weniger bekannte Technik, die aber – genauso wie AJAX – schon lange technisch *möglich* ist, jedoch nur von einigen „Insidern“ beherrscht wurde. Die Tatsache, dass Comet noch sehr unbekannt ist, liegt wahrscheinlich daran, dass zum einen erst Anfang 2006 jemand einen Begriff dafür definiert hat (vgl. [RUSSELL2006]) und dass es zum anderen nicht so viele Einsatzgebiete dafür gibt, wie z.B. für AJAX. Außerdem ist es ungleich schwerer zu implementieren als AJAX.

Comet ermöglicht es dem Server, jederzeit bei Eintreffen eines Events mit dem Client (Browser) zu kommunizieren. Eigentlich ist das absolut unvereinbar mit dem HTTP-Protokoll, das die Verbindungsaufnahme nur vom Client aus kennt. Da HTTP, wie oben schon erläutert, ein zustandsloses Protokoll ist, kann der Server auch nicht nach einer ersten Verbindungsaufnahme des Clients von sich aus eine Verbindung aufbauen. Damit der Server nun jederzeit Kontakt mit dem Client aufnehmen kann, muss die vom Client einmal aufgebaute Verbindung einfach offengehalten werden (andere Namen für dieses Prinzip sind daher auch *long lived Http* und *slow load technique*). Auf diese Weise kann der Server bei Bedarf Daten in die offene Verbindung hineinschreiben. Abbildung 2.4 verdeutlicht die Kommunikationsabläufe einer Comet Webanwendung (siehe hierzu auch im Vergleich Abbildung 2.2). Entscheidend ist hier, dass Events, die beim Server auflaufen, ohne Zeitversatz zum Client geschickt werden können. Eine klassische Anwendung hierfür ist z.B. eine Chat-Anwendung. Sendet ein Client zu einem anderen einen Textzeile, wird diese in einer Comet-Webanwendung ohne Zeitversatz beim

Empfänger angezeigt – natürlich abgesehen von den Verbindungszeiten.

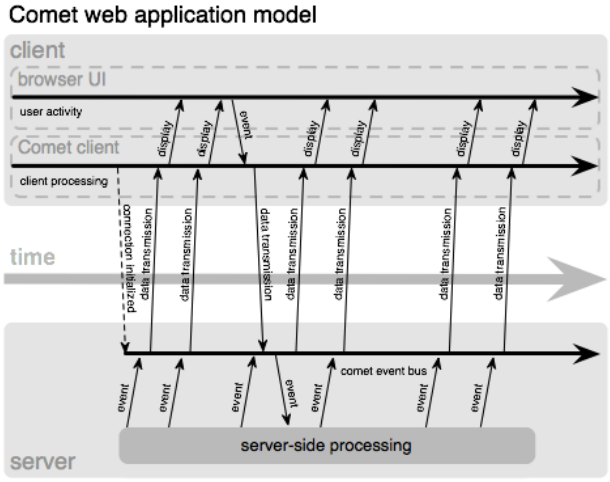


Abbildung 2.4: Comet web application model, Quelle: [RUSSELL2006]

Die Alternative hierzu wäre *Polling*, also das ständige Nachfragen beim Server, ob neue Events aufgelaufen sind. In manchen Anwendungsfällen wird diese Variante effizienter sein, vor allem wenn ein gewisser Zeitversatz zwischen Eintreffen eines Events und Übertragung zum Client akzeptabel ist. Jedenfalls bedeuten die ständigen HTTP-Anfragen eine beträchtliche Last für den Server. Comet steht an diesem Punkt jedoch auch nicht besser da, da für jeden aktiven Client eine Verbindung offen gehalten werden muss. Beide Alternativen sind auf eine andere Art hungrig nach Ressourcen und stellen daher ein Problem für die Skalierbarkeit einer Webanwendung dar. Je nach gewählter Alternative sollte der Server also entsprechend ausgewählt oder konfiguriert werden.

Für die Implementierung von Comet gibt es verschiedene Möglichkeiten. Im Detail wird darauf jedoch in Abschnitt 4.2.1 eingegangen. Gemeinsam ist allen Ansätzen jedoch, dass eine HTTP-Verbindung zwischen Client und Server ständig offen gehalten wird. Neben der Ressourcen-Belegung auf dem Server hat dies jedoch noch einen weiteren Nachteil. HTTP 1.1 sieht vor, dass jeder Client (Browser) nur zwei gleichzeitige Verbindungen zu einem Server aufbauen kann. Durch Comet wird aber nun eine davon ständig belegt, d.h. die Webanwendung muss mit *einer* verbleibenden Verbindung auskommen. Das mag auf den ersten Blick kein Problem darstellen, wird aber dann zum Problem, wenn mehrere Anfragen parallel laufen *könnten*, aber auf Grund dieser Beschränkung sequentiell abgefertigt werden müssen. Das ist z.B. der Fall, wenn mehrere Bilder auf der Oberfläche angezeigt werden (jedes Bild wird in einer eigenen HTTP-Verbindung vom Server geladen), oder wenn viele XMLHttpRequests anfallen, z.B. durch schnell aufeinander folgende Benutzerinteraktionen. Umgangen werden kann dieses Problem, indem verschiedene Server eingesetzt werden. Es reicht aber auch schon, mit verschiedenen Subdomains zu arbeiten, die alle auf den gleichen Server verweisen. Sinnvoll ist hier ein eigener Server (bzw. Subdomain) für Bilder und andere Medien, ein weiterer für die Comet-Verbindung und schließlich natürlich einer für sonstige Serveranfragen (im Fall einer SPA also für XMLHttpRequests).

Comet widerspricht der ursprünglichen Dokumenten-zentrierten Architektur des Web natürlich in noch größerem Maße als AJAX es tut. Spontane Änderungen an der Oberfläche ohne Zutun des Benutzers sind verwirrend und nehmen dem Benutzer das Gefühl der Kontrolle über die Webanwendung. Außerdem enstehen hier natürlich wieder ganz neue Fragen an die Rolle der Browser-History in Bezug auf serverseitige Änderungen der Oberfläche. Diese Probleme gelten im Übrigen genauso für Server-Events mit Polling.

Comet sollte also nur mit Bedacht angewendet werden und dort vermieden werden, wo es alternative Lösungswege gibt. In einigen Anwendungsfällen ist man jedoch auf diese Technik angewiesen. Der Entwickler muss dann dafür sorgen, dass dem Benutzer immer klar ist, was gerade warum passiert. Oft ergibt sich das jedoch schon aus dem Anwendungskontext: Integriert man eine Chat-Funktion in eine Webanwendung, wird es den Benutzer nicht verwirren, wenn ohne sein Zutun Nachrichten in dem Chat-Fenster auftauchen. Handelt es sich um eine Überwachungsanwendung, erwartet der Benutzer ebenso plötzliche Änderungen – natürlich sollten diese trotzdem entsprechend hervorgehoben werden. Ein großes Anwendungsgebiet für Comet sind auch kollaborative Webanwendungen, in denen mehrere Benutzer gemeinsam mit einem Objekt interagieren. In diesem Anwendungsfall ist es durchaus eine große Herausforderung, Events, die von anderen Benutzern kommen, intuitiv verstehbar auf der Oberfläche anzuzeigen.

Comet ermöglicht also, richtig angewendet, einige neue Anwendungsgebiete und -funktionen. In Abbildung 2.5 wird eine kollaborative Webanwendung schematisch dargestellt. Diese Mehrbenutzersicht auf eine Webanwendung ist bisher noch eher ungewöhnlich, da eine laufende Anwendungsinstanz sich eigentlich immer nur auf einen Benutzer bezieht. Durch Comet ist es nun möglich, dass mehrere Benutzer in *einer* Anwendungsinstanz arbeiten und darin auch gegenseitig sichtbar sind und miteinander direkt kommunizieren können (dargestellt durch die roten Pfeile). Natürlich stimmt das nur auf der konzeptionellen Ebene – technisch gesehen können die einzelnen Benutzer nur mit dem Server kommunizieren.

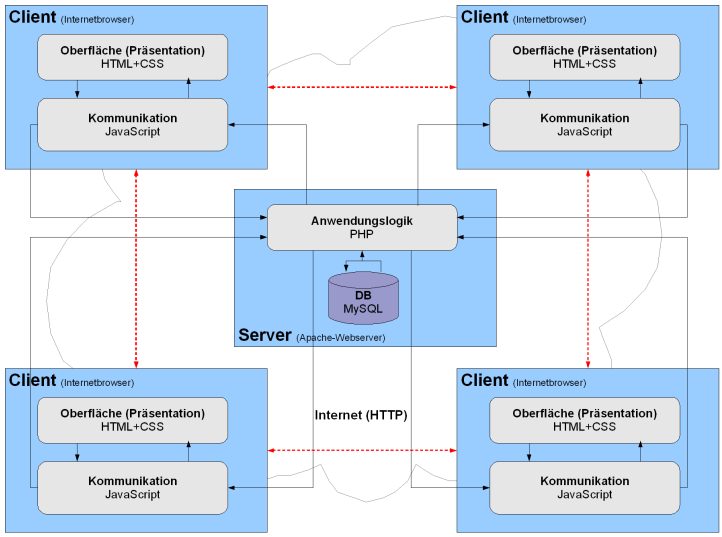


Abbildung 2.5: Modell einer kollaborativen Webanwendung mit Comet

Inwieweit dieses Modell mit der REST-Architektur (siehe 2.3.2) in Konflikt gerät, hängt wieder von der Anwendung ab. Generell kann man sagen, dass die Anwendung desto skalierbarer wird, je mehr Logik auf die Clients ausgelagert wird. Allerdings kann die *Selbst*verwaltung von mehreren Benutzern eine komplexe Angelegenheit werden – je nach Anwendung wird der Server sich dabei also auch mehr oder weniger an der Verwaltung des Anwendungsstatus beteiligen müssen und ist demnach nicht mehr REST-konform.

Comet-Anwendungen im Allgemeinen und kollaborative Anwendungen im Besonderen bedeuten für den Server also große Ressourcen-Anforderungen.

2.5 RichGUIs

Ein von Benutzern am meisten mit Web 2.0 assoziierter Aspekt ist sicherlich die komfortable,

funtktionsreiche und interaktive, aber auch attraktive Oberfläche der Webangebote. Oft wird das in dem Begriff RichGUI (Rich Graphical User Interface) umschrieben. RichGUI ist jedoch nicht äquivalent zu Usability, d.h. Interaktivität, Funktionalität und ein angenehmes Erscheinungsbild garantieren noch keine gute Benutzbarkeit eines Webangebots. Im Sinne von Web 2.0 soll jedoch durch die sinnvolle Ausnutzung des *Möglichen* die Benutzbarkeit verbessert werden. Das bedeutet im Einzelnen:

- Durch geschickte Verwendung von AJAX wird eine flüssige Bedienung der Oberfläche ermöglicht.
- Die Oberfläche wird vollständig von JavaScript kontrolliert. Das heißt, dass Anpassungen der Oberfläche nie durch die Generierung von Markup auf dem Server, sondern nur durch DOM-Manipulationen auf dem Client realisiert werden. Bis zu einem gewissen Grad wird dies schon länger mit DHTML praktiziert, aber erst im Zusammenspiel mit AJAX sind dadurch beliebig funktionale Oberfläche möglich. Die Seite im Browser ist also nicht mehr ein statisches Dokument, sondern eine dynamische Oberfläche.
- Durch Ausnutzung der grafischen Fähigkeiten von aktuellen Browsern und heutigen Bandbreiten wird das Erscheinungsbild der Oberfläche verbessert (die Wichtigkeit eines guten Designers bleibt davon natürlich unberührt).

Die Probleme, die bei RichGUIs auftreten, wurden schon unter 2.3 im Kontext von AJAX behandelt. Im Optimalfall fungiert sie als eine von mehreren Zugriffsmöglichkeiten auf den Service eines Webangebots. Durch die Trennung von Daten und Präsentation ist z.B. das Anbieten eines maschinenlesbaren RSS-Feeds kein großes Problem. Bei Webangeboten, die nicht primär einen funktionalen Charakter haben, werden RichGUIs momentan eher in einer sehr moderaten Ausführung eingesetzt, d.h. so dass die Benutzung ohne JavaScript möglich ist. Hier wird wahrscheinlich die Kosten(Sicherstellung der Accessibility)-Nutzen(eine dynamische Oberfläche kommt erst bei einem funktionalen Webangebot richtig zur Geltung) Rechnung nicht aufgehen.

2.6 Web Feeds und Services

Die Trennung von Daten und Präsentation ist ein Schlüsselkonzept von Web 2.0. Die Daten sind erstens getrennt von der Präsentation gespeichert (das ist eine schon lange etablierte Praxis) und werden zweitens auch getrennt von der Präsentation übertragen (nur mit AJAX möglich, siehe 2.3.3). Das bedeutet, dass ein (Daten-)Objekt auch nicht an eine Seite (das ist in diesem Fall die Präsentation des Objekts) gebunden ist, sondern im Prinzip entkoppelt von jeder Bindung für verschiedene Zwecke verwendet werden kann. Viele Webangebote stellen Daten in der Konsequenz auch unabhängig von ihrer Präsentation zur Verfügung – natürlich in einer strukturierten, maschinenlesbaren Form. Hierbei gibt es zwei leicht verschiedene Ansätze:

Am bekanntesten sind die sogenannten *Web Feeds*. Das Format der Wahl ist hier XML, bzw. Derivate davon, wie RSS oder Atom. Web Feeds werden bei Webangeboten eingesetzt, die regelmäßig eine bestimmte Art von Inhalt veröffentlichen, wie z.B. Weblogs, Nachrichtenseiten oder auch Podcasts. In der Regel enthält ein Web Feed die neuesten Inhaltsobjekte, oder zumindest eine Zusammenfassung davon (bzw. eine URL bei nicht textuellen Inhalten) und kann durch externe Programme ausgewertet werden. Diese Programme sind entweder sogenannte Newsreader, die die Web Feeds von mehreren Webangeboten sammeln, oder aber andere Webangebote, die die Inhalte weiterverwenden wollen. Solche Web Feeds können auch für Screenreader oder ähnliche Programme eine große Hilfe sein, da sie die Inhalte gleich in der richtigen Struktur anliefern. Der Zugriff auf einen Web Feed erfolgt über eine bestimmte URL, die über HTTP angefragt wird.

Der zweite Ansatz geht noch ein Stück weiter - es handelt es sich dabei um *Web Services*. Ein

Web Service hat nicht nur eine statisch definierte Funktion (also z.B. die aktuellsten Inhalte eines Webangebots zu liefern), sondern bietet eine Reihe von Funktionen, die auch noch mit Parametern genauer bestimmt werden können. Oft ist ein Web Service dabei an ein bestehendes Webangebot gekoppelt. Ein Web Service kommuniziert meistens in XML, d.h. sowohl Anfragen also auch Antworten werden in XML ausgetauscht (beliebt ist auch das JavaScript-nahe Format JSON). Beispiele für Webangebote mit Web Service sind z.B. Google, Amazon, YouTube und Flickr. Dort kann man jeweils über eine XML-Schnittstelle bestimmte Funktionen des Webangebots abfragen, so z.B. bei Google eine Suchanfrage durchführen, oder bei YouTube Filme zu einem bestimmten Stichwort finden. Die Antwort kann dann von einem Programm (also z.B. auch von einem anderen Webangebot) ausgewertet und weiterverarbeitet werden. Auf diese Weise können auch sogenannte Mashups erstellt werden, die die Funktionen von verschiedenen Web Services in einem Webangebot zusammenfassen.

Im Kontext von Web 2.0 ist das grundlegende Prinzip bei alledem, dass ein Webangebot - unabhängig von seiner Browseroberfläche - Daten maschinenlesbar zur Verfügung stellt. Dadurch können Webangebote gegenseitig voneinander profitieren.

2.7 Benutzerzentriertes Web

Symptomatisch für Web 2.0 - Webangebote ist die starke Einbeziehung der Benutzer. Normalerweise gibt es mindestens die Möglichkeit, Inhaltsobjekte zu kommentieren, oft kann man diese auch anhand eines bestimmten Systems (meistens 0 bis 5 Sterne) bewerten. Es findet also eine qualitative Einordnung von Inhalten durch den Benutzer statt, sowohl in statistisch auswertbarer, als auch in sprachlich formulierter Form.

Eine weitere Methode ist das *Tagging,* bei dem Inhaltsobjekte mit Stichwörtern (*Tags*) versehen werden können. Dadurch entsteht eine Klassifizierung von Inhalten, die vom Benutzer ausgeht - auch *Folksonomy* (zusammengesetzt aus *Folk* und *Taxonomy*) genannt. Oft ist das Tagging jedoch nur für den möglich, der ein Inhaltsobjekt erstellt – hier gibt es jedoch auch Ausnahmen, z.B. Google Video.

Anhand dieser von den Benutzern generierten Daten wird nun, je nach Webangebot mehr oder weniger automatisch, die Navigation aufgebaut. Die Navigationsstruktur ist also nicht statisch, sondern dynamisch. Durch das Tagging ist eine beliebige und mehrdimensionale Klassifizierung von Inhalten möglich und durch die verschiedenen Bewertungsmöglichkeiten eine Sortierung nach der von den Benutzern wahrgenommenen Qualität. In der Regel werden hier auch weitere statistische Daten, wie Anzahl von Aufrufen, oder Anzahl von Verweisen auf ein Inhaltsobjekt, mit einbezogen.

Am Beispiel von YouTube lässt sich die Funktionsweise einer dynamischen benutzerorientierten Navigation gut erkennen (Abbildung 2.6). „Top Rated“ und „Top Favorites“ weisen auf eine direkt von den Benutzern ausgehende Qualitätsäußerung hin (Stichwort „Top“). „Most Viewed“, „Most Discussed“ und „Most Linked“ weisen auf indirekt vom Benutzer ausgehende quantitative Merkmale hin (Stichwort „Most“), die zwar nicht zwingend Qualität bedeuten, aber auf eine allgemeine Relevanz oder Wichtigkeit hinweisen (*„wenn so viele es anschauen, oder darüber reden, muss es wichtig oder interessant sein.“*). YouTube mischt diese dynamischen Elemente außerdem mit statischen, wie z.B. „Recently Featured“ (redaktionell ausgewählte Videos) oder „Categories“ (vordefinierte Kategorien - zusätzlich zu dynamischem Tagging).

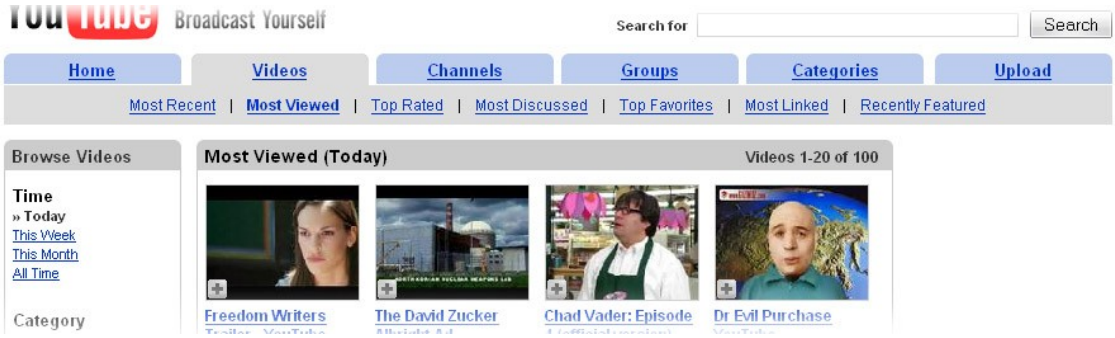


Abbildung 2.6: Dynamische Navigation von YouTube, Quelle: [YOUTUBE2006]

Eine weitere wichtige dynamische Navigationsart in diesem Kontext sind die sogenannten *tag clouds* (Abbildung 2.7). Hierbei werden die vom Benutzer eingegebenen Tags alphabetisch geordnet hintereinander gezeigt. Entscheidend dabei ist jedoch, dass häufiger verwendete Tags größer als weniger verwendete Tags angezeigt werden. Abhängig vom Kontext bekommt man so einen schnellen Eindruck über thematischen Schwerpunkte und kann gleich in relevante Themen einsteigen. Beobachtet man den Aufbau der Startseiten von aktuellen Web 2.0 – Webangeboten, scheint es jedoch, dass die zentralere Methode zur Tag-Navigation die Stichworteingabe in ein Textfeld ist. Auf diese Weise kann zum einen nach mehreren Tags gleichzeitig gesucht werden, zum anderen können bei der Suche sonstige textuell vorhandenen Informationen (wie Objektbeschreibungen, Titel oder volle Texte) berücksichtigt werden. Die Stärke von tag clouds ist dagegen eindeutig die Visualisierung von thematischen Schwerpunkten.

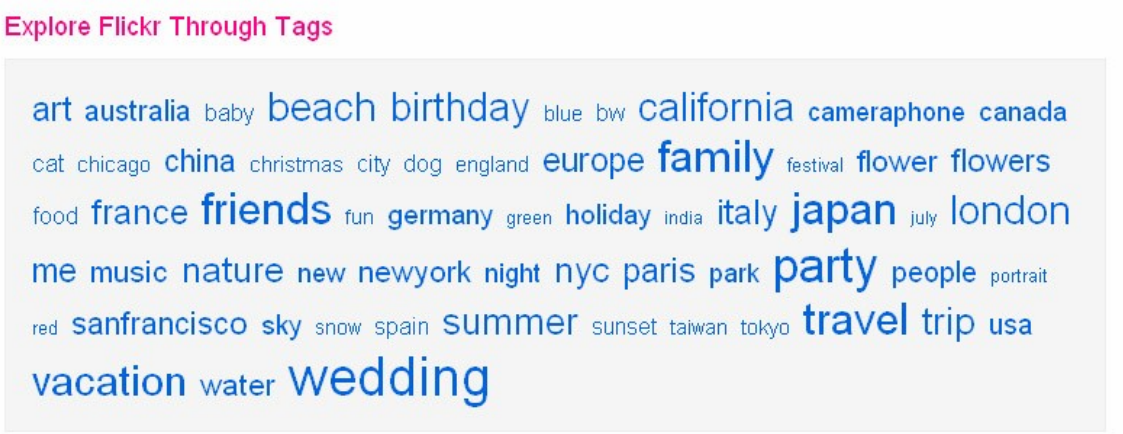


Abbildung 2.7: Tag cloud von Flickr, Quelle: [FLICKR2006]

Typisch für Web 2.0 Webangebote ist außerdem, dass auch die Inhalte selbst ganz oder teilweise von den Benutzern generiert werden. Die meisten der momentan erfolgreichsten Web 2.0 Webangebote (im Sinne von Popularität) sind nach diesem Prinzip aufgebaut: YouTube, Google Video, Flickr, Wikipedia, Digg, Del.icio.us, Technorati und weitere. Der Anbieter stellt hier also „nur“ ein Gerüst zur Verfügung und der Rest läuft quasi automatisch. Einen näheren Blick wert ist dabei das Webangebot Technorati. Dort wird der Inhalt von sämtlichen Blogs im Sinne eines Mashups auf einer Plattform verfügbar bemacht. Dabei nutzt Technorati nicht nur die Folksonomy der jeweiligen Blogs aus, sondern bezieht auch benutzergenerierte Bewertungen und statistische Daten jeder Art mit ein. Technorati ist im Endeffekt also eine dynamische Navigation über die gesamte Blogosphäre hinweg und zeigt damit auf eindrückliche Weise die Möglichkeiten von Web 2.0. So reicht z.B. ein Aufruf der Startseite, um einen Eindruck davon zu bekommen, worüber im Internet gerade hauptsächlich gesprochen wird.

2.8 Das Web als Anwendungsplattform

Im Zuge von Web 2.0 hat sich das Web immer mehr als mögliche Anwendungsplattform qualifiziert. Im Verlauf dieses Kapitels wurden verschiedene Gründe dafür gezeigt, wie z.B. die asynchronen Kommunikationsmöglichkeiten oder RichGUIs. Ein weiterer Aspekt ist die immer größere Verfügbarkeit von Breitband-Internetverbindungen, oft gekoppelt mit einem Flatrate-Tarif. Dazu kommt, dass Internetzugänge auch abseits von zu Hause fast überall verfügbar sind.

Die technischen und infrastrukturellen Gegebenheiten ermöglichen also heute, auch komplexere, bis jetzt nur auf Desktop laufende Anwendungen als Webanwendung zu realisieren. Die technische Machbarkeit allein sagt jedoch nichts darüber aus, ob es wirklich sinnvoll ist, das Web als Anwendungsplattform zu „missbrauchen". Webanwendungen bringen jedoch in der Tat einige Vorteile mit sich, die im Folgenden herausgestellt werden sollen.

Beobachtet man einige schon länger aktive Webanwendungen, so fällt auf, dass diese den Beinamen „Beta" niemals aufgegeben haben. Das weist auf eine wichtige Charakteristik von Webanwendungen hin: Es gibt keinen „Release" einer Anwendung mehr, sondern nur noch den Zustand „Perpetual Beta" (immerwährende Beta). Es wird also praktisch *live* entwickelt – sobald irgendeine sinnvolle Nutzung möglich ist, wird die Anwendung öffentlich zugänglich gemacht und fortan daran weiterentwickelt. Dieses Vorgehen ist deswegen überhaupt möglich, da es für Webanwendungen keinen Rollout wie in der „Offline-Welt" gibt, also das ausliefern und installieren einer Software beim Kunden. Eine Webanwendung bedarf keiner Auslieferung und keiner Installierung – sie ist unmittelbar verfügbar, sobald der Betreiber sie auf einem öffentlichen Webserver installiert hat. Der aufwendige und kostspielige Prozess des Rollout entfällt also völlig. Ein weiterer mit Releases und Rollout verknüpfter Aspekt sind Patches und Bugfixes – also Update und Fehlerbereinigung der Software. Eine Webanwendung bedarf auch hier keines Rollouts – Veränderungen an der Anwendung sind sofort, ohne Zutun von irgendeiner Seite bei allen Benutzern verfügbar. Dies ist zum einen komfortabel für den Benutzer, weil er immer automatisch auf der aktuellsten Version arbeitet. Zum Anderen bedeutet es aber auch einen großen Sicherheitsgewinn, da Sicherheitslücken sofort geschlossen werden können und fortan kein Problem mehr darstellen, da alle immer mit der aktuellen Anwendung arbeiten. Schließlich ist der auch mit dem Rollout zusammenhängende Aspekt der Lizenzierung (oder negativ ausgedrückt: der Raubkopie) für Webanwendungen kein komplexes Problem mehr. Da die Software nicht zum Benutzer ausgeliefert wird, hat der Anbieter die volle Kontrolle über die Anwendung. Für Anwendungen im Web fällt also ein großer Bereich des Software-Engineering einfach weg. Das bedeutet sowohl für den Betreiber als auch für den Benutzer Kostenersparnis und Komfortgewinn. Wie auch momentan im Web zu beobachten ist, gibt das den Betreibern auch wesentlich mehr Flexibilität, um Ideen und Konzepte auszuprobieren – in der Zeit, wo für eine Desktop-Anwendung aufwendige Studien und Testläufe durchgeführt werden, ist eine Webanwendung schon im Live-Betrieb und hat, wenn der Versuch erfolgreich war, schon viele registrierte Kunden. Kommt ein Konzept nicht an (wohlgemerkt im Live-Betrieb, nicht etwa bloß in einer Studie), kann man in einem noch frühen Stadium reagieren und Anpassungen vornehmen. Auf jeden Entwicklungsschritt erfolgt ein automatisches Feedback durch die Benutzer, die gleichzeitig wertvolle Tester wie auch wertvolle Kunden sind. Dieses Feedback erfolgt in Form von statistischen Zahlen wie Zugriffsdaten, Registrierungen, Abmeldungen, etc., aber auch in Form von direktem Feedback und Feature-Requests. Dem Betreiber kann es nur Recht sein, wenn die Benutzer sagen, was sie sich wünschen, und die Benutzer bekommen – zu Recht – das Gefühl, dass sie an der Anwendung mitwirken. Es wird *ihre* Anwendung, die (endlich mal) genau das leistet, was man sich wünscht. Google z.B. beherrscht dieses Prinzip bis zur Perfektion. Alle Webanwendungen befinden sich im Perpetual Beta Status und alle Tage kommen nützliche Features zu den Anwendungen hinzu. Dabei werden konkrete Feature-Requests von den Benutzern berücksichtigt.

Ein Punkt, der darüberhinaus viel Gewicht für die Anwendungsentwicklung hat, ist die Be-

triebssystemunabhängigkeit von Webanwendungen. Viel wurde und wird versucht, um dies im Desktop-Bereich zu erreichen, doch das Web bietet diesen Vorteil „out-of-the-box“. Es soll hier natürlich nicht verschwiegen werden, dass es auch im Web ein ähnliches Problem gibt – nur heißt es dort *Browserkompatibilität*. Es ist nicht trivial, eine komplexe Anwendung auf allen Browsern lauffähig zu machen. Durch die Verwendung von Frameworks (vgl. 3.5.1) kann dieses Problem jedoch deutlich verkleinert werden – jedenfalls ist es nicht vergleichbar mit dem Aufwand, der für die betriebssystemspezifische Programmierung einer Anwendung aufgewendet werden muss. Eine gut programmierte Webanwendung *läuft* auf allen Betriebssystemen, für die es einen aktuellen Browser gibt (vgl. [PAULSON2005]).

Ein weiteres großes Plus für Webanwendungen ist ihre Benutzer- und Dienstmobilität. Diese aus dem Mobilfunkbereich bekannten Begriffe sagen aus, dass ein Benutzer unabhängig von Ort und Gerät auf einen Dienst (hier: Webanwendung) zugreifen kann und dabei auch im Laufe seiner Benutzung immer wieder Ort und Gerät wechseln kann. Der Dienst ist also in keiner Weise an einen Ort oder an ein Gerät gebunden. Während bei Desktop-Anwendungen Ersteres durch die Verwendung eines Laptops oder PDAs leicht erreicht werden kann, ist Letzteres nur mit großem Aufwand und auch nur mit speziell angepassten Geräten möglich. Webanwendungen hingegen lassen sich überall benutzen, wo ein Webbrowser mit Internetanschluss vorhanden ist. Je nach Webanwendung kann dies zwar auf aktuelle Browser eingeschränkt sein, was den Aspekt der Dienstmobilität jedoch nicht berührt, da aktuelle Browser überall auf der Welt in Reichweite sind. In vielen Fällen liegt auch eine Datenmobilität vor, da Daten auf dem Webserver gespeichert werden und somit auch unabhängig vom Gerät verfügbar sind. Das wirft natürlich Fragen über Datensicherheit und -schutz auf, was aber an dieser Stelle nicht vertieft werden soll. Jedenfalls muss in diesem Bereich - je nach Anwendung - einiges an Extra-Aufwand gegenüber Desktop-Anwendungen in Kauf genommen werden, oder aber es muss auf Daten- und damit wahrscheinlich auch auf Dienstmobilität verzichtet werden. Kann die Sicherheit der Daten jedoch zufriedenstellend in den Griff bekommen werden, so bedeutet dies die Lösung eines weiteren komplexen Problems der Desktop-Welt, nämlich die Datensynchronisation. Datenmobilität ist heutzutage ein großes Thema, da der durchschnittliche Benutzer mit mehr als einem Gerät gleichzeitig arbeitet: Ein Rechner im Büro, ein Rechner zu Hause, ein PDA und ein Handy gehören zur Standardausstattung. Vorausgesetzt, ein Internetzugang ist für alle Geräte verfügbar (und bezahlbar) und es wird eine Webanwendung eingesetzt, die auf den Browsern der Geräte läuft, ist Synchronisation nicht mehr nötig, da alle Daten zentral gespeichert werden.

Wenn nun eine echte Benutzer-, Dienst- und Datenmobilität vorliegt, so wird das Endgerät, mit dem der Benutzer interagiert, *wieder* zu einer Art „Thin Client“. Die Anwendung läuft auf einem leistungsstarken Server und der Benutzer greift mit einem einfachen Terminal (ohne besondere eigene Funktionalität) darauf zu. In der Zeit vor dem Personal Computer war der Grund für diese Architektur, dass die Endgeräte einfach nicht leistungsfähig genug waren, heute sind es, wie oben erläutert, andere Gründe. Natürlich gibt es (noch?) viele Anwendungen, die für das Web ungeeignet sind, außerdem bedürfen auch Webanwendungen z.T. leistungsstarker Endgeräte, daher wird es weiterhin komplexer Endgeräte bedürfen. Fakt ist jedoch, dass auch heute schon ein einfacher Rechner mit Browser und Internetzugang eine sehr große Funktionalität bereitstellt – vorausgesetzt der Benutzer nutzt entsprechende Webanwendungen.

Da Webanwendungen prinzipbedingt auf einem zentralen Server laufen, entsteht ein weiterer Vorteil: alle Benutzer einer Webanwendung sind von Haus aus über den Server vernetzt. Die Architektur einer Webanwendung begünstigt also kollaborative Features. Die gemeinsame Arbeit an den gleichen Daten ist zwar auch im Web keine triviale Angelegenheit. Doch gestaltet sich die Realisierung wesentlich unkomplizierter, da die Infrastruktur für die Vernetzung und Datenhaltung schon vorliegt. Alle greifen auf die gleiche Datenbasis zu und können – wie oben gezeigt – direkt miteinander kommunizieren.

Im Spannungsfeld der oben beschriebenen Punkte existieren zwei Trends: *WebOS* und *Webified*

Desktop. Während ersterer Trend auf die Auflösung des heutigen Desktops hinzielt, hat letzterer eher zum Ziel den Desktop durch einzelne Web-Features aufzuwerten. Ein Beispiel ist z.B. die Bemühung von Microsoft, seine Office-Suite konsequent um kollaborative Features zu erweitern. Dies ist sicherlich wünschenswert, doch damit fehlen einer Desktop-Anwendungen immer noch die meisten der oben genannten Vorteile von Webanwendungen. Um diese zu erreichen, müsste jede Desktop-Anwendung ihre Daten im Web speichern und auch eine alternative Web-Version der Anwendung bieten. In diesem Bereich gibt es jedoch auch schon Ansätze von Microsoft. Momentan sind allerdings viele Bemühungen in Richtung WebOS zu beobachten. Ernstzunehmende Anbieter, die vollwertige webbasierte Anwendungs-Suites anbieten, sind hier unter Anderem Zimbra, Zoho, 37Signals, SynapseLife und natürlich Google (jeweils unter *.com erreichbar). Google ist hier mit seiner riesigen Benutzerbasis die Referenz, wenngleich andere Anbieter oft die ausgefeilteren Anwendungen haben.

Google fährt in diesem Bereich eine interessante Doppelstrategie. Das Produkt Google Desktop bietet dem Benutzer die Möglichkeit, Daten von Webanwendungen, sowie lokale Daten gleichzeitig in einem Index zu durchsuchen und einzusehen. Außerdem bietet das Programm an, lokale Daten ins Web zu übertragen, damit sie von anderen Rechnern aus zugreifbar sind. Letztlich geht die Strategie von Google jedoch klar in Richtung WebOS. So bietet das Unternehmen z.B. unter dem Slogan „Google Apps for Your Domain“ anderen Unternehmen und Universitäten an, seine Webanwendungen auf der Domain des Kunden laufen zu lassen (https://www.google.-com/a/). Für den Privatkunden stellt Google mittlerweile eine ganze Reihe von – weitgehend kostenlosen - Webanwendungen bereit: Den beliebten Mailclient Gmail (gmail.google.com), einen Timeplaner (calendar.google.com), Textverarbeitung und Tabellenkalkulation (docs.google.com), einen Web Feed Reader (reader.google.com), eine Bildverwaltung (picasaweb.google.com, gekoppelt mit einer Desktop-Applikation), einen Instant Messenger (integriert in Gmail) und weitere. Da, wo es Sinn macht, unterstützen alle Anwendungen kollaborative Features. Die anderen Anbieter bieten ein vergleichbares Portfolio.

Es gibt auch schon Bestrebungen, ein vollwertiges Web-Betriebssystem zu erschaffen. Dies würde dann dem „Thin Client“-Gedanken noch stärker entsprechen. Hier gibt es z.B. das Open Source Projekt EyeOS (www.eyeos.net), das schon sehr vielversprechend aussieht.

3 Die hohe Kunst des Web-Engineering

> "Web-based system development is like gardening - like a garden, Web-based system will continue to evolve, change and grow. However, a good initial infrastructure is required to allow the growth to occur in a controlled, but flexible and consistent manner, and to foster creativity, refinement and change." [MURUGESAN1999, Seite 4]

Das Web ist ein dezentrales Netz, in dem prinzipiell jeder ein Webangebot veröffentlichen kann. Zugangsbeschränkungen oder -hindernisse sind sind zwar existent, aber nicht Thema dieser Arbeit. Jedenfalls ist es in zivilisierten Ländern auch für den „normalen" (also nicht-professionellen) Benutzer (im Folgenden *Endbenutzer* genannt) leicht möglich, Zugriff auf einen „Webspace" zu bekommen, um Inhalte im Internet verfügbar zu machen. An diesem Punkt setzt diese Arbeit an. Es ist natürlich durchaus attraktiv, ein eigenes, weltweit verfügbares Webangebot zu haben – dementsprechend gibt es verschiedene Werkzeuge, um dem Endbenutzer bei der Realisierung dieses Ziels zu helfen. Technikaffinere Leute kommen auch ohne solche Hilfestellungen aus und bringen sich zur Realisierung eines Webangebots nötige Minimalkenntnisse selbst bei. Natürlich gibt es dann auch die nicht näher bestimmbaren „Profis", die sich irgendwie alle gut auskennen, aber alle unterschiedliche Wege haben, ein Webangebot zu realisieren. Das spannt eine Skala auf, zwischen dem absolut unbedarften Endbenutzer, der einen Homepage-Baukasten mit Inhalten befüllt, und dem versierten Profi, der mit ausgewachsenen Entwicklungsumgebungen arbeitet und sich etablierter Engineering-Methoden bedient (selten!). Das besondere am Web ist, dass alle auf der gleichen Spielwiese agieren – sie alle sind Betreiber von einem gemeinsamen Netzwerk. Murugesan et al. stellten schon 1999 fest:

> „... the type of individuals who build/develop Web-based systems are vastly varied in their background, skills, knowledge and system understanding" [MURUGESAN1999, Seite 5].

Dieser Sachverhalt ist auf der einen Seite eine wichtige Grundlage des Web, da nur durch den freien und leichten Zugang zur Veröffentlichung von Webangeboten Vielfalt und Meinungsfreiheit möglich ist. Auf der anderen Seite bedingt das jedoch auch einen niedrigen Qualitätsstandard, da es keine Standards gibt, an die sich jeder halten *muss* (Einschränkung: für Webangebote von öffentlichen Einrichtungen gibt es z.B. in Deutschland und den USA Pflichtstandards). Sogar im professionellen Bereich sind bestimmte Standards für die Entwicklung von Webangeboten keine Selbstverständlichkeit (siehe 3.7.2) – bei den Endbenutzern sieht es folglich in diesem Bereich erst recht düster aus. Und hier kommt Web-Engineering ins Spiel – es soll die Entwicklung von qualitativ hochwertigen Webangeboten sicherstellen. Im Folgenden wird die Disziplin des Web-Engineering näher beleuchtet; im weiteren Verlauf der Arbeit wird es im Besonderen darum gehen, wie Web-Engineering auch für den Endanwender greifbar und umsetzbar sein könnte.

3.1 Definition

Grundsätzlich lehnt sich Web-Engineering vom Begriff und Inhalt her an die verwandte Disziplin des Software-Engineering an. In 2.8 wurde jedoch schon gezeigt, dass für Anwendungen im Web in einigen Punkten grundlegend andere Prinzipien gelten, als für deren Desktop-Pendants. Wenn nun das Web zudem immer mehr zur Anwendungsplattform wird und Weban-

wendungen folglich immer komplexer werden und Einzug in das tägliche Leben nehmen, so wird deutlich, dass für das Web hier eine eigene Disziplin gebraucht wird.

Ginige et al. stellen außerdem fest:

> "Developing Web-based systems is significantly different from traditional software development and poses many additional challenges. There are subtle differences in the nature and life cycle of Web-based and software systems and the way in which they're developed and maintained. Web development is a mixture between print publishing and software development, between marketing and computing, between internal communications and external relations, and between art and technology." [GINIGE2001, Seite 16]

Grundsätzlich kann in Bezug auf den Lebenszyklus einer Webanwendung jedoch erstmal von einem iterativen Wasserfallmodell geredet werden, das so auch ähnlich aus dem Software-Engi-

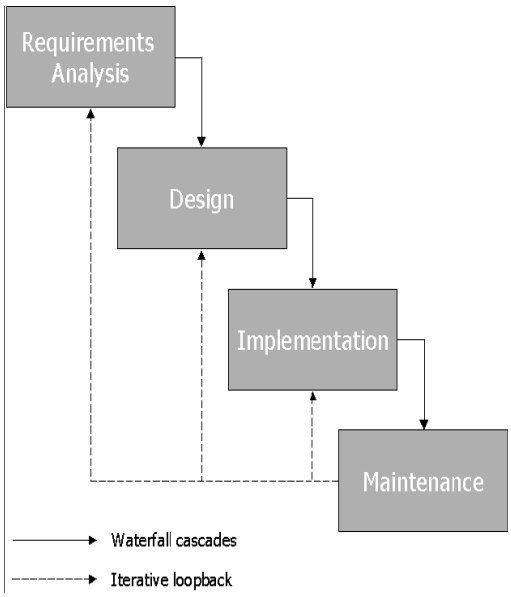


Abbildung 3.1: Life Cycle of a Web Service, Quelle: [KIRDA2001]

neering bekannt ist (vgl. Abbildung 3.1). Die Iterationsfrequenz ist allerdings bei Webanwendungen wesentlich höher. Die Stufen des Wasserfalls werden außerdem nicht streng sequentiell abgearbeitet. Das liegt zum einen an dem „Perpetual Beta"-Status, wie in 2.8 gezeigt wurde. Webanwendungen werden sehr schnell in einen lauffähigen Live-Status gebracht und dabei parallel weiterentwickelt. Dies erinnert an die „Extreme Programming" Methode, bei der ebenso viele kurze Zyklen immer wieder durchlaufen werden und sehr schnell ein lauffähiges Programm angestrebt wird (öfteres Refactoring wird dabei in Kauf genommen). Zum Anderen ist es für Webanwendungen charakteristisch, dass regelmäßig Änderungen am grafischen sowie am logischen Design fällig werden. Kirda et al. schreiben hierzu:

> „Most Web sites today change their appearance at least once a year to stay attractive. Minor changes in the look-and-feel of a site several times a year are very common, and major modifications are not rare. Modifications are motivated by better understanding of user needs based on previously gained feedback, new requirements, optimization strategies and new market directions." [KIRDA2001, Seite 3]

Es stellt sich nun die Frage, ob unter diesen Gegebenheiten feste Strukturen und Methoden im Sinne eines „Engineering“ überhaupt sinnvoll sind. Von einem wissenschaftlichen Standpunkt aus wird diese Frage sicherlich mit „Ja“ beantwortet werden; in der Praxis sieht das jedoch oft ganz anders aus – siehe hierzu Abschnitt 3.7.2. Murugesan et al. schreiben dazu ganz treffend, dass Web-Engineering-Ansätze, um sich nicht selbst ad absurdum zu führen, genug Flexibilität und Kreativität erlauben müssen. Sie müssen außerdem die Vielzahl an Endgeräten, Endbenutzern und speziellen Anforderungen des Web, sowie vor allem die Diversität der Entwickler von Webangeboten in Erfahrung, Können und Wissen berücksichtigen (vgl. [MURUGESAN1999]).

Deshpande et al. beschreiben die Disziplin des Web-Engineering mit folgenden Worten:

> „Web Engineering deals with the process of developing, deploying and maintaining Web applications. The main themes of Web Engineering encompass how to successfully manage the diversity and complexity of Web applications development, and, hence, to avoid potential failures that may have serious implications.“ [DESHPANDE2002, Seite 14]

Es wird hier also betont, dass Web-Engineering den ganzen Lebenszyklus eines Webangebots umfasst und dabei helfen soll, die besonderen Schwierigkeiten in der Webentwicklung zu meistern.

3.2 Ziele

Die Ziele von Web-Engineering werden hier in 3 Dimensionen aufgeteilt:

- **Flexibilität:** Webangebote sind ständigen Veränderungen ausgesetzt. Eine einfache und schnelle Anpassung muss also sichergestellt werden.
- **Accessibility:** Der Zugriff auf das Web erfolgt von verschiedensten Benutzern, mit den verschiedensten Endgeräten. Je nach Anforderung muss eine ausreichende Funktionsweise eines Webangebots unter allen Gegebenheiten sichergestellt werden.
- **Usability:** Webangebote fallen genauso wie Desktop-Programme oder Ticket-Automaten in das Arbeitsfeld der Mensch-Maschine-Interaktion. Folglich ist es ein erstrebenswertes Ziel, die Maschine (das Webangebot) so zu gestalten, dass sie für den Menschen maximal intuitiv benutzbar ist.

3.2.1 Flexibilität

„Web engineering is the 'process of designing for change'“ [KIRDA2001, Seite 3]. Diese Aussage fasst Web-Engineering in Bezug auf Flexibilität gut zusammen. „Änderbarkeit“ wird also zu einem zentralen Designprinzip in der Entwicklung erhoben. Änderbarkeit muss, je nach Anforderung, in verschiedenen Ebenen gewährleistet sein.

In den meisten Fällen wird bei einem Webangebot eine möglichst große Benutzerzahl angestrebt. In anderen Fällen bleibt zwar die Benutzerzahl in einem eingeschränkten Bereich, aber die Datenmenge wird mit der Zeit immer größer. Jedenfalls ist es für ein (erfolgreiches) Webangebot symptomatisch, dass auch die Anforderungen mit der Zeit steigen. Damit ein Webangebot auch bei explodierenden Nutzerzahlen und Datenmengen noch performant läuft und benutzbar ist, muss es *skalierbar* sein, d.h. es muss schnell und einfach an gestiegene Anforderungen anpassbar sein. Dabei ist ein weiterer entscheidender Punkt, dass die technischen Anforderungen möglichst gleichmäßig und nicht etwa exponentiell mit dem Webangebot wachsen. Wird die Skalierbarkeit nicht von Anfang an berücksichtigt, ist - bei entsprechendem

Wachstum - ein Zusammenbruch des Webangebots fast unvermeidbar. Soll das Webangebot dann weiter aufrechterhalten werden, müssen oft tiefe Umstrukturierungen erfolgen. Da man sich dafür jedoch keine Zeit nehmen will oder kann, müssen schnelle Lösungen her. Die Folge ist unsaubere Programmierung, ein instabiles System und unnötig hohe Hardwareanforderungen. Skalierbarkeit ist auch in Bezug auf die Bedienbarkeit eines Webangebots von Bedeutung. Geht der Umgang mit hundert Inhaltsobjekten noch leicht von der Hand, so muss das noch lange nicht für Millionen der Fall sein.

In aller Regel ändert sich der Inhalt eines Webangebots in kurzen Abständen, oft sogar mehrmals täglich. Hinzu kommen, wie oben schon erläutert, gelegentliche Änderungen in Layout und Funktionalität. Entscheidend ist nun hier die möglichst strikte Trennung zwischen Inhalt, Layout und Logik, damit diese jeweils getrennt voneinander verwaltet werden können. Dies lehnt sich an die MVC-Architektur (Model-View-Controller) aus dem Software-Engineering an; durch die hohe Änderungsrate ist diese Architektur für Webanwendungen noch in viel größerem Maße entscheidend, wie sie es sowieso schon für herkömmliche Software ist. Sind etwa im Extremfall Daten, Layout und Teile der Anwendungslogik alle in der gleichen HTML-Datei verknüpft, so wird jeder Änderungsschritt in einem der Bereiche zu einer komplexen Angelegenheit mit potentiellen Nebenwirkungen auf die anderen Bereiche. Liegt hier aber eine klare Trennung vor, so können Änderungen schnell und sicher durchgeführt werden. Darüber hinaus gewinnt man einen weiteren Vorteil: einzelne Komponenten können nun für verschiedene Zwecke wiederverwendet werden. Wird eine Webanwendung z.B. auf einem Smartphone ausgeführt, so soll die Anwendungslogik sicher unverändert bleiben – aber die Art und Präsentation der Inhalte ändert sich. Oder es erfolgt ein Redesign des Webangebots – im Optimalfall muss dafür der Programmierer überhaupt nicht behelligt werden, weil die Layout-Komponente komplett unabhängig von der Programmlogik bearbeitet werden kann.

Was die Verwaltung von Inhalten angeht, finden sich unter dem Oberbegriff „Content Management“ eine Vielzahl von Werkzeugen. Nichtsdestotrotz muss hier darauf geachtet werden, dass das Einfügen und Ändern von Inhalten leicht und schnell erfolgen kann – schließlich sind die Inhalte am häufigsten von Änderungen betroffen. Beachtet werden sollte hier insbesondere, dass Inhalte oft von technisch nicht versierten Benutzern verwaltet werden. Eine strikte Trennung von Layout und Inhalt liegt vom technischen Standpunkt aus zwar auf der Hand, ist für einen Benutzer aber oft nicht intuitiv. In der Praxis muss hier oft ein Mittelweg gefunden werden zwischen der Gestaltungsfreiheit von Inhalten und dem Trennungsgebot zwischen Daten und Layout. Dieser Punkt wird im Verlauf dieser Arbeit noch vertieft.

Eine hohe Flexibilität verbunden mit einer hohen Änderungsrate bringt auch einen erhöhten Bedarf für ein umfassendes Versionssystem mit sich. Änderungen sollten zurückverfolgt werden können und getrennt nach Inhalt, Layout und Logik zurückgesetzt werden können. Liegt hier wiederum keine Trennung vor, so bedingen sich verschiedene Änderungen gegenseitig und sind faktisch nicht mehr sauber rückgängig zu machen.

3.2.2 Accessibility

Accessibility ist – gerade im Kontext des aktuellen Web 2.0 - Hypes – ein großes Thema im Web-Engineering. Im Web hat man es mit vielen verschiedenen Endgeräten und Browsern zu tun, anders etwa als z.B. bei Desktop-Anwendungen, die im Allgemeinen für eine bestimmte Plattform entwickelt werden. Es ist daher ein wichtiges Kriterium für ein Webangebot, dass es auf allen (oft utopisch) bzw. möglichst vielen Endgeräten und Browsern benutzbar ist. Als Härtetest gilt hier in der Regel die Benutzbarkeit eines Webangebots auf Screenreadern. Diese werden hauptsächlich dafür eingesetzt, um blinden Benutzern die Benutzung von Webangeboten zu ermöglichen. Inhalte werden dem Benutzer dabei sequentiell „vorgelesen“ - man kann sich vorstellen, dass die Benutzung eines Webangebots auf diese Weise sehr mühsam ist,

außerdem bleiben dem Benutzer so auch nicht-textuelle Inhalte verschlossen. Um eine zufriedenstellende Funktionsweise eines Screenreaders zu gewährleisten, sollte ein Webangebot genügend Meta-Informationen zur Verfügung stellen. So sollten z.B. die Bereiche der Oberfläche entsprechend ihrer Funktion ausgewiesen werden (Navigation, Werbung, Inhalt) und nicht-textuelle Inhalte mit Beschreibungstexten versehen werden.

Ein weiteres oft verlangtes Accessibility-Kriterium ist, dass das Webangebot auch ohne JavaScript lauffähig und benutzbar ist. Das bedeutet gerade im Kontext von Web 2.0 eine große Herausforderung, da sowohl AJAX als auch dynamische Benutzeroberflächen (im Sinne einer RichGUI) ohne JavaScript nicht lauffähig sind. Das stellt zum einen für die oben angesprochenen Screenreader ein Problem dar, aber auch in vielen aktuellen Browsern ist JavaScript deaktiviert (oft aus Sicherheitsgründen), bzw. nicht ausreichend unterstützt (ältere Browser oder mobile Endgeräte). An dieser Stelle sei jedoch angemerkt, dass man zumindest im Bereich von komplexen Webanwendungen nicht zuviel verlangen kann. Die Bedienbarkeit einer komplexen Anwendung über einen Screenreader ist zwar sehr wünschenswert, kann aber aufgrund der eingeschränkten Möglichkeiten nur bedingt realisiert werden, da bestimmte Funktionalitäten überhaupt erst durch die technischen Möglichkeiten aktueller Browser möglich sind (siehe Kapitel 2). Es gibt sicherlich noch viel zu forschen und zu erfinden, um blinden Menschen die Bedienung von komplexeren Anwendungen zu ermöglichen; dies ist jedoch nicht ein Problem von Webanwendungen im Speziellen, sondern betrifft das Feld der Mensch-Maschine-Interaktion im Allgemeinen. Accessibility ist also in besonderem Maße für informationsorientierte Webangebote von Bedeutung.

3.2.3 Usability

Usability ist ein qualitatives Merkmal von Benutzeroberflächen, oder -schnittstellen, dass sich daran misst, wie einfach oder intuitiv eine Oberfläche zu benutzen ist. Wenngleich das natürlich auch subjektive Aspekte enthält, so gibt es doch einige objektiv messbare Merkmale, an denen man Usability festmachen kann. Dort, wo es möglich ist, sollten bekannte Usability-Prinzipien in den Web-Engineering-Prozess integriert werden, um die Entwicklung von benutzbaren Webangeboten zu unterstützen:

> „Engineering is about systematic, disciplined and quantifiable approaches to create usable systems.“ [DESHPANDE2002, Seite 12]

Nielsen untergliedert Usability in fünf Komponenten (vgl. [NIELSEN2003], hier auf Webangebote angewendet):

- **Learnability (Erlernbarkeit)**: Wie leicht/schnell ist die Benutzung eines Webangebots erlernbar?
- **Efficiency (Effizienz)**: Wie schnell können Arbeitsschritte durchgeführt werden?
- **Memorability (Behaltbarkeit)**: Wie leicht ist es nach längerer Pause wieder in die Benutzung hineinzufinden?
- **Errors (Fehler)**: Wie oft werden von Benutzern Fehler begangen? Haben diese Fehler schwere Konsequenzen; können diese rückgängig gemacht werden?
- **Satisfaction (Zufriedenheit)**: Wie angenehm ist die Benutzungserfahrung?

Für ein Webangebot ist hier im Besonderen die Konsistenz zu allgemeinen Standards der Oberflächengestaltung und eine logische und intuitive Navigation von Bedeutung.

3.3 Modellbasierte Ansätze

Wie auch im Software-Engineering gibt es im Web-Engineering eine Reihe modellbasierter Ansätze, die die Entwicklung unterstützen sollen. Allen diesen Ansätzen ist gemein, dass man mit ihnen auf einer hohen Abstraktionsebene arbeitet, ohne sich zunächst um Implementierungsdetails kümmern zu müssen. Die Implementierung erfolgt dabei zum Schluß meist semi-automatisch durch einen Code Generator. Durch die Trennung verschiedener Ebenen eines Webangebots - wie etwa das Daten, Navigation und Präsentation – und durch die Unabhängigkeit von der tatsächlichen Implementierung können viele Prozesse automatisiert und viele Einzelkomponenten wiederverwendet werden. Dadurch soll eine schnellere Entwicklung von Webangeboten ermöglicht werden, die dabei gleichzeitig sicherer, robuster und anpassbarer sind.

Diese wünschenswerten Ziele müssen jedoch auch im Kontext der Probleme von modellbasierten Ansätzen gesehen werden. In der Praxis werden die hier beschriebenen Ansätze jedoch so, wie sie in der Literatur vorgegeben werden, nur selten eingesetzt. Vor Allem semiprofessionelle Entwickler haben Probleme mit der abstrakten Arbeitsweise. Für sie erschließt sich außerdem nicht der Nutzen, insbesondere im Hinblick auf den hohen Einarbeitungs- und Lernaufwand (vgl. 3.7.2). Im professionellen Bereich werden oft proprietäre Ansätze eingesetzt, da die streng formellen Methoden aus der Literatur in der Praxis als unhandlich empfunden werden. Kirda et al. schreiben, dass sich die Notation der gesamten Struktur eines Webangebots in einer UML-ähnlichen Form, wie es bei allen vorgestellten Ansätzen der Fall ist, als nicht skalierbar für sehr komplexe Webangebote erwiesen hat [KIRDA2001]. Dem wird in anderen Veröffentlichungen natürlich widersprochen, wenngleich dabei in der Tat nur Fallstudien vorgewiesen werden, die sich innerhalb einer bestimmten Komplexitätsgrenze bewegen.

Jedenfalls ist der Entwickler auf die Möglichkeiten des Modells eingeschränkt. Manuelle Anpassungen an der Implementierung sind bei komplexen Webangeboten in der Praxis kaum zu vermeiden. Danach lassen sich Änderungen am Modell jedoch nicht mehr einfach so in ein laufendes Webangebot integrieren. Atterer kritisiert außerdem, dass die Möglichkeiten in den Präsentationsmodellen sehr eingeschränkt und umständlich sind, und dass Usability-Aspekte von aktuellen Ansätzen nicht beachtet werden, bzw. nicht in den Modellen ausgedrückt werden können [ATTERER2005].

3.3.1 OOHDM

Gemäß OOHDM (Object-Oriented Hypermedia Design Method) erfolgt die Entwicklung von Webangeboten (genauer: Hypermedia-Anwendungen) in einem vierteiligen Prozess [SCHWABE1998]:

1) **Conceptual Design**: Hier wird das konzeptionelle Modell der Anwendung erstellt. Es beschreibt die in der Anwendung vorkommenden Objekte (Entity-Objekte) und deren Beziehung untereinander. Zudem werden Verhaltens-Objekte definiert, die wichtige Algorithmen und Funktionen der Anwendung repräsentieren.

2) **Navigation Design**: Aufbauend auf das konzeptionelle Modell werden im Navigationsmodell Navigationsobjekte definiert, die als Sichten (Views) auf die Entity-Objekte fungieren, analog zu den aus relationalen Datenbanken bekannten Views.

3) **Abstract Interface Design**: Interface-Objekte dienen schließlich als Zugriffspunkt auf die Navigationsobjekte. Das Interface-Modell definiert dabei, welche Interface-Objekte der Benutzer sieht und wie er damit interagieren kann.

4) **Implementation**: In der Implementierungsphase wird schließlich aus den erstellten Modellen eine Anwendung erzeugt. Dies ist kein fest definierter Teil von OOHDM,

> sondern kann auf verschiedene Weise erfolgen. Insbesondere besteht dabei keine Einschränkung in Bezug auf Programmiersprache oder Zielplattform.

In jedem Schritt wird also ein Modell erstellt oder darauf Bezug genommen. Dabei erfolgen die Schritte sowohl iterativ, als auch inkrementell. Laut Schwabe et al. dient das dem folgenden Zweck:

> „From an architectural point of view, separating concerns among conceptual, navigational and interface objects helps to produce applications that are easier to extend and/or maintain.“ [SCHWABE1998, Seite 28]

Ein wichtiger Grundsatz von OOHDM ist, dass die Designphasen absolut unabhängig von der Implementierung sind. Diese ist im Prinzip auch kein Bestandteil mehr von OOHDM, wenngleich es hier auch Werkzeuge wie z.B. OOHDM-Web gibt, die aus einem OOHDM-Modell ein lauffähiges Webangebot generieren.

Schwabe et al. zeigten außerdem, dass mit OOHDM auch die Wiederverwendung von Objekten des konzeptionellen sowie des Navigationsmodells möglich ist [SCHWABE2001]. So kann die Entwicklung durch die Verwendung von immer wiederkehrenden Strukturen und Objekten vereinfacht und beschleunigt werden.

3.3.2 WebML

WebML (Web Modelling Language) dient zur Beschreibung eines Webangebots auf einer hohen Abstraktionsebene. Alle Konzepte werden dabei in XML oder einer äquivalenten grafischen Notation definiert. In WebML werden dabei folgende Modelle unterschieden (vgl. [CERI2000]):

- **Structural Model**: Ähnlich wie bei dem Conceptual Design bei OOHDM werden hier *Entities*, das sind Container für Datenelemente, und ihre Beziehung untereinander definiert.
- **Composition Model:** Die Darstellung von Entities wird hier in *Units* modelliert. Es gibt verschiedene Typen von Units, die flexiblen Zugriff auf Entities ermöglichen. So können z.B. alle Instanzen einer Entity in einer Übersicht angezeigt werden, oder nur die bestimmten Attribute einer einzigen Instanz. Mehrere Units können in einer *Page* zusammengefasst werden.
- **Navigation Model:** Units und Pages können durch Links miteinander verbunden werden. Dabei wird unterschieden zwischen kontextuellen Links, denen Daten der Quell-Unit mitgegeben werden können, und nicht-kontextuellen Links.
- **Presentation Model:** Mit XSL Stylesheets können hier beliebige Präsentationsregeln definiert werden. So können Seiten z.B. in HTML oder WML gerendert werden.
- **Personalization Model:** Für die Erzeugung von personalisierten Inhalten können hier Regeln definiert werden.

Mit WebML kann ein Webangebot abstrakt spezifiziert werden, unabhängig von der gewünschten Programmiersprache auf dem Server bzw. der Präsentationsart auf dem Client.

Das *Composition Model* sowie das *Navigation Model* bilden zusammen das *Hypertext Model*. So können auf dem selben Datenmodell verschiedene „Hypertexte“, bzw. *site views* definiert werden. Dabei ist eine *site view* ein Graph von untereinander verlinkten *Pages.*

Mit WebML konnten schon komplexe Rich Internet Applications (RIA) [BOZZON2006], sowie kollaborative Webanwendungen [CERI2001] erfolgreich modelliert werden.

Zu WebML existiert ein CASE-Tool names WebRatio [WEBRATIO2006] mit dem Modelle in der UML-ähnlichen grafischen WebML-Notation erzeugt und mittels eines Code Generators automatisch in eine lauffähige Webanwendung (in Java) umgewandelt werden können. Die Code Generation Rules sind in XSL definiert und können beliebig an andere Bedürfnisse angepasst werden.

3.3.3 UWE

UWE (UML-based Web Engineering) ist ein Ansatz, bei dem - wie es der Name schon vermuten lässt - Modelle ausschließlich in UML notiert werden [KOCH2002]. Das hat zum Vorteil, dass vorhandene UML-Werkzeuge verwendet werden können; außerdem ist UML bei vielen Entwicklern schon bekannt und sehr gut dokumentiert. Ähnlich wie bei OOHDM, gibt es bei UWE eine Aufteilung in ein *Conceptual Model,* ein *Navigational Model* und ein *Presentational Model*. Diese Modelle werden von dem *UWEXML preprocessor* in ein XML-Format umgewandelt, aus dem dann wiederum durch den *UWEXML generator* semi-automatisch das Webangebot generiert wird. Das CASE-Tool ArgoUWE [KNAPP2003] ermöglicht die Modellierung in UML, sowie die semi-automatische Code Generierung durch die Integration von UWEXML.

UWE hat, mehr noch als andere Ansätze, den gesamten Web-Engineering-Prozess im Blick. So gibt es hier z.B. auch Aktivitätsdiagramme zur Modellierung von Arbeitsschritten in einem Webangebot, Zustandsdiagramme zur Visualisierung von Webszenarien, oder Deployment-Diagramme, die den Auslieferungsprozess eines Webangebots beschreiben sollen [KOCH2002].

3.4 WYSIWYG Webentwicklungswerkzeuge

Es ist an dieser Stelle nicht möglich, einen vollständigen Überblick über WYSIWYG Webentwicklungswerkzeuge zu geben, da es zu viele verschiedene Anbieter in diesem Bereich gibt. Vielmehr soll an dieser Stelle an dem Beispiel von Adobe Dreamweaver [ADOBE2006b] (ein von vielen Entwicklern benutztes Programm, vgl. [RODE2005]), das Prinzip dieser Werkzeuge im Allgemeinen untersucht werden.

Der ursprüngliche Hauptzweck von Dreamweaver, die Bearbeitung von Webseiten im WYSIWYG-Modus, ist auch heute noch der zentrale Bereich des Programms. Webseiten können mit Dreamweaver ähnlich flexibel bearbeitet werden, wie es die meisten Benutzer etwa von aktuellen Textverarbeitungsprogrammen wie Microsoft Word kennen. Das ist ein entscheidender Faktor dieser Gruppe von Werkzeugen, da die Umsetzung des gewünschten Layouts eines der Hauptprobleme für viele Entwickler von Webangeboten ist (vgl. 3.7). Allerdings werden dadurch einige Probleme bedingt. Der WYSIWYG-Ansatz verleitet zur Vermischung von Inhalt, Layout und Navigation. Dabei bearbeitet man konzeptbedingt *Seiten* und nicht Inhalte. In aktuellen Dreamweaver-Versionen gibt es einige Bestrebungen, um diesen Problemen entgegenzuwirken, z.B. durch Verwendung von zentralen CSS-Sheets, Templates und seitenübergreifenden Navigationselementen. Dies ändert jedoch nichts an dem ursächlichen Grundkonzept, dass ganze *Seiten* (in der Regel HTML-Dateien) im WYSIWYG-Modus bearbeitet werden. Seitenübergreifende Operationen bleiben umständlich und fehleranfällig.

Nichtsdestotrotz bietet Dreamweaver einige lobenswerte Web-Engineering-Features. So überwacht das Programm auf Wunsch automatisch verschiedene Accessibility-Richtlinien und überprüft auf validen Code und korrekte Links. Bei potentiellen Browserkompatibilitätsproblemen schlägt das Programm ebenso Alarm. In den neueren Versionen will sich Dreamweaver auch als Entwicklungswerkzeug für Web*anwendungen* verstanden wissen. Damit verbunden sind Funktionen zur Anbindung von Datenbanken, sowie Unterstützung von diversen Skriptsprachen wie z.B. PHP und ASP.

3.5 Frameworks und Toolkits

3.5.1 JavaScript Frameworks

Es gibt unzählige JavaScript Frameworks, die an dieser Stelle nicht umfassend behandelt werden können. Einige verstehen sich als Framework, andere als Toolkit, wieder andere als Library – im Prinzip erfüllen sie aber alle ähnliche Zwecke:

- Schaffung einer Abstraktionsschicht, auf der Java-Script-Code browserunabhängig läuft.
- Erweiterung der Fuktionalität der JavaScript-Implementierung aktueller Browser:
  - Vereinfachung/Kapselung von oft gebrauchtem Programmcode, um kürzeren und übersichtlicheren Code zu erhalten.
  - komfortablere und erweiterte Möglichkeiten zur DOM-Manipulation, insbesondere zur Steuerung der Oberfläche
  - Bereitstellung und Wiederverwendung von Oberflächenkomponenten (*Widgets*).
  - Bereitstellung von für Webanwendungen relevanten Funktionalitäten wie ein flexibleres Event-System und eine unkomplizierte Datenkommunikation

Die Benutzung eines dieser Frameworks ist mittlerweile Standard in der Webentwicklung, da dadurch die Entwicklung bedeutend vereinfacht wird. Besonders die automatisch sichergestellte Browserkompatibilität ist dabei ein entscheidender Vorteil. Zur Verwendung eines Frameworks muss dabei in der Regel nur eine Java-Script-Datei im Kopf des HTML-Dokuments eingebunden werden. Insbesondere wenn sich, wie bei vielen aktuellen Webangeboten der Fall, ein großer Teil der Programmierung auf den Client verlagert, ist die Verwendung eines (guten) Frameworks sehr anzuraten.

Ein sehr verbreitetes Framework ist *Prototype* [STEPHENSON2006]. Zusammen mit der darauf aufbauenden Effektbibliothek *script.aculo.us* [FUCHS2006] bildet es in einem kompakten Paket eine solide Basis für JavaScript-lastige Webangebote mit einer komplexen Oberfläche. Diese Kombination wird auch in dem beliebten Webentwicklungs-Framework *Ruby on Rails* (vgl. 3.5.3) verwendet. Zwei weitere aktive und verbreitete Projekte in diesem Bereich sind *jQuery* [RESIG2006] und *mootools* [PROIETTI2006]. Diese Frameworks haben alle gemeinsam, dass sie eine vor allem kompakte (im Sinne von Kilobytes) Basis bieten wollen, auf der einfach strukturierter Java-Script-Code geschrieben werden kann, in dem oft gebrauchter Code in dem Framework gekapselt wird. Als Anschauungsbeispiel soll hier eine Codezeile von der jQuery-Webseite [RESIG2006] dienen:

```
$("p.surprise").addClass("ohmy").show("slow");
```

Dieser Code durchsucht das ganze Dokument nach Paragraph-Elementen mit der CSS-Klasse „surprise“, fügt diesen Elementen die CSS-Klasse „ohmy“ hinzu und blendet sie schließlich langsam ein. Ohne ein Framework würde eine ganze Seite Code gebraucht, um das gleiche Ergebnis zu erreichen.

Zwei weitere Frameworks tun sich weniger durch ihre Kompaktheit, sondern durch ihren Umfang hervor: Die *Yahoo! UI Library* (YUI) [YAHOO2006] und das *Dojo Toolkit* [DOJO2006]. Der Gesamtumfang des Quelltextes dieser Frameworks bewegt sich dabei im Bereich von mehreren Megabytes. Neben der Funktionalität, die auch oben angesprochene Frameworks bieten, werden hier auch eine Menge von Widgets zur Verfügung gestellt, die der Entwickler in sein Webangebot einbinden kann. Im Fall von Dojo reicht es dabei, bestimmte Tags in HTML mit

dem Attribut „dojoType“ zu belegen. Das Framework ersetzt betreffende Elemente dann automatisch mit dem gewünschten Widget. Neben der einfachen Handhabung hat das auch den Vorteil, dass ein Webangebot so auch bei deaktiviertem JavaScript benutzbar bleibt, da in diesem Fall das ursprüngliche Element erhalten bleibt. Dojo bietet außerdem noch ein exzellentes Event-System, sowie ein umfangreiches Kommunikationssystem, das alle denkbaren Kommunikationsarten für Webangebote unterstützt und dabei auch Zugriff auf die Browser-History (Vor und Zurück) bietet. Auf diese Weise können mit Dojo einige große Accessibility- und Usability-Probleme gelöst werden. In der neuesten Version (4.0) wurden die Accessibility-Features von Dojo noch bedeutend erweitert. So bietet es Unterstützung für erweiterte Tastaturfunktionen, sowie für die Kompatibilität mit Screenreadern, oder Browsern im Kontrast-Modus für Sehbehinderte.

3.5.2 Rich Internet Application Frameworks

Neben den in JavaScript implementierten Frameworks und Toolkits gibt es auch Frameworks, die die Entwicklung von Webangeboten (insbesondere von Rich Internet Applications, im Folgenden RIA genannt) in einer eigenen Sprache ermöglichen. Zur Ausführung im Browser wird die Anwendung dann in ein im Browser lauffähiges Format kompiliert. Der Entwickler kann sich also auf die Gestaltung der Oberfläche und die Programmierung der Programmlogik konzentrieren, ohne sich dabei um die technische Umsetzung und die korrekte Darstellung in verschiedenen Browsern kümmern zu müssen. Jedes Framework bietet dem Entwickler dabei ein reichhaltiges Sortiment an vordefinierten Oberflächenkomponenten für RIAs (Rich Internet Applications) an.

Ein Open-Source-Framework in diesem Bereich ist **OpenLaszlo** [OLASZLO2006]. OpenLaszlo-Anwendungen werden in LZX programmiert, ein XML-Dialekt mit eingebettetem JavaScript. Der LZX Code ist dabei absolut unabhängig von der Implementierung auf der Zielplattform (vgl. Abbildung 3.2). Bei der Ausführung wird der Code dann vom OpenLaszlo Compiler in ein vom Browser verständliches Format übersetzt. Momentan wird nur Flash als Ausgabeformat unterstützt, in Zukunft werden OpenLaszlo-Anwendungen jedoch auch in DHTML ausgegeben werden können. Weitere Ausgabeformate können mit der Zeit hinzugefügt werden. Eine Besonderheit von OpenLaszlo ist, dass die Anwendung auf dem Webserver kompiliert wird, sobald ein Zugriff auf die Anwendung erfolgt. Das bedeutet, dass für den Entwickler der Schritt der Kompilierung entfällt – er kann seinen Code direkt auf den Webserver laden. Das bedeutet allerdings auch, dass für die Ausführung von OpenLaszlo-Anwendungen ein spezieller Server notwendig ist. Kürzlich wurde jedoch ein separater Compiler zur Verfügung gestellt, mit dem eine Anwendung vorkompiliert werden kann und somit auf allen Webservern lauffähig ist. In diesem Modus sind jedoch einige Möglichkeiten von OpenLaszlo nicht verfügbar.

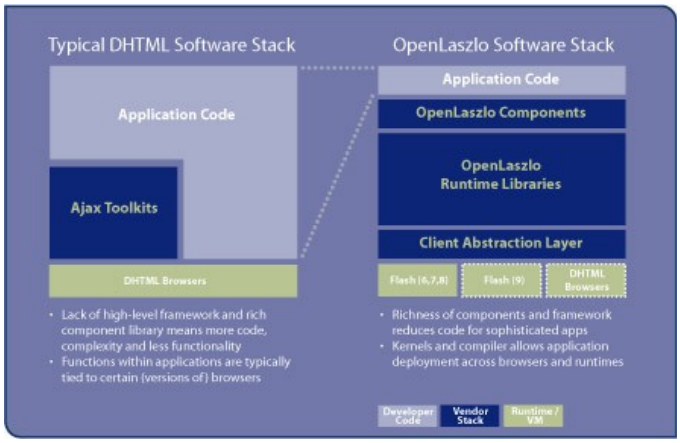


Abbildung 3.2: OpenLaszlo Software Stack, Quelle: [OLASZLO2006]

Eine einfache OpenLaszlo-Anwendung, mit der die Datensätze einer XML-Datei ausgegeben werden, sieht folgendermaßen aus:

```
<canvas>
  <dataset name="dset" src="phonebook.xml"/>
  <simplelayout axis="y"/>
  <view datapath="dset:/phonebook/employee">
    <simplelayout axis="x"/>
    <text datapath="firstName/text()"/>
    <text datapath="lastName/text()"/>
    <text datapath="phone/text()"/>
  </view>
</canvas>
```

Das RIA-Framework **Flex 2** von Adobe ist ähnlich aufgebaut [ADOBE2006]. Anwendungen werden bei Flex 2 mit den Sprachen MXML (Definition der Oberfläche) und ActionScript (Programmierung der Logik) erstellt. ActionScript ist eine JavaScript ähnliche objektorientierte Programmiersprache; der Code kann dabei entweder in die MXML-Dateien eingebunden oder in separaten Dateien abgelegt werden. Das Ausgabeformat ist hier jedoch ausschließlich Flash. Für Flex 2 bietet Adobe eine auf Eclipse basierende Entwicklungsumgebung an, die allerdings kostenpflichtig ist. Alle Dateien lassen sich jedoch auch mit einem Texteditor erstellen und mit dem kostenlosen Flex-Compiler in ausführbare Flash-Dateien umwandeln. Die Entwicklungsumgebung bietet einen visuellen Oberflächendesigner, sowie Live-Debugging und natürlich obligatorische Features wie Code-Highlighting und -Suggestion.

Adobe selbst gibt die Verbreitung seines Flash-Plugins, das zum Betrieb von Flex-Anwendungen im Browser notwendig ist, mit 97% an. Trotzdem stellt die Beschränkung auf das Flash-Format ein Accessibility-Problem dar, das jedoch im Kontext von RIAs in vielen Fällen von nicht so großer Bedeutung sein wird. Auf der Gegenseite bietet Adobe eine ausgewachsene Entwicklungsumgebung, eine einheitliche Zielplattform, sowie wesentlich bessere Möglichkeiten in der Oberflächengestaltung als in aktuellen Browsern.

Einen ganz anderen Ansatz verfolgt das **GWT** (Google Web Toolkit) [GOOGLE2006]. Hier wird das Webangebot komplett in Java programmiert. Dabei können alle für Java verfügbaren Entwicklungsumgebungen und -methoden verwendet werden. Der GWT-Compiler konvertiert die Java-Anwendung schließlich in JavaScript und HTML. Während der Entwicklung kann der Java-Code allerdings auch direkt im „Hosted Mode“ auf der JVM zum Testen und Debuggen ausgeführt werden. Dieser Ansatz ermöglicht Java-Programmierern einen nahtlosen Einstieg in die Webentwicklung.

3.5.3 Ruby on Rails

Ruby on Rails (im Folgenden *Rails*) ist ein serverseitiges Open-Source-Framework, das auf die Entwicklung von datenbankgestützten Webangeboten mit der Skriptsprache Ruby spezialisiert ist. Rails setzt dabei voll auf das Model-View-Controller Prinzip, also auf die strikte Trennung zwischen Datenmodell, Datenansicht und Programmlogik. Dadurch wird ein klar strukturierter und leicht wartbarer Code erreicht. Die MVC-Architektur wird dabei komplett von Rails verwaltet – der Entwickler kann sich direkt um die Definition der anwendungsrelevanten Teile kümmern, getrennt nach Datenschema, Datenansicht und Anwendungslogik. Rails fügt diese Teile dann zur Laufzeit zusammen. Viele typische Arbeitsschritte werden von Rails dabei automatisch durchgeführt, so werden z.B. aus dem Datenschema automatisch Ein- und Ausgabeseiten erzeugt, die der Entwickler nur noch anpassen muss. Die Realisierung von Eingabe-Validierung reduziert sich dabei für den Entwickler auf die Angabe eines Constraints für ein Datenfeld – die Umsetzung erfolgt wiederum automatisch von Rails. Jeder von Rails erzeugte Code ist dabei leicht änderbar. Die Änderung des Datenschemas, sowie die Migration von Da-

tenbanken kann auch bei einem schon laufenden Webangebot leicht erfolgen – möglich ist das durch die saubere MVC-Architektur und den sehr flexiblen Aufbau von Rails.

Der Ansatz von Rails kommt den oben beschriebenen modellbasierten Web-Engineering-Ansätzen recht nahe, da eine Aufteilung in Struktur, Logik und Präsentation erfolgt, verbunden mit einigen Vorteilen, die auch modellbasierten Ansätzen zu eigen sind. Der Vorteil von Rails ist dabei jedoch, dass man direkt zu Ergebnissen kommt, da es keine Trennung zwischen Design und Implementierung gibt. Auch wenn das gerade in modellbasierten Ansätzen bewusst gewünscht ist, entspricht es doch nicht dem Stil und den Wünschen von vielen Programmierern. Rails ermöglicht eine sehr schnelle Realisierung von funktionalen Prototypen, behält dabei jedoch trotzdem eine hohe Flexibilität.

Einige bekannte Web 2.0 - Webangebote wurden mit Rails realisiert, wie z.B. Backbase (www.backbase.com) und 43things (www.43things.com) [ROR2006].

3.6 Vorgefertigte Systeme

Viele Webangebote setzen auf vorgefertigte Systeme auf, die dann mehr oder weniger angepasst werden. Ein wesentliches Unterscheidungsmerkmal dieser Systeme ist dabei, wieviele Möglichkeiten das System bietet und wie hoch der Grad der Anpassbarkeit ist. Meistens ist ein System spezialisiert auf eine bestimmte Art von Webangebot, z.B. Blogs oder Wikis, manchmal lassen sich mit einem System auch ganz verschiedenartige Webangebote erstellen. Grundsätzlich kann man bei vorgefertigten Systemen zwei Arten unterscheiden:

- **Vorinstallierte Systeme** laufen auf dem Webserver eines bestimmten Anbieters. Nach einer (evtl. kostenpflichtigen) Registrierung erzeugt der Anbieter eine neue Instanz des von ihm angeboteten Systems. Dieses neu erzeugte Webangebot kann dann – innerhalb der vom Anbieter festgelegten Grenzen – angepasst und mit Inhalt gefüllt werden. Unter einer bestimmten Adresse ist das Webangebot dann für alle zugreifbar. Oft ist das eine Subdomain unter der Domain des Anbieters. Der Vorteil ist für den Benutzer (von einem Entwickler kann man in diesem Fall nicht sprechen) hier, dass er sich nicht um die Bereitstellung eines Webservers und die Installation des Systems kümmern muss.
- **Herunterladbare Systeme** müssen von dem „Webmaster“ selbst auf einem Webserver installiert werden. Dabei müssen oft auch noch Konfigurationsdateien angepasst werden. Dies erfordert zum einen Zugriff auf einen Webserver und zum anderen ein wenig Erfahrung im Umgang mit verschiedenen Web-Technologien. Der große Vorteil ist jedoch, dass hier (theoretisch) keine Beschränkungen in der Anpassung des Systems bestehen und dass das Webangebot unter einer eigenen Domain laufen kann.

Da viele Webangebote sich doch sehr ähneln, gibt es auch für die meisten Anforderungen schon passende vorgefertigte Systeme. Der Vorteil liegt auf der Hand: Man erhält ein ausgereiftes System, ohne auch nur eine Zeile Code geschrieben zu haben. Der Nachteil ist, dass man in der Entwicklung des Webangebots jetzt mehr oder weniger auf die Möglichkeiten des vorgefertigten Systems beschränkt ist. Handelt es sich um ein selbst installiertes System, kann man zwar theoretisch beliebige Änderungen daran vornehmen. Allerdings verlangt das erstens die Einarbeitung in den Quellcode, und zweitens verliert man dadurch die Möglichkeit, später eine neuere Version des Systems zu verwenden. Durch den Einsatz eines vorgefertigten Systems verlässt man also den Boden seiner bevorzugten Entwicklungsumgebung und Programmiersprache und vertraut sich der webbasierten Umgebung des Systems an. In manchen Fällen muss man dabei eine spezielle Skriptsprache oder den Umgang mit den Schnittstellen des Systems lernen. Diese Entscheidung muss sorgfältig getroffen werden und sollte im Sinne eines ordentlichen Web-Engineering erst nach erfolgter Anforderungsanalyse sowie nach erfolgtem Design

der Anwendungsstruktur und -oberfläche erfolgen.

3.6.1 Blogs

Blogs sind in ihrer Funktionalität recht klar definiert und werden daher fast ausschließlich mit vorgefertigten Systemen realisiert. Es gibt zahlreiche Anbieter, die vorinstallierte Blogs anbieten - die größten sind Blogger von Google (blogger.com), Windows Live Spaces von Microsoft (spaces.live.com) und MySpace (myspace.com). Netcraft berichtet, dass allein im September 2006 mit Windows Live Spaces 1,3 Millionen Webangebote erstellt wurden und mit Blogger über 400.000 [NETCRAFT2006]. Durchschnittlich werden momentan insgesamt 2,3 Millionen neue Webangebote pro Monat erstellt – ein großer Teil kommt dabei von den Blogging-Anbietern. Auch wenn diese Zahlen sicher nicht überbewertet werden dürfen, so ist doch zumindest erkennbar, dass vorinstallierte Systeme heute eine bestimmende Rolle in der Realisierung von Webangeboten haben.

Herunterladbare Blog-Systeme wie z.B. Wordpress [AUTOMATIC2006] haben heute schon eine beachtliche Funktionalität und werden nicht selten an Stelle von herkömmlichen CMS eingesetzt (vgl. [BOYINK2005]). Der Vorteil von Blogs ist, dass sie „out-of-the-box“ eine etablierte Navigation für inhaltsreiche Webangebote inklusive einer Suchfunktion bieten und außerdem Inhalt und Layout sauber trennen. Die Gestaltung von Blogs erfolgt immer über Templates und CSS. Im Bereich der Inhaltsverwaltung haben viele Blog-Systeme ein ausgewachsenes Rechte- und Workflowsystem. Wordpress bietet eine automatische Rechtschreibprüfung und verbessert sogar die typographische Qualität von Texten anhand etablierter Prinzipien. Die Bereitstellung von Inhalten in maschinenlesbarer Form (Webfeeds) funktioniert dabei ebenso „out-of-the-box“ wie die Kommentierung und Verlinkung (Trackback, Pingback, Permalink) von Inhalten. Wordpress unterstützt außerdem noch das Einrichten von „Seiten“ - der zentralen Metapher in den meisten herkömmlichen CMS. So lassen sich damit auch Webangebote realisieren, die neben der dynamischen Navigation noch eine statische Struktur brauchen.

Etablierter Standard bei vielen Blog-Systemen ist außerdem die Möglichkeit, neue Einträge über Email einzutragen. Wordpress bietet darüber hinaus noch eine Schnittstelle für Mobiltelefone.

Aktuelle Blog-Systeme ermöglichen also schon von Haus aus ein hohes Maß an Accessibility, Usability und Flexibilität – sofern man nicht allzu große Fehler in der Bedienung und Anpassung des Systems macht.

3.6.2 Content-Management-Systeme

Von der Bedeutung des Wortes her sind Content-Management-Systeme (im Folgenden CMS) eigentlich nur für den Bereich der Inhaltsverwaltung zuständig, also nur für einen abgegrenzten Teilbereich von Webangeboten. Die meisten Web-CMS gehen jedoch darüber hinaus und verwalten mehrere Bereiche eines Webangebots, also z.B. auch die Darstellung des Inhalts, die Navigation, oder die Benutzerverwaltung. Die zentralen Vorteile von CMS sind die Trennung des Inhalts von Layout und Struktur, sowie eine Webschnittstelle (Backend), mit der Inhalte einfach bearbeitet und hinzugefügt werden können. Oft werden über die Webschnittstelle auch weitere Bereiche des Webangebots verwaltet.

Im Prinzip sind die oben beschriebenen Blog-Systeme auch Content-Management-Systeme, sie übertreffen dabei sogar so manches herkömmliche CMS in Funktionalität und Komfort. Blogs haben jedoch eine recht spezielle, eindimensional angelegte Struktur und werden daher in der Regel nicht mit einem CMS assoziiert. Es gibt jedoch so viele Systeme, die sich mit dem Titel „CMS“ schmücken, dass eine eindeutige Charakterisierung schwierig ist. Im Folgenden werden daher drei verbreitetere CMS beispielhaft behandelt.

Bei **TYPO3 [**TYPO32006] werden Inhalte in hierarchisch strukturierten Seiten organisiert. Das CMS hält sich also konzeptionell eng an die Seiten-Metapher. Die Struktur des Webangebots wird in einer Baumstruktur angezeigt, in der die einzelnen Seiten hierarchisch angeordnet sind und an der auch die Navigation abgeleitet wird (vgl. Abbildung 3.3).

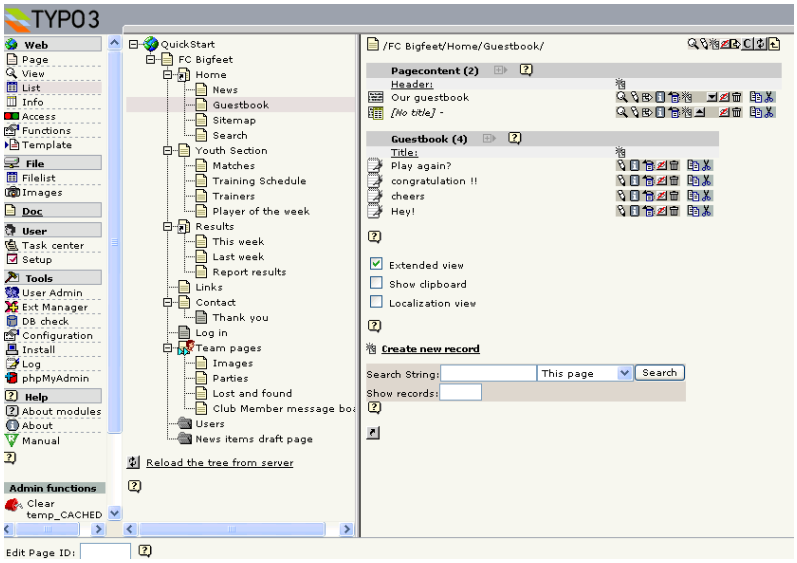


Abbildung 3.3: TYPO3-CMS, Quelle: [TYPO32006]

In der in dieser Arbeit durchgeführten Umfrage zu Verwaltungs-Werkzeugen für Webangebote wurde das TYPO3-CMS als sehr umständlich bewertet, allerdings wurde die Funktionalität des CMS sehr hervorgehoben (Details unter 3.7.1). Die Oberfläche wirkt sehr überladen (vgl. Abbildung 3.3) - die offensichtlich große Funktionalität ist daher sehr schwer zugänglich. Es ist außerdem fraglich, ob die Strukturierung durch Seiten die optimale Wahl für eine datenbankgestütztes CMS ist. Die Seiten-Metapher entstammt den Ursprüngen des Web, als Webangebote aus statischen HTML-Dateien bestanden – bei der Verwaltung eines Webangebots ohne CMS war man daher gezwungen auf der Seitenebene zu arbeiten. Dieses Konzept hat TYPO3 übernommen und bis heute nicht abgelegt. Es findet hier also keine konsequente Trennung zwischen Inhalt und Struktur, bzw. Navigation statt. Bleibt der Umfang eines Webangebots innerhalb eines gewissen Rahmens, lässt es sich mit TYPO3 noch gut überblicken und verwalten. Für regelmäßig aktualisierte Webangebote muss jedoch die Skalierbarkeit dieser Struktur in Frage gestellt werden.

Joomla! [JOOMLA2006] hat eine etwas dynamischere Struktur als TYPO3. Inhalte werden hier in *Sections* organisiert. Die grundsätzliche Struktur eines Inhaltsobjekts ist dabei festgelegt. Die einzige *Seite,* die bei Joomla! zu finden ist, ist die Startseite, auf der einzelne Inhaltsobjekte und Komponenten angeordnet werden können. Die Navigation eines Webangebots wird über die Verwaltung beliebig vieler Menus definiert. In den Menus können sowohl Links zu einzelnen Inhaltsobjekten, als auch zu Sections, Modulen und funktionalen Komponenten des Systems angeordnet werden.

RedDot [REDDOT2006] ist nicht so sehr bekannt wie die oben beschriebenen CMS, was sicher auch daran liegt, dass es ein kommerzielles Produkt ist. Es bietet jedoch einige interessante und innovative Ansätze, die hier einer Erwähnung wert sind.

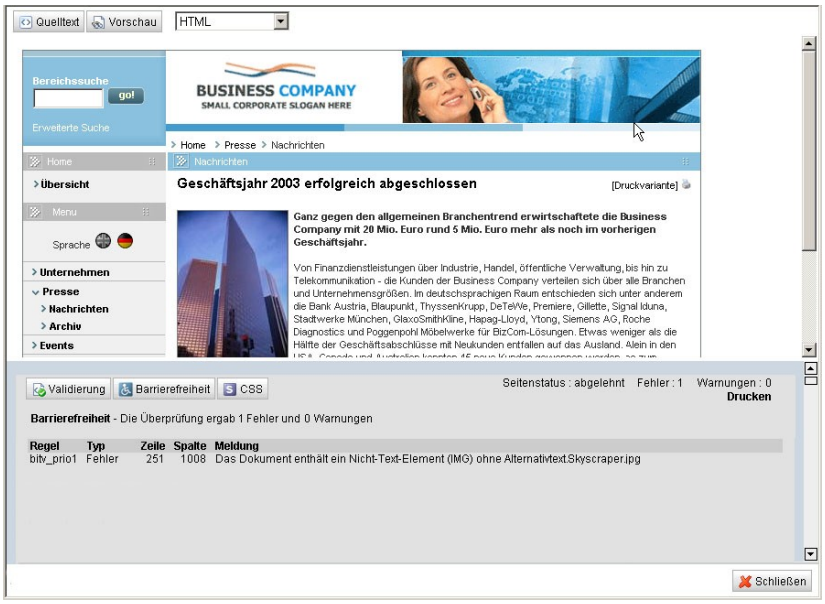


Abbildung 3.4: RedDot CMS, Quelle: [REDDOT2006]

Die Bearbeitung des Webangebots erfolgt nicht in einem gesonderten Backend, sondern in der Frontend-Ansicht, also in einer Art WYSIWYG-Ansatz. Dabei werden alle editierbaren Elemente in einer Ansicht mit einem roten Punk (in Anlehnung an den Firmenname „RedDot“) versehen. Klickt man auf diese Punkt, kann man das betreffende Element an derselben Stelle editieren. RedDot überprüft dabei eingefügte Inhalte sofort anhand bestimmter Kriterien. Neben einer Rechtschreibprüfung bietet das CMS auch eine Überprüfung auf validen Code und Barrierefreiheit (Accessibility). Wenn z.B. ein Bild ohne Alternativtext eingefügt wird, so schlägt das System Alarm und verhindert die Veröffentlichung (vgl. Abbildung 3.4).

3.6.3 Webanwendungs-Generatoren

Die oben angesprochenen vorgefertigten Systeme drehen sich mehr oder weniger um die Veröffentlichung oder Erzeugung von Inhalten. Einen Schritt weiter gehen Webanwendungs-Generatoren wie Ning [NING2006]. Im Prinzip kann der registrierte Benutzer auf der Plattform fast beliebige Webanwendungen entwickeln. Das Grundprinzip dabei ist, dass eine vordefinierte Anwendung dupliziert wird und dann vom Benutzer beliebig angepasst werden kann. Dabei kann der Quellcode (PHP) der Anwendung auf der Plattform editiert werden; es besteht außerdem die Möglichkeit, beliebige Dateien (also auch Quellcode) hochzuladen. Die Anwendung kann dabei die Infrastruktur (Datenbank, Webspace) und die vordefinierten Komponenten von Ning verwenden. Die Bearbeitung von Quellcode in einem Browserformular bzw. der Upload einzelner Quellcode-Dateien über den Browser ist jedoch für größere Entwicklung kaum praxistauglich. Außerdem ist es fraglich, ob nicht-professionelle Benutzer in der Lage sind, Anpassungen an einem Quellcode vorzunehmen. Die Stärke von Ning liegt also eindeutig in der Duplizierung von vorgefertigten Anwendungen. Anpassungen in Funktionalität und Layout sind zwar beliebig flexibel, richten sich auf Grund des Low-Level-Ansatzes aber eher an erfahrene Benutzer bzw. Entwickler. Dadurch, dass auch Webanwendungen von anderen Benutzern dupliziert werden können, ist die Auswahl so groß, dass auch unerfahrene Benutzer zu der eigenen Webanwendung kommen können, die sie sich vorstellen. In Bezug darauf ist eine Studie von Rode interessant, in der er herausgefunden hat, dass 75% der Webanwendungen, die Nicht-Programmierer gerne erstellen würden, durch fünf Grundanwendungen abgedeckt werden können [RODE2004].

Eine bedeutende Einschränkung ist natürlich auch, dass die Webanwendungen unter der

Domain von Ning laufen. Eine Navigationsleiste des Anbieters wird dabei in jeder Webanwendung am Anfang der Seite angezeigt.

3.7 Praxis und Wünsche von Webentwicklern

Theorie und Praxis gehen bekanntlich oft auseinander – das ist auch im Bereich der Webentwicklung nicht anders. In einem gewissen Sinn ist auch der Entwickler ein Endanwender – Entwicklungsmethoden und -werkzeuge müssen seinen Wünschen und Zielen entsprechen. Daher soll es in diesem Abschnitt um die aktuelle Praxis, sowie um die Wünsche und Ziele von den an der Webentwicklung beteiligten Personen gehen.

3.7.1 Umfrage zu Verwaltungswerkzeugen für Webangebote

Im Rahmen dieser Arbeit wurde eine Online-Umfrage mit dem Titel „Umfrage zu Management-Tools für Webseiten“ durchgeführt (siehe Anhang A). Das Ziel dieser Umfrage war, herauszufinden, wie die Praxis in der Benutzung von webbasierten Verwaltungswerkzeugen für Webangebote aussieht. Dabei sollten außerdem Hindernisse für den erfolgreichen Umgang mit diesen Werkzeugen, sowie die Wünsche der Benutzer herausgefunden werden. Von den 27 Teilnehmern hatten 26 schon mal bei einem Webangebot in irgendeiner Weise mitgewirkt, 21 davon haben dabei auch ein CMS benutzt. In Bezug auf allgemeine Erfahrung mit Webseiten bewerteten sich alle als „erfahren“ oder „sehr erfahren“. 70% der Teilnehmer bewerteten sich mindestens als „erfahren“ in der Mitarbeit bei einem Webangebot; in Bezug auf die technische Umsetzung schätzten sich sogar 81% als mindestens „erfahren“ ein, wobei nur 50% davon sich als Programmierer bezeichnet haben (bei insgesamt 44% Programmierern). Es handelt sich also tendenziell um technisch erfahrene Leute, die schon Erfahrungen im Aufbau und Betreiben von Webangeboten gesammelt haben. Im Folgenden werden einige interessante Ergebnisse der Umfrage wiedergegeben.

Die Frage „Wie wurden/werden die verschiedenen Bereiche der Webseite umgesetzt, bei der Sie beteiligt waren/sind?“ ließ einige interessante Trends erkennen (vgl. Abbildung 3.5). Es lässt sich sagen, dass ein „All-In-One Tool“ (im Folgenden WMS – Web Management System – genant), das *alle* Bereiche in der Entwicklung und Verwaltung eines Webangebots abdeckt, so gut wie nicht in den Ergebnissen auftaucht. Erwartungsgemäß wurde die Inhaltsverwaltung von den meisten durch ein CMS realisiert, außerdem wurden von 46% die technische Umsetzung sowie von 27% die Benutzerverwaltung einem CMS-ähnlichen Werkzeug zugeordnet. Letzteres lässt sich sicher darauf zurückführen, dass viele aktuelle CMS eine Benutzerverwaltung enthalten. Ersteres weist darauf hin, dass hier keine eigene Programmierung erfolgt ist, sondern ein vorgefertigtes Webangebot verwendet worden ist, das dann über ein CMS angepasst wurde.

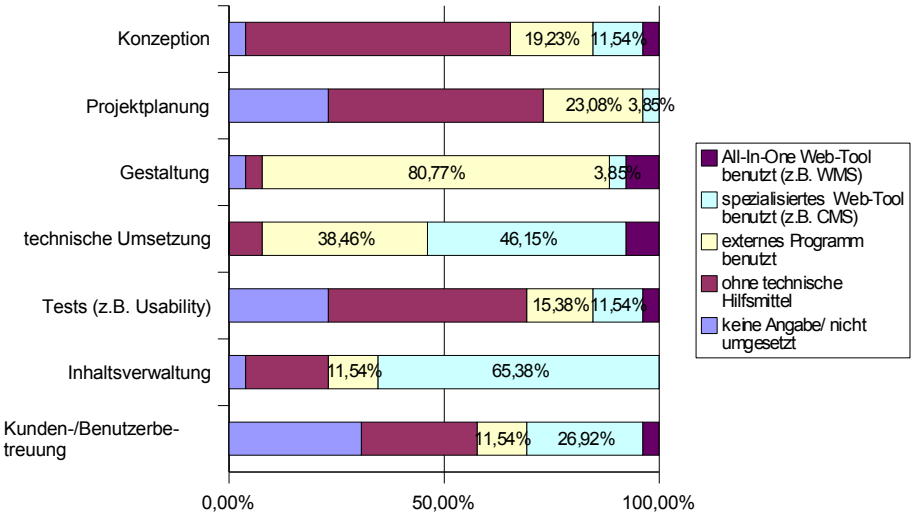


Abbildung 3.5: Umfrage: Wie wurden genannte Bereiche bei der Realisierung eines Webangebots umgesetzt (N=26)

Interessant ist, dass zwar 81% der Teilnehmer die Gestaltung in einem externen Programm umgesetzt haben, aber 69% die Integration der Gestaltung in ein WMS für sinnvoll oder sehr sinnvoll halten würden (vgl. Abbildung 3.6). Die Umsetzung der Gestaltung stellte sich außerdem als größte Hürde bei der Erstellung des Webangebots heraus. Dabei wurde vor allem das Thema Browserkompatibilität als Problem genannt (7 Nennungen). Fünf Mal wurde prinzipiell die mangelnde Möglichkeit oder Fähigkeit zur Umsetzung des gewünschten Layouts beklagt, vier weitere Male wurde mangelnde Flexibilität in der grafischen Umsetzung in Bezug auf das benutzte CMS negativ erwähnt. Es lässt sich also schließen, dass hier ein großer Bedarf herrscht und dass gestalterische Möglichkeiten in einem WMS prinzipiell begrüßt und gewünscht werden.

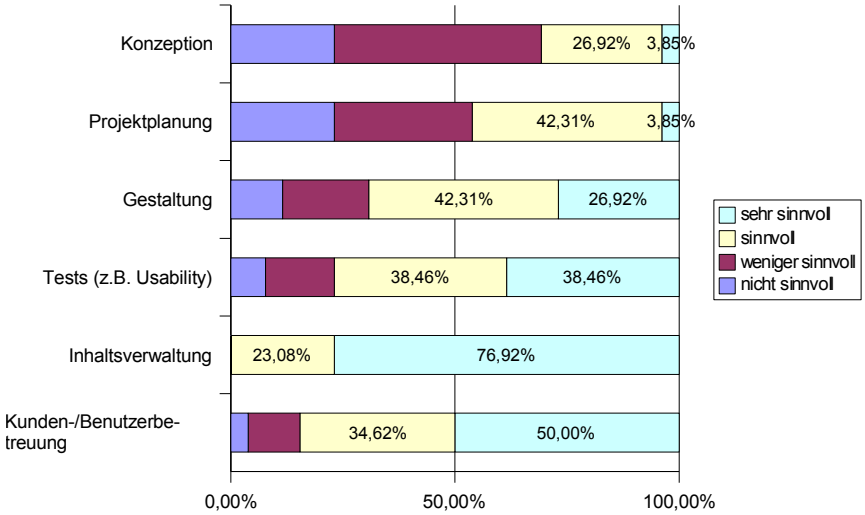


Abbildung 3.6: Umfrage: Bewertung der Integration genannter Bereiche in ein All-in-One Tool (N=26)

Ein weiterer auffälliger Punkt ist, dass die Bereiche Konzeption, Projektplanung und Tests von der großen Mehrheit entweder gar nicht, oder ohne technische Hilfsmittel durchgeführt wurde. Demgegenüber steht, dass 77% der Teilnehmer eine Integration von Tests in ein WMS begrüßen würden. Bei Projektplanung und Konzeption herrscht dabei etwas mehr Zurückhaltung. Es lässt sich also zumindest schließen, dass ein Bedarf an besseren Möglichkeiten zum Testen da ist.

Unter den Teilnehmern befanden sich acht Typo3-Nutzer – genug, um zumindest einen Eindruck davon zu bekommen, wie dieses verbreitete CMS aus der Praxis heraus bewertet wird. Der Lernaufwand wurde hierbei einstimmig mit „schlecht“ oder „sehr schlecht“ bewertet, die Nutzungsgeschwindigkeit von der Mehrheit eher negativ und der Funktionsumfang fast durchweg positiv (vgl. Abbildung 3.7). Darüberhinaus fielen in der Bewertung Begriffe wie „umständlich“, „langsam“, „Verwirrung“, „keine Usability“ oder „gewöhnungsbedürftig“. Es lässt sich also sagen, dass Typo3 also vor allem wegen seinem Funktionsumfang verwendet wird, obwohl es ansonsten eher als unzugänglich und umständlich angesehen wird. Das könnte ein Hinweis darauf sein, dass viele Benutzer sich Systeme mit möglichst umfassenden Möglichkeiten wünschen und sogar bereit sind, sie zu nutzen, selbst wenn sie nur sehr schlecht benutzbar sind.

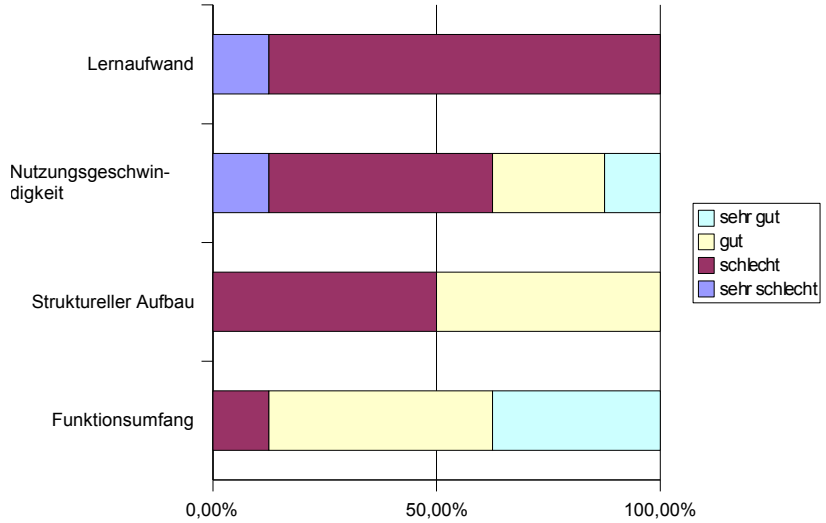


Abbildung 3.7: Umfrage: Bewertung von Typo3-Benutzern über das CMS (N=8)

Eine wichtige Frage war, welchen Ansatz die Teilnehmer für die Bearbeitung von Inhalten bevorzugen würden: einen rein struktur-orientierten, eine Mischung aus struktur- und gestaltungsorientiertem, oder einen WYSIWYG-Ansatz (vgl. Abbildung 3.8). Die Ergebnisse hierzu decken sich z.T. mit dem schon oben beschriebenen Wunsch nach mehr Gestaltungsfreiheit. Einen rein strukturierten Ansatz würden nur 11% der Teilnehmer vorziehen. Interessant dabei ist, dass diese ausschließlich aus erfahrenen Programmierern bestehen. Der Rest teilt sich auf die anderen beiden Ansätze auf, wobei WYSIWYG mit 48% etwas öfter bevorzugt wurde, als der „Hybrid-Ansatz“ (41%). Eine interessante Zusatzinformation ist, dass 73% der Teilnehmer, die sich für einen Hybrid-Ansatz ausgesprochen haben, es für sinnvoll oder sehr sinnvoll halten, den Bereich der Gestaltung in ein WMS zu integrieren.

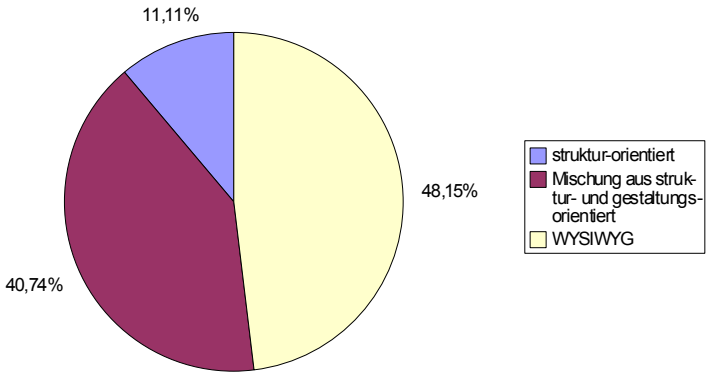


Abbildung 3.8: Umfrage: Bevorzugter Ansatz zur Bearbeitung von Inhalten (N=27)

Diese Ergebnisse weisen auf einen starken Wunsch nach Gestaltungsfreiraum hin, wie er auch von WYSIWYG-Desktop-Anwendungen bekannt ist. Es scheint, dass sich Benutzer mit einem zu stark struktur-orientierten Ansatz, wie er vom Aspekt der Trennung von Inhalt und Layout her eigentlich wünschenswert wäre, nicht so leicht anfreunden können. Hier besteht also die Herausforderung, dem Benutzer möglichst viel Freiraum zu geben ohne dabei die Flexibilität des Webangebots (im Sinne von 3.2.1) zu verlieren.

3.7.2 Weitere Studien

Wie im Verlauf dieser Arbeit schon deutlich wurde, hat man es im Web mit einer hohen Diversität an Entwicklern von Webangeboten zu tun – vom selbsternannten Hobby-Webmaster bis zum akademisch ausgebildeten Software-Architekt ist alles dabei. Dementsprechend gehen auch die Entwicklungspraxis sowie die Wünsche der einzelnen Entwickler weit auseinander.

Im professionellen Bereich wurden in den letzten Jahren einige Studien zur Entwicklungspraxis von Webangeboten durchgeführt. Schon 1998 befragte Whitley dazu in seiner Studie „Methodism in practice“ drei Firmen unterschiedlicher Größe, die preisgekrönte Webangebote entwickelt hatten [WHITLEY1998]. Er fand heraus, dass bei keiner der interviewten Firmen streng formelle Methoden wie z.B. OOHDM angewendet wurden, sondern eher proprietäre Methoden, die jeweils nach Kontext ausgesucht und angepasst und manchmal auch wieder völlig verworfen wurden. Grundsätzlich verstand man zu enge Methoden als unflexibel und unpassend für das Web.

McDonald führte 2001 Interviews mit Entwicklern aus sieben verschiedenen Web-Unternehmen aus Großbritannien durch [MCDONALD2001]. Als wichtigste Erkenntnis ergab sich hier, dass die verschiedenen Phasen im Webentwicklungs-Prozess kaum beachtet wurden. Bei einem starken Schwerpunkt auf die Implementierung wurden vor allem die Anforderungsanalyse zu Beginn der Entwicklung und ausreichende Evaluierung am Ende vernachlässigt. McDonald identifiziert diesen Sachverhalt als Hauptgrund für die großen Schwierigkeiten, die die interviewten Entwickler mit einer erfolgreichen Fertigstellung ihrer Projekte hatten. Er stellte zudem einen sehr hohen Leistungsdruck fest, der in dem hohen Marktdruck und der kurzen durchschnittlichen Entwicklungszeit von kaum 3 Monaten begründet ist. Dies bestätigen auch die Ergebnisse von Barry et al., die 2001 zwei größer angelegte Studien unter irischen Unternehmen durchführten:

> „There's increasing demand for them to deliver high-quality, complex, Web-based software products rapidly, and even the most advanced RAD processes are incapable of doing so" [BARRY2001, Seite 9].

Sie sehen diesen Zeit- und Leistungsdruck auch als eine der Ursachen für die mangelnde Übernahme von formalen Methoden in der Entwicklergemeinschaft – diese seien nicht praxistauglich und bedürften zu langer Einarbeitungszeit. Das Hauptproblem sehen sie jedoch nicht darin, dass es etwa Schwierigkeiten beim Verstehen oder Benutzen der Methoden gäbe. Laut 61,5% der Befragten seien diese einfach viel zu umständlich.

Lang et al. betonen jedoch in ihrer aktuellen Studie von 167 irischen Webentwicklungs-Firmen, dass die Mehrheit der professionellen Entwickler sich heute strukturierter und gut dokumentierter Prozesse bedient [LANG2005]. Damit wollen sie der These widersprechen, dass heutige Webangebot „ad hoc" entwickelt, bzw. „gehackt" würden und sich das Web daher in einer Krise befände. Allerdings bestätigen sie die schon 1998 von Whitley gemachte Beobachtung, dass dabei proprietäre Methodiken angewandt werden und nicht etwa von der Forschung vorgeschlagene Methoden wie OOHDM. 95% der befragten Entwickler nutzten selbstgemachte Ansätze und nur 2% haben jemals nicht-proprietäre Methoden angewandt. Auch der Mitautor dieser Studie, Brian Fitzgerald, hatte schon 1997 in einer Studie die Beobachtung gemacht, dass Methodiken aus der Forschung von erfahrenen Programmierern nicht in der vorgeschriebenen Form „sklavisch" übernommen werden [FITZGERALD1997]. Vielmehr würden Methoden vom Entwickler individuell „zurechtgeschneidert", oder nur teilweise angewandt – je nachdem wie es pragmatisch erscheint.

Etwas negativere Erfahrungen wie Lang et al. machten Taylor et al. in einer 2002 durchgeführten Studie bei 25 Webentwicklungs-Firmen in Großbritannien [TAYLOR2002]. 68% der von ihnen untersuchten Firmen wendeten überhaupt keine formellen Techniken an und entwickelten in der Tat „ad hoc". Nur 7 der 25 Firmen konnten formale Testprozeduren aufweisen, nur 13 Firmen hatten formale Richtlinien für Web-Projekte aufgestellt. Bei der Befragung der Entwickler stellten sie außerdem fest, dass niemand akademische Literatur als nützliche Quelle in Bezug auf die Entwicklung von Webangeboten betrachtete.

Es wird also klar, dass man in Bezug auf die aktuelle Praxis in der Entwicklung von Webseiten keine allgemeingültigen Aussagen treffen kann. Es lässt sich jedoch festhalten, dass Webentwickler sich nicht an akademischen „Vorgaben" orientieren, sondern in der Wahl von Methoden und Prozessen sehr pragmatisch sind. Aus akademischer Sicht wird „pragmatisch" sicher anders definiert – Entwickler sehen sich in der Praxis jedoch extrem kurzen Produktzyklen ausgesetzt und entscheiden flexibel, was ihrer Meinung nach unter den speziellen Gegebenheiten zielführend ist. Wenngleich es Ausnahmen gibt, führt das jedoch in vielen Fällen zu einem Ad-Hoc-Ansatz in der Entwickelung, dem es vor allem einer gründlichen Anforderungsanalyse sowie späterer Evaluierung mangelt.

Auch im semi- bis nicht-professionellen Bereich finden sich einige interessante Studien. Die Bezeichnung „Webmaster" soll dabei symbolhaft für den semi- bis nicht-professionellen Entwickler stehen, der Webangebote im kleinen bis mittleren Maßstab erstellt. Betrachtet man die oben beschriebenen Ergebnisse, so liegt jedoch die Vermutung nahe, dass auch viele Entwickler von professionellen Webentwicklungs-Firmen in diesen Bereich fallen. Nichtsdestotrotz agieren Webmaster tendenziell auf einer viel informelleren Ebene, abseits von festgelegten Methoden und Prozessen. Geht man davon aus, dass ein großer Teil aller Webangebote von *Webmastern* erstellt wird, so wird die Wichtigkeit von hilfreichen Werkzeugen in diesem Bereich deutlich.

Rode et al. fanden in verschiedenen Studien heraus, dass Webmaster, die keine Programmierer sind, die Entwicklung von Webangeboten auf einer hohen Abstraktionsebene betrachten [RO-

DE2003]. Sie erwarten, dass viele Low-Level Funktionen wie Datenbankzugriff oder Sessions „out-of-the-box“ funktionieren. Als Konsequenz wird in [RODE2004] vorgeschlagen, in einem Webentwicklungswerkzeug folgende Funktionalitäten automatisch im Hintergrund bereitzustellen:

- Suche
- Statistiken
- Cronjobs, also regelmäßige, zeitgesteuerte Vorgänge
- Alle technischen Details in Bezug auf Datenbanken (Schema, Foreign Keys)
- Abfrage und Anzeige von Daten aus der Datenbank (Übersicht und Detailansicht)
- Benutzerverwaltung, Sessionverwaltung und Authentifizierung
- Rechteverwaltung
- Validierung von Formulareingaben
- kontextbedingte Ausgabe – also z.B. die Darstellung eines Login-Formulars abhängig davon, ob der Benutzer authentifiziert ist oder nicht.

Nicht-Programmierer sind diese Funktionen von der Benutzung anderer Webangebote gewöhnt und empfinden sie als selbstverständlich. Sie erwarten daher, dass diese Funktionen ohne besonderes Zutun verwendet werden können. Rode et al. berichten, dass sich viele Webmaster zur Umsetzung ihres gewünschten Layouts mehr freie Gestaltungsmöglichkeiten im Sinne von „Direct Manipulation“ wünschen, also im Prinzip einen WYSIWYG-Ansatz. Dies deckt sich mit den Beobachtungen, die in der Umfrage zu dieser Arbeit gemacht wurden.

In Bezug auf Nicht-Programmierer fassen Rode et al. die Erwartungen, die an ein Webentwicklungswerkzeug gestellt werden, folgendermaßen zusammen:

> „From a tool that intends to support nonprogrammers in the development of dynamic web sites they expect a rounded feature set that addresses all facets of web development. They want the tool to be accessible by providing predefined templates, and wizards while still leaving the developer in full control of the details.” [RODE2004, Seite 8]

Es sollten also möglichst viele vordefinierte Komponenten auf einer hohen Abstraktionsebene zur Verfügung gestellt werden, die der Webmaster dann verwenden und anpassen kann.

Rode et al. untersuchten außerdem die Erfahrungen von Webmastern im semi-professionellen Bereich. Dazu führten sie 2005 eine Umfrage mit 31 Teilnehmern, sowie eine intensivere Studie mit 10 Teilnehmern durch [RODE2005]. Unter Anderem untersuchten sie dabei die Frage, was den Webmastern bei der Entwicklung ihrer Webangebote am meisten Probleme machte (Abbildung 3.9). Das Ergebnis bestätigt die schon in 3.7.1 gemachte Feststellung, dass Browserkompatibilität und die Umsetzung des Layouts ein großes Problem darstellen. Als größte Schwierigkeit identifizieren Rode et al. jedoch das Gewährleisten der Sicherheit des Webangebots.

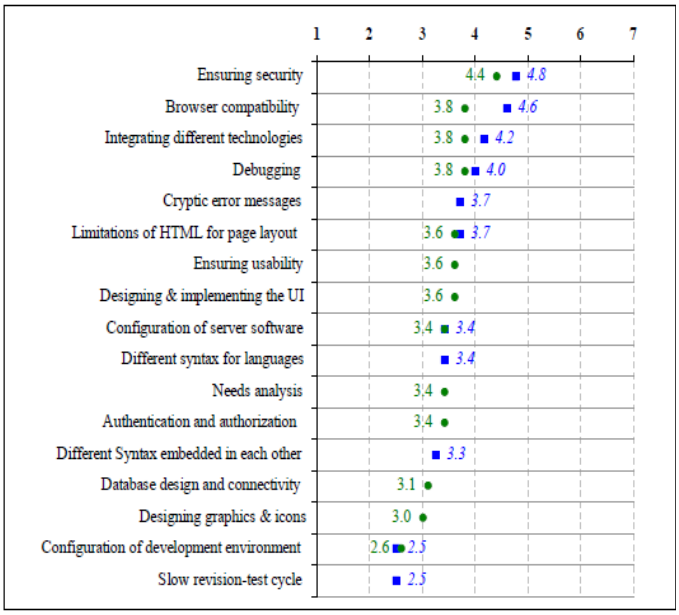


Abbildung 3.9: Responses to question about problems in web application development in two different surveys (1=not a problem at all; 7=severe problem), Quelle: [RODE2005]

Rode et al. charakterisierten den semi-professionellen Webmaster außerdem als *Benutzer*, der schnelle Ergebnisse sehen möchte. Die informelle Entwicklungsweise der Webmaster wies dabei keine separaten Phasen auf wie Analyse, Prototyping oder Testing, sondern wurde eher als ein evolutionärer Entwicklungsprozess beschrieben, der auf schnelle Fertigstellung ausgerichtet ist. Ausgehend von diesen Beobachtungen wurden unter Anderem folgende Anwendungen in Bezug auf Webentwicklungswerkzeuge vorgeschlagen [RODE2005]:

- Werkzeuge sollten robuster und vor allem *schneller* funktionieren und iterative Entwicklung unterstützen.

- Es sollten vordefinierte Komponenten angeboten werden. Dabei sollte dem Entwickler aber möglichst viel Kontrolle gelassen werden, um diese nach seinen Vorstellungen anzupassen.

- Automatisierbare Arbeitsschritte wie HTML-Validierung oder Accessibility-Checks sollten auch automatisch erledigt werden können, da dafür sonst in der Regel keine Zeit aufgebracht wird.

- Dem informellen Entwicklungsstil vieler Webmaster sollte entgegengekommen werden, da für sie ansonsten kein schneller Nutzungsvorteil ersichtlich ist.

4 W2MS – Web 2.0 Management System

Im Verlaufe dieser Arbeit wurde das Spannungsfeld zwischen einem „Web der unbegrenzten Möglichkeiten“ und der nüchternen Realität der Entwicklung desselben aufgezeichnet. In diesem Kapitel sollen schließlich beide Seiten in einem Konzept zusammengebracht werden. Es geht – analog zum Titel dieser Arbeit – um ein browserbasiertes Entwicklungs- und Verwaltungssystem für Webangebote, das dem „Status Quo“ des Web entspricht, sowie Aspekte des Web-Engineering mit einbezieht.

Der Name des hier beschriebenen Systems, bzw. Werkzeugs ist *W2MS*. Der Name soll zunächst eine Anlehnung an *CMS* sein, um die Verwandtheit mit Content-Management-Systemen anzudeuten, unter denen man im Kontext des Web im Allgemeinen browserbasierte Systeme versteht, die die Erstellung und Verwaltung eines Webangebots erleichtern. Das „W“ in W2MS soll jedoch auf einen ganzheitlicheren Ansatz hindeuten, der nicht nur auf die Verwaltung des Inhalts (**C**ontent) beschränkt ist, sondern möglichst viele Problemfelder im Kontext von **W**ebangeboten umfasst. Die „2“ steht, analog zur Web 2.0 - Definition in dieser Arbeit (vgl. 2.2), für die Berücksichtigung aktueller Techniken und Strukturen des Web. Da W2MS nicht *nur* ein CMS ist, sollte man es einer neuen Klasse von Werkzeugen zuordnen, die an dieser Stelle unter der Bezeichnung *WMS* (ohne die 2) eingeführt wird.

Auch wenn es sicher wünschenswert wäre, kann W2MS nicht alle Aspekte dieses (riesigen) Spannungsfelds zufriedenstellend abdecken. Die Suche nach einer allumfassenden, „ultimativen“ Anwendung oder Methode verbietet die Optimierung für den Spezialfall und ist damit letztliche kontraproduktiv. Mit dem vorliegenden Konzept wird daher eine Lösung für einen Teilbereich des bisher aufgespannten Problemfelds vorgeschlagen. Die Zielsetzung von W2MS wird im Folgenden einleitend beschrieben.

4.1 Vision und Zielsetzung

Ein Werkzeug ist für einen bestimmten Benutzer dann nützlich, wenn es die Funktionalität bietet, die er braucht, und wenn er es einfach und zufriedenstellend anwenden kann. In der Praxis ist ein Benutzer jedoch immer beschränkt durch die Werkzeuge, die er benutzt. Ein Beispiel hierfür ist die Reduzierung von persönlichen Webangeboten auf das Blog-Schema, die durch die massenhafte Benutzung von Blogging-Plattformen bedingt ist [NETCRAFT2006]. Hier wird eine einfache Erstellung von Webangeboten ermöglicht – allerdings ist es fraglich, ob das auf der jeweiligen Plattform bereitgestellte (Blogging-)Werkzeug jeden Wunsch erfüllt, den die Benutzer eigentlich haben. Das Resultat davon ist, dass viele Leute ihr Webangebot nach ein und demselben Schema aufbauen, weil für sie benutzbare Werkzeuge dafür verfügbar sind. Dies trifft im weiteren Maße auch auf alle anderen Werkzeuge zu. So hat ein CMS auch ein bestimmtes vordefiniertes Schema, das zwar flexibler ist als die Werkzeuge von Blogging-Plattformen, aber dennoch im Ergebnis immer zu ähnlich strukturierten Webangeboten mit ähnlicher Funktionalität führt. So ist es nicht verwunderlich, dass das Aufkommen von neuen Webangebots-Formen gleich die Schaffung des „Web 2.0“-Begriffs zur Folge hatte. Laut der Web 2.0 - Definition von O'Reilly gehören CMS zu der „Web 1.0-Ära“ [OREILLY2005], da sie symbolhaft dafür stehen, dass ein Webangebot nur von dem Betreiber selbst mit Inhalt befüllt und verwaltet wird. Wie in dieser Arbeit schon gezeigt, ist diese Sicht kritikwürdig, da etablierte Prinzipien nicht automatisch überflüssig werden, nur weil sie jemand mit der Bezeichnung „1.0“ herabwürdigt. Ein heutiges CMS sollte jedoch in der Tat die Möglichkeit mit einbeziehen, dass Inhalte von Benutzern beigesteuert werden können.

W2MS soll an dieser Stelle ohne ein vordefiniertes Schema auskommen, sei es in Bezug auf die

Struktur, als auch auf die Verwaltung und Anzeige der Inhalte. Der Schwerpunkt wird dabei klar auf inhaltsorientierte Webangebote gelegt. Zusätzlich soll einfache Funktionalität, die mit den Operationen Dateneingabe und -speicherung durchgeführt werden kann, möglich sein. Web*anwendungen*, also Webangebote mit funktionalem Schwerpunkt, sind nicht im Fokus von W2MS.

Die Benutzung von W2MS soll dabei auch für Nicht-Programmierer möglich sein. Da der Verzicht auf ein vordefiniertes Schema ein gewisses Abstraktionsvermögen verlangt, um eigene vage Vorstellungen in eine Struktur zu übersetzen, ist hier aber eher der „technisch interessierte“ Benutzer als Zielgruppe anzusehen, der vielleicht auch schon auf anderen Wegen versucht hat, ein Webangebot zu erstellen, dabei aber auf verschiedene Grenzen gestoßen ist. Eine weitere Zielgruppe sollen erfahrenere Benutzer sein, die im Prinzip fähig wären, ein Webangebot auch ohne Hilfsmittel zu erstellen. In diesem Fall würde der Nutzen von W2MS in der Vereinfachung und Qualitätsverbesserung des Entwicklungsprozesses bestehen. Die Zielgruppe von W2MS wird also im Bereich zwischen dem technisch begabten Nicht-Programmierer und dem semi-professionellen Entwickler angesiedelt. Diese Eingrenzung schließt jedoch den professionellen Entwickler nicht prinzipiell aus, da dieser eine Arbeitserleichterung sicher nicht ablehnen wird, solange die Funktionalität gewährleistet wird, die er benötigt. Im Kontext von Content-Management, dass ja auch ein Teil von W2MS ist, müssen als Unterzielgruppe auch die Bearbeiter von Inhalten mit einbezogen werden – für den CMS-Teil müssen auch technikferne Benutzer berücksichtigt werden.

In Bezug auf diese Zielgruppe wurde in Abschnitt 3.7 schon festgestellt, dass hier ein sehr informeller Entwicklungsstil herrscht. Im Mittelpunkt steht der Wunsch nach schnellen Ergebnissen, Usability und Accessibility werden nur selten bzw. nur oberflächlich überprüft, da dazu gesonderte Werkzeuge bemüht werden müssen. Die Verwendung von modellbasierten Ansätzen erscheint den Meisten als nicht zielführend, da dies einen zusätzlichen Lernaufwand bedeutet – ohne direkte Ergebnisse.

Die größten Probleme haben die Entwickler hier mit der Browserkompatibilität und mit der Umsetzung des gewünschten Layouts. Die Meisten würden sich einen WYSIWYG-Ansatz wünschen, der in ein WMS integriert ist. Besonders Nicht-Programmierer erwarten, dass Dinge wie Datenbank-, Session- und Rechteverwaltung ohne besonderes Zutun vorhanden sind. Entwickler wollen jedoch bei alledem immer das Gefühl der vollen Kontrolle über das System haben und möglichst viele Details ändern können.

In der Zielgruppe herrscht also durchaus ein Bedarf für Werkzeuge, die den Entwickler ganzheitlicher unterstützen. Außerdem ist ein aus Sicht des Web-Engineering ungenügender Entwicklungsstil festzustellen, dem mit Werkzeugen entsprechenden entgegengewirkt werden kann. Die Existenz solcher Werkzeuge ist auch im Hinblick auf die Qualität des Web im Allgemeinen von Bedeutung, da dadurch die Einhaltung von Standards und Qualitätskriterien auch für Webangebote von nicht-professionellen Entwicklern Wirklichkeit wird.

Die Vorteile von modellbasierten Ansätzen sollen so weit wie möglich in W2MS integriert werden, ohne den Benutzer all zu sehr in der Umsetzung seiner Ideen aufzuhalten. Jedenfalls muss die Trennung von Struktur, Inhalt, Navigation und Präsentation sichergestellt werden, damit die Flexibilität des Webangebots gewährleistet wird. Der zentrale Punkt ist hier, dass Inhalte nicht auf „Seiten“ abgelegt werden, sondern als Instanzen von Komponenten gespeichert werden. Ein „Inhalt“ ist also ein eigenständiges Objekt, das völlig unabhängig von „Seiten“, Navigation und Layout existiert. Komponenten können frei definiert, untereinander verknüpft und mit unterschiedlichen Ansichten (Views) versehen werden. Die Bearbeitung von Ansichten sollte sich dabei an WYSIWYG annähern. Komponenten können darüberhinaus mit Zugriffsrechten versehen werden – auf diese Weise soll z.B. auch definiert werden können, ob bestimmte Komponenten von Besuchern eines Webangebots „befüllt“ werden können. W2MS soll dabei

als Mehrbenutzersystem ausgelegt sein, indem es konsequent kollaborative Funktionen bereitstellt. So sollen Benutzer gleichzeitig Objekte bearbeiten und dabei auch miteinander kommunizieren können.

Der Benutzer sollte sich dabei nicht um die Verwaltung der Datenbank und die technische Umsetzung von Sessions sowie eines Rechtesystems kümmern müssen. Diese Funktionalitäten soll W2MS schon bereitstellen. Weiterhin soll das System automatische Tests durchführen, ohne dass der Benutzer dafür speziell in Aktion tritt – denn wahrscheinlich wird er das nicht tun.

Ein wichtiges Kriterium für W2MS ist schließlich, dass es schnell, intuitiv und zufriedenstellend benutzbar ist – in anderen Worten: es muss eine gute Usability haben. Dies ist ein entscheidender Faktor, der – wie zu Beginn dieses Abschnitts erläutert – darüber entscheidet, ob das Werkzeug letztlich auch genutzt wird. Hierzu gehört zunächst eine schnelle und übersichtliche Oberfläche, die dem Benutzer aber trotzdem das Gefühl der Kontrolle über die gesamte Funktionalität gibt. Dazu gehört auch, dass alle Vorgänge im System dem Benutzer verstehbar mitgeteilt werden, jedoch ohne dass es seinen Arbeitsfluß stört. Auf jede Interaktion sollte dabei eine Rückmeldung des Systems erfolgen, die den Benutzer über Erfolg oder Misserfolg in Kenntnis setzt. Die Navigation, sowie die Bedienelemente und -schritte sollten über das ganze System hinweg konsistent sein, um eine schnelle Erlernbarkeit, sowie einen hohen Erinnerungsfaktor zu erreichen. Alle Benutzeraktionen sollten leicht rückgängig gemacht werden können, um dem Benutzer so eine frustfreie Bedienung zu ermöglichen. Für alle Objekte, die im System auftauchen, sollte eine History angelegt werden, mit Hilfe derer man ein Objekt auf einen bestimmten Bearbeitungsstand in der Vergangenheit zurücksetzen kann.

W2MS ist also als ein browserbasiertes System anzusehen, dass den nicht- bis semiprofessionellen Benutzer ganzheitlich bei der Erstellung und Verwaltung eines Webangebots unterstützt. Der Benutzer kann frei definieren, aus welchen Komponenten sein Webangebot bestehen soll, wie diese zusammenhängen und wie sie präsentiert werden. Durch die Abdeckung des gesamten Lebenszyklus eines Webangebots kann das System vermehrt Aspekte des Web-Engineering integrieren und einen Mindestgrad an Flexibilität, Accessibility und Usability sicherstellen.

4.2 Grundkonzept - ein funktionaler Prototyp

Das Konzept von W2MS wurde in zwei Stufen entwickelt. Zunächst wurde an einer technischen Basis für die Umsetzung der beschriebenen Zielsetzungen gearbeitet. Darauf aufbauend wurde die oben schon angesprochene Komponentenarchitektur, sowie eine grundlegende Oberfläche nach den oben festgelegten Kriterien implementiert. Dieser erste Prototyp umfasst einen Teil der angesetzten Funktionalität und wurde in einem Benutzertest evaluiert. Anhand der gewonnenen Erkenntnisse während der Entwicklung des Prototypen und durch die Evaluierung wurde schließlich ein erweitertes Konzept entsprechend der Zielsetzung erstellt. Die zentralen Bereiche, sowie wichtige Funktionalitäten des Systems wurden auch grafisch in exemplarischen Programmansichten umgesetzt. Das erweiterte Konzept wird dann, abzüglich der in diesem Abschnitt schon behandelten Bereiche, in Abschnitt 4.3 beschrieben.

4.2.1 Technische Basis

Grundsätzlich wird für W2MS der Ansatz einer *single-page application* (kurz SPA, siehe 2.3.2) gewählt, um die gesetzten Anforderungen an die Oberfläche realisieren zu können. Die weiteren Vorteile und Nachteile dieses Ansatzes sind in 2.3.2 detailliert beschrieben. In der Konsequenz bedeutet das, dass ein großer Teil der Anwendungslogik auf dem Client in JavaScript implementiert wird. Auf dem Server kommt die freie Skriptsprache PHP, sowie die freie Datenbank MySQL zum Einsatz. Der Client-Code setzt auf den Frameworks Prototype [STEPHEN-

SON2006], sowie Dojo [DOJO2006] (Details jeweils unter 3.5.1) auf. Die Entscheidung für Dojo fiel vor allem wegen der hilfreichen Funktionen in Bezug auf die Browser-History, die das Framework anbietet.

Wie für eine SPA obligatorisch, kommuniziert der Client bei W2MS über XMLHttpRequest mit dem Server (vgl. 2.3). Um einen Mehrbenutzerbetrieb von W2MS zu gewährleisten, wurde jedoch außerdem der in 2.4 beschriebene Comet-Ansatz implementiert. Jede Kommunikation wird bei W2MS von der Klasse „Event“ gesteuert, die sowohl auf dem Client, als auch auf dem Server implementiert ist (vgl. Abbildung 4.1).

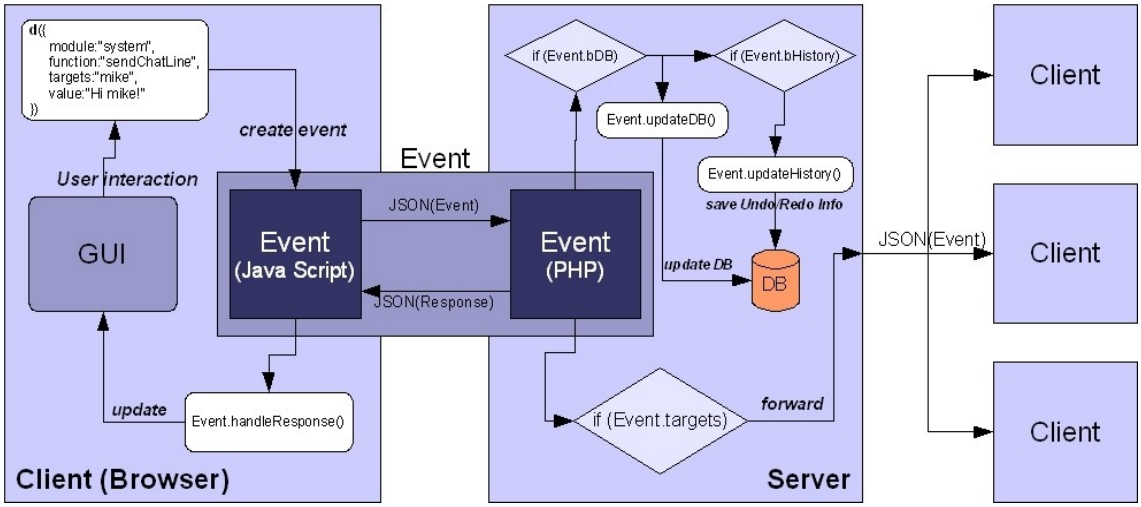


Abbildung 4.1: Event-gesteuerte Client/Server-Kommunikation in W2MS

Erfolgt beim Client eine Benutzerinteraktion, die eine Kommunikation zum Server erfordert, so wird daraus ein entsprechendes Event generiert und ausgeführt. Daraufhin wird es in das native JavaScript-Format JSON konvertiert, zum Server übertragen und dort wiederum in ein Event-Objekt umgewandelt und ausgeführt. Der Client erhält daraufhin die Antwort wiederum im JSON-Format. Dies geschieht durch den Aufruf der Callback-Funktion des Events. Die Klasse *Event* verbirgt also alle Netzwerkvorgänge und verhält sich daher ähnlich wie ein *Remote Procedure Call,* bzw. wie RMI von Java. Die Besonderheit hier ist jedoch, dass jedes Event auf dem Server in einer History mit Undo- und Redo-Informationen gespeichert werden kann. Über Dojo wird das ausgeführte Event an den Zurück-Button des Browsers gebunden. Dabei wird das Event-Attribut „historyStatus“ auf „undo“ gesetzt. Wird dieses Event nun durch die Betätigung des Zurück-Buttons ausgeführt, so wird anhand der Undo-Informationen in der History das Event rückgängig gemacht. Welche Aktionen dabei genau ausgeführt werden, kann für jedes Event separat definiert werden. Eine weitere Besonderheit ist die Weiterleitung von Events an weitere aktive Clients. Durch das Attribut „targets“ können einem Event beliebig viele Empfänger zugeordnet werden. Der Server leitet in diesem Fall das Event im JSON-Format an die Empfänger-Clients weiter. Diese müssen natürlich über Comet mit dem Server verbunden sein.

Die Comet-Implementierung von W2MS ist in Abbildung 4.2 dargestellt (einleitende Informationen zu Comet sind in 2.4 zu finden). Sobald ein Client W2MS startet, wird ein *long-loading* XMLHttpRequest zum Server aufgebaut – also eine Verbindung, die vom Server offen gehalten wird. Läuft nun beim Server ein Event für diesen Client auf, z.B. von einem weiteren Client, so wird dies im JSON-Format in die offene Verbindung geschrieben. Die Verbindung wird daraufhin vom Server beendet. Durch die Beendigung der Verbindung wird auf dem Client die Callback-Funktion des XMLHttpRequests ausgeführt und das Event kann entgegengenommen und ausgeführt werden. Daraufhin wird sofort wieder eine *long-loading* Verbindung zum Server aufgebaut, damit weitere Events geschickt werden können. Falls zwischen zwei *long-loading*

Verbindungen ein Event beim Server für den entsprechenden Client aufläuft, so wartet der Server, bis wieder eine Verbindung möglich ist.

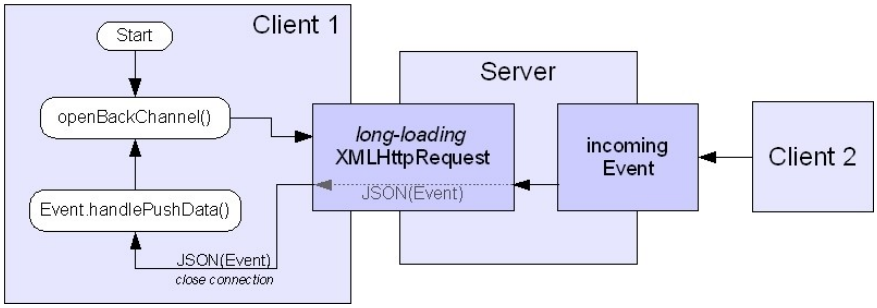


Abbildung 4.2: Comet-Implementierung von W2MS

Die Implementierung von Comet auf der Serverseite ist nicht ganz trivial. Wird ein PHP-Skript auf dem Server von einem Client aufgerufen, so kann die gestartete Skript-Instanz zwar mittels einer Endlosschleife leicht aktiv gehalten werden. Allerdings besteht keine triviale Möglichkeit auf diese laufende Skript-Instanz von einer anderen Instanz aus zuzugreifen. Um nun eine Kommunikationsmöglichkeit zwischen zwei Skript-Instanzen zu schaffen, wird von der Skript-Instanz der *long-loading* Verbindung ein Socket geöffnet, mit dem sich andere Instanzen verbinden können. Sobald in diesen Socket Daten geschrieben werden, wird die Skript-Instanz beendet und somit gleichzeitig der Abbruch der *long-loading* Verbindung herbeigeführt, was wiederum beim Client zur Folge hat, dass die Callback-Funktion mit den gewünschten Daten aufgerufen wird.

Comet wurde daneben noch in einer zweiten Variante implementiert, bei der die *long-loading* Verbindung nach dem Übertragen eines Events nicht abgebrochen wird. Dies hat allerdings zur Folge, dass auf dem Client in sehr kurzen Abständen (~10-50ms, je nach Anforderung) in den Datenstrom der Verbindung geschaut werden muss, um festzustellen, ob neue Daten vorliegen. Zudem funktioniert diese Variante nicht im Internet Explorer, da dieser den Zugriff auf den Datenstrom einer laufenden XMLHttpRequest-Verbindung nicht zulässt. Positiv ist bei dieser Variante, dass die Verbindung zum Client immer offen bleibt und keine Ausfallzeiten zwischen Abbau und Aufbau der Verbindung anfallen. Allerdings muss die Verbindung nach einer bestimmten Menge an übertragenen Daten auch hier neu aufgebaut werden, da sonst der auf dem Client aufgelaufene Datenstrom so umfangreich ist, dass dieser nicht mehr effizient abgefragt werden kann (wohlgemerkt alle 10ms). Jedenfalls wurde diese Variante bei W2MS als subjektiv langsamer empfunden. Daraus dürfen jedoch keine allgemeinen Schlüsse gezogen werden, da hier Implementierungsdetails und die Anforderungen der jeweiligen Anwendung eine große Rolle spielen.

4.2.2 Flexible Architektur

Ein zentraler Punkt von W2MS ist, dass es in seiner Struktur nicht festgelegt, bzw vordefiniert ist. Diese wird von dem Benutzer, bzw. Entwickler, durch das Erstellen von Komponenten definiert. Komponenten beschreiben die Art von Objekten, die in dem jeweiligen Webangebot vorkommen – vergleichbar mit Klassen in der Programmierung. Die Inhalte eines Webangebots sind dann die Instanzen dieser Klassen (also Komponenten) und werden hier daher auch als Inhalts*objekte* bezeichnet. Eine Komponente setzt sich zunächst aus Grundtypen wie Textzeile, Textblock, Datum, Bild oder Video zusammen. Dabei können diese Typen jeweils durch Parameter näher bestimmt werden. So kann man z.B. einen Textblock auf drei Abschnitte reduzieren, oder für ein Bild eine bestimmte Pixelbreite vorgeben. Zusätzlich zu den Grundtypen kann eine Komponente auch aus anderen Komponenten zusammengesetzt sein. Dies kann auf drei verschiedene Weisen geschehen:

- Bei der Definition einer Komponente kann die Struktur einer anderen Komponente **importiert** werden. So könnte z.B. eine Komponente „EnhancedImage“, die aus einem Bild sowie einem Beschreibungstext besteht, in verschiedenen Komponenten zum Einsatz kommen.

- Eine Komponente kann **Inhaltsverknüpfungen** zu einer anderen Komponente als Datenfeld enthalten. Dadurch kann z.B. eine Instanz von „Article“ eine Verknüpfung zu einer Instanz von „Author“ enthalten. Hier findet also eine Wiederverwendung von Inhalten statt, die einmal definiert und zentral geändert werden können.

- Einer Komponente können **Subkomponenten** zugeordnet werden. Ist z.B. „Comment“ eine Subkomponente von „Article“, so können jeder Instanz von „Article“ beliebig viele Instanzen von „Comment“ zugeordnet werden.

Für jede Komponente können beliebig vielen Ansichten definiert werden. Diese Ansichten können wiederum auf einer globalen Ansicht des Webangebots angeordnet werden. Beides ist in einer WYSIWYG-ähnlichen Ansicht möglich, auf der Elemente mit Drag&Drop verschoben werden können (dies ist nicht in dem funktionalen Prototyp implementiert, siehe hierzu 4.3). Der Benutzer erhält also freie Gestaltungsmöglichkeiten – allerdings nur auf der Komponentenebene und nicht bei Inhaltsobjekten.

Auf der Komponentenebene werden nicht nur Ansichten definiert, sondern ebenso Zugriffsrechte. Für jede Benutzergruppe können dabei die Rechte „Read“, „Insert“, „Edit“, „Delete“ und „Publish“ entweder aktiviert oder deaktiviert werden. Dies wird im Folgenden an drei Beispielen verdeutlicht:

- **Fall 1 – Artikel mit Kommentarfunktion:** Um die Komponente „Artikel“ mit einer Kommentarfunktion zu versehen, muss man die Komponente „Kommentar“ dort zunächst als Subkomponente eintragen. Im Allgemeinen wird man den Besuchern bei der Komponente „Artikel“ nur ein „Read“ erlauben. In der Komponente „Kommentar“ würde man jedoch zusätzlich noch ein „Insert“ zulassen. Abhängig davon, ob man „Publish“ ebenfalls zulässt, wären die Kommentare dann moderiert oder nicht. Unabhängig von Benutzergruppen können außerdem noch für den Besitzer eines Objekts die Rechte „Read“, „Edit“ und „Delete“ vergeben. Damit könnte man definieren, ob ein Besucher einen abgegebenen Kommentar später noch bearbeiten oder löschen darf.

- **Fall 2 – Anfrageformular:** Um ein Anfrageformular zu realisieren, würde man eine Komponente „Anfrage“ erstellen, mit entsprechenden Datenfeldern versehen und ggf. noch die Input-View anpassen. Im Allgemeinen will man hier, dass zwar jeder eine Anfrage eintragen kann, dass aber kein Besucher die Anfragen lesen kann. In diesem Fall würde man für die Gruppe der Besucher „Insert“ zulassen, aber „Read“ nicht erlauben.

- **Fall 3 – Erfahrungsberichte von registrierten Benutzern:** Will man ein Webangebot realisieren, in dem registrierte Benutzer Erfahrungsberichte abgeben können, so könnte man der Benutzergruppe der registrierten Benutzer für die Komponente „Erfahrungsbericht“ die Rechte „Read“, „Insert“ und „Publish“ zuweisen. Für die Benutzergruppe der Besucher würde allerdings nur das Recht „Read“ aktiviert.

Die technische Umsetzung von Komponenten wird von W2MS im Hintergrund automatisch erledigt. Das bedeutet, dass die Komponenten schon während ihrer Erstellung sofort benutzbar sind. Es erfolgt kein gesonderter Schritt, in dem die spezifizierten Komponenten mit ihren Feldern, Ansichten und Rechten in ein lauffähiges Webangebot umgewandelt werden müssten. Alle Vorgänge wirken sich direkt „live“ auf das Webangebot aus. Dabei kann beliebig zwischen Strukturdefinition, Ansichtsgestaltung, Rechteverwaltung und Inhaltsverwaltung hin- und hergesprungen werden.

Für jede Komponente wird in der Datenbank eine eigene Tabelle mit einer gesonderte Spalte für jedes Feld erzeugt. Jedes Inhaltsobjekt wird jedoch zusätzlich noch in einer globalen Tabelle eingetragen, damit es mit einer eindeutigen ID referenzierbar ist. Auf diese Weise können z.B. *tag clouds*, die über die Komponentengrenze hinausgehen, viel einfacher realisiert werden. Zudem werden in dieser Tabelle auch Daten gespeichert, die unabhängig von der Komponente von Bedeutung sind, wie z.B. eine Referenz auf den Besitzer des Objekts und diverse Timestamps. Die Komponentendefinitionen speichert W2MS als serialisierte Objekte in einer eigenen Tabelle.

Die Rechteverwaltung ist in W2MS mittels einer *access control list* in einer eigenen Datenbanktabelle umgesetzt. Für jedes Paar von Benutzergruppe und Komponente kann dort eine Zeile angelegt werden. Falls ein solches Paar nicht angelegt ist, werden dafür alle Rechte als „nicht erlaubt“ interpretiert. Es ist also alles verboten, solange es nicht explizit mittels der *access control list* zugelassen wird.

Wie oben schon angedeutet, kann in W2MS jedes Objekt mit *tags* versehen werden. Darüber hinaus besteht die Möglichkeit, dass jeder Besucher eine Bewertung für ein Objekt abgeben kann (dies kann in der Komponentendefinition deaktiviert werden). Da diese Funktionen jeweils über die Komponentengrenze hinausgehen, werden diesbezügliche Daten auch in eigenen Tabellen gespeichert.

Wie in dieser Arbeit schon gezeigt, ist eine ordentliche Trennung von Funktion, Struktur, Inhalt, und Präsentation unabdingbar, will man ein stabiles, erweiterbares und änderbares Webangebot erhalten. Es wurde jedoch außerdem der Wunsch nach möglichst viel Freiheit und WYSIWYG-Ansätzen bei den Entwicklern deutlich. Diese beiden Seiten zusammen zu bringen ist ein wichtiges Ziel von W2MS. Dadurch, dass das ganze System auf eine Komponentenarchitektur aufbaut, werden einige große Schwächen von *reinen* WYSIWYG-Ansätzen von vornherein verhindert, da Struktur, Inhalt und Layout getrennt definiert und somit auch zentral geändert und wiederverwendet werden können.

4.2.3 Oberfläche

Wie schon in 4.2.1 beschrieben, ist W2MS als SPA umgesetzt. Nach einer Benutzerinterkation erfolgt also kein Neuladen der Browserseite, sondern die Kommunikation läuft – entkoppelt von der Oberfläche – im Hintergrund ab. Das bedeutet jedoch auch, dass bei jeder Benutzerinteraktion explizit dafür gesorgt werden muss, dass auf der Oberfläche eine der Interaktion entsprechende Rückmeldung angezeigt wird. Der Benutzer sollte niemals im Unsicheren darüber gelassen werden, ob ein Vorgang wirklich durchgeführt wurde, bzw. ob Daten wirklich gespeichert wurden. In W2MS sind mehrere Features integriert, die genau das leisten sollen und dem Benutzer damit das Gefühl der Kontrolle über die Anwendung geben sollen.

Abbildung 4.3: Aktivitätsanzeige in W2MS

Die grundlegendste Rückmeldung, die dem Benutzer gegeben wird, ist die Aktivitätsanzeige in der oberen rechten Ecke der Oberfläche (Abbildung 4.3). Diese zeigt dem Benutzer immer an, wenn gerade eine Verbindung aktiv ist. Dies kann jedoch systemintern für bestimmte Verbindungen deaktiviert werden, die für den Benutzer nicht relevant sind, wie z.B. die Comet-Verbindung zum Server. Hier würde eine Aktivitätsanzeige nur verwirren. Die hier verwendete Animation von einem sich drehenden Kreis ist auch aus anderen Anwendungen bekannt und

wird daher leicht mit der gewünschten Bedeutung assoziiert. Bei Benutzeraktionen, die einen Wechsel in der Programmansicht zur Folge haben, kann die Oberfläche zusätzlich für die Dauer der Aktivität blockiert und ausgegraut werden. So wird verhindert, dass der Benutzer Interaktionen tätigt, die nicht mehr korrekt verarbeitet werden können.

Ein gängiges Prinzip in herkömmlichen Webanwendungen ist die Verwendung von umfangreichen Formularen, die mittels eines „Speichern"-Buttons zum Server übertragen werden. Dies liegt hauptsächlich darin begründet, dass man hier zur Übertragung/Speicherung von Daten ein Neuladen der Browserseite in Kauf nehmen muss und daher möglichst viele Daten auf einmal übertragen will. Bei einer SPA sind diese Art von Formularen, inklusive des „Speichern"-Buttons, nicht mehr nötig. Wird bei W2MS vom Benutzer irgendwo eine Veränderung vorgenommen, so wird diese sofort zum Server übertragen. Hier ist es wiederum wichtig, dass der Benutzer sich sicher sein kann, dass die Speicherung wirklich vorgenommen wurde. Hierzu wird bei W2MS in der unteren rechten Ecke für eine Sekunde die Meldung „saved" eingeblendet (Abbildung 4.4). Diese Meldung ist außerdem mit einem grünen Hintergrund versehen, so dass der Benutzer die Bedeutung selbst dann noch versteht, wenn er die Meldung nicht fokussiert. Es konnte in Benutzertests beobachtet werden, dass die Meldung zwar sehr schnell nicht mehr bewusst wahrgenommen wird, dass der gewünschte Effekt dabei aber trotzdem noch erhalten bleibt.

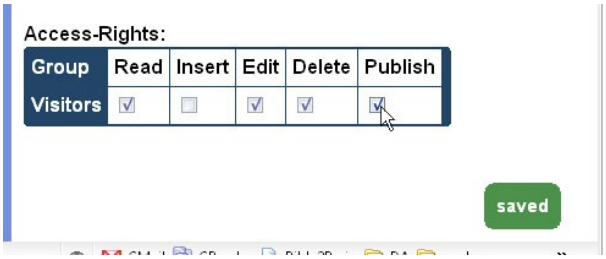


Abbildung 4.4: Rückmeldung über erfolgte Speicherung bei W2MS

Wie schon in 4.2.1 erläutert, ist W2MS ein Mehrbenutzersystem, bei dem auch Events, die nicht direkt von dem Benutzer selbst ausgehen, auf der Oberfläche angezeigt werden müssen. Dies kann für den Benutzer eine sehr verwirrende Angelegenheit werden und im schlimmsten Fall kann er ein Gefühl des Kontrollverlusts bekommen. In W2MS werden daher dezente Benachrichtigungen eingesetzt, die den Benutzer zwar nicht in seinem Arbeitsfluss stören, ihn aber doch darüber in Kenntnis setzen, was gerade passiert (ist). Diese Benachrichtigungen werden als transparente Box in der Mitte des oberen Drittels der Oberfläche – also unübersehbar – angezeigt (Abbildung 4.5).

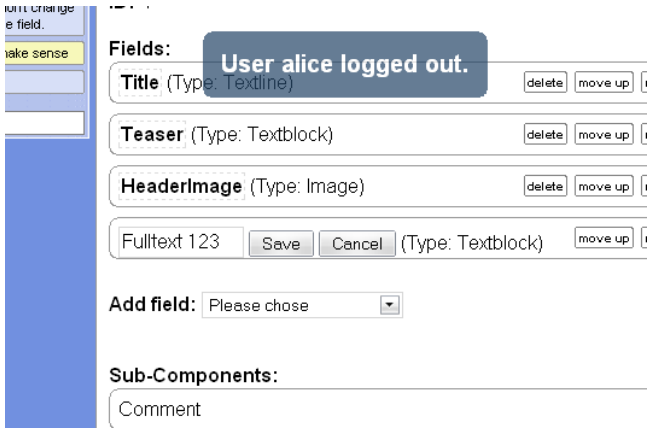


Abbildung 4.5: Benachrichtigungs-Einblendung in W2MS

Durch die Transparenz verliert der Benutzer nicht die Sicht auf seine Arbeitsfläche und die Box wirkt auch nicht so aufdringlich. Das Besondere hier ist außerdem, dass die Benachrichtigung sofort verschwindet, sobald der Benutzer die Maus bewegt oder irgendeine Taste drückt. Dadurch wird der Arbeitsfluss des Benutzers nicht unterbrochen. Erfolgt keine Interaktion, bleibt die Benachrichtigung stehen. Dadurch ist sichergestellt, dass der Benutzer die Nachricht nicht verpasst, falls er gerade nicht am Rechner sitzt. Durch nicht zu schnelles Ein- und Ausblenden wird außerdem eine zu große Ablenkung des Benutzers, die durch abrupte Veränderungen an der Oberfläche entstehen könnte, verhindert.

Ungeachtet dessen ist die gleichzeitige Interaktion zwischen verschiedenen Benutzern innerhalb einer Webanwendung eine noch recht ungewöhnliche Praxis und stellt daher eine potentielle Problemquelle dar. Die Tatsache, bzw. das Problem, dass verschiedene Benutzer (zufällig) zur gleichen Zeit an einem gleichen Objekt arbeiten könnten, besteht jedoch schon immer – nur bedeutete das, dass dabei fast unvermeidlich Inkonsistenzen, bzw. Überschreibungen von Änderungen auftreten. Der Ansatz, dass Benutzer miteinander beim Bearbeiten eines Objektes interagieren können, bringt zwar neue Schwierigkeiten mit sich, ist aber ein Schritt in die richtige Richtung. W2MS bezieht diese Mehrbenutzersicht von Grund auf mit ein. Wenn ein Benutzer eine Änderung vornimmt, wird sofort bei allen Benutzern, die sich auf der gleichen Ansicht der Anwendung befinden, die Oberfläche entsprechend angepasst. In einer Seitenleiste auf der linken Seite werden zum einen alle Benutzer angezeigt, die momentan gleichzeitig die Anwendung benutzen, und zum anderen die Benutzer, die sich gerade auf der gleichen Ansicht der Anwendung befinden (Abbildung 4.6). Dadurch soll das Bewusstsein dafür hervorgerufen werden, dass andere Benutzer „anwesend“ sind.

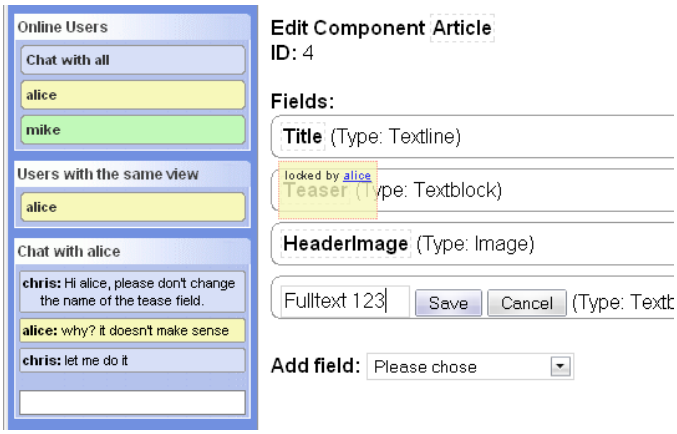


Abbildung 4.6: Gleichzeitige Bearbeitung einer Komponente in W2MS

Durch einen Klick auf den Namen eines Benutzers kann jederzeit ein Chat begonnen werden, so dass eine unkomplizierte Kommunikation untereinander möglich ist. Jedem Benutzer wird dabei eine eigene Farbe zugewiesen, mit der alle Aktionen, die von ihm ausgehen, auf der Oberfläche gekennzeichnet werden. So ist bei einer Änderung auf der Oberfläche immer auf einen Blick ersichtlich, dass diese von einem bestimmten Benutzer herbeigeführt wurde.

Wie in Abbildung zu sehen ist, wird in W2MS ein Textobjekt gesperrt, sobald ein anderer Benutzer dieses bearbeitet. Wie man sehen kann, können jedoch andere Textobjekte, die sich auf der gleichen Ansicht befinden, trotzdem bearbeitet werden. In der hier gezeigten Abbildung bearbeiten also zwei Benutzer gleichzeitig die gleiche Komponente und können sich dabei gegenseitig zuschauen. Sperren werden dabei nur auf einer relativ feinen Granularitätsebene eingesetzt. Die gleichzeitige Bearbeitung des gleichen Texts ist jedoch eine sehr komplexe Angelegenheit und sehr fehleranfällig, da die Synchronisation von Änderungen hier ein nicht

triviales Problem darstellt. Der Mittelweg mit feinen Sperren erschien hier als gute Lösung; für eine gute Zusammenfassung zum Thema Echtzeit-Synchronisation von Texten sei an dieser Stelle auf [FRASER2006] verwiesen.

Weiter oben wurden schon die umfangreichen Formulare aus herkömmlichen Webanwendungen angesprochen. In diesen Formularen wird in der Regel ein komplettes Objekt mit allen möglichen Attributen und Einstellungen in den Bearbeitungszustand versetzt. W2MS kennt so einen globalen Bearbeitungszustand nicht. Jeder Wert, der änderbar ist, lässt sich bei W2MS „in-place“ bearbeiten. Textobjekte, die änderbar sind, sind bei W2MS durch die Umrandung mit einer gestrichelten Linie gekennzeichnet. Durch einen Klick auf den Text lässt sich das Textobjekt dann an der selben Stelle ändern, ohne dass dies Auswirkungen auf die umgebenden Elemente auf der Oberfläche hätte (Abbildung 4.7).

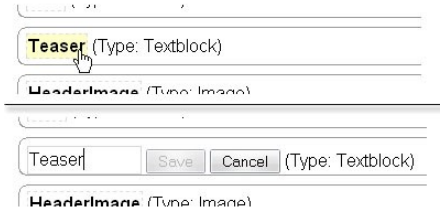


Abbildung 4.7: "edit-in-place" in W2MS

In 2.3.4 wurde schon gezeigt, dass – entgegen der allgemeinen Meinung – mit AJAX eine viel sinnvollere Verwendung der Vor- und Zurück-Buttons im Kontext von Webanwendungen möglich ist. Wie außerdem in 4.2.1 erläutert, lässt sich in W2MS jedes Event durch die Betätigung des Zurück-Buttons rückgängig machen und analog durch die Betätigung des Vorwärts-Buttons wiederholen. W2MS bindet also die Browsernavigation sinnvoll in die Bedienung der Oberfläche mit ein. Für den Benutzer ist allerdings nicht immer eindeutig klar, welchen Effekt die Betätigung des Zurück-Buttons hat. Wenn er z.B. von der Komponentenübersicht in die Einzelansicht einer Komponente wechselt und dort ein Textobjekt ändert, so wären zwei Funktionsweisen des Zurück-Buttons denkbar: Zum einen könnte zur vorherigen Ansicht gewechselt werden, oder es könnte die Änderung an dem Textobjekt rückgängig gemacht werden. Da in W2MS jede Aktion rückgängig gemacht werden kann, ist hier letztere Variante der Standardfall. Um ungewollten Effekten vorzubeugen, wird dieses Problem jedoch in W2MS abgefangen, bevor es für Irritationen sorgen kann. Für den Fall, dass durch die Betätigung des Zurück-Buttons der Datenbankzustand geändert wird, wird der Benutzer durch eine Meldung darauf hingewiesen (Abbildung 4.8). An dieser Stelle hat er die Möglichkeit mit der Aktion fortzufahren oder die andere Variante zu wählen, nämlich die Rückkehr zur letzten Ansicht.

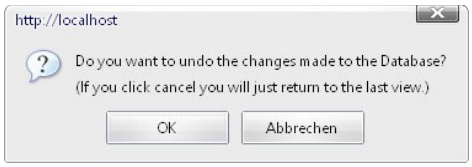


Abbildung 4.8: Warnmeldung bei Betätigung des Zurück-Buttons in W2MS

4.2.4 Evaluierung

Mit dem oben vorgestellten Prototypen von W2MS wurde ein Benutzertest durchgeführt, um Stärken und Schwächen auszumachen und weitere Anregungen für die Erweiterung des Konzepts zu bekommen. An dem Test nahmen fünf Personen teil, die alle der Gruppe der Nicht-Pro-

grammierer zugeordnet werden können. Vier der Personen können als *technisch interessiert* eingestuft werden, da sie aus technischen Studiengängen, bzw. Berufen kamen. Eine Person kam aus einem sozialen Studiengang und zeigte sich auch als technisch nicht interessiert.

Der Testaufbau bestand aus drei vernetzten Rechnern, auf denen W2MS in einem Firefox-Browser gestartet wurde. Auf einem der Rechner liefen Webserver und Datenbank für den serverseitigen Teil von W2MS. Bei allen Tests wirkte ein geschulter Mitarbeiter mit, der sich von einem der Rechner aus ebenfalls als Benutzer in W2MS einloggte, um bewusst bestimmte Events starten zu können, damit so die Reaktion der Teilnehmer gezielt untersucht werden konnte. Den Teilnehmern wurde zu Beginn gesagt, dass es sich bei W2MS um einen Prototypen von einem Werkzeug handelt, mit dem man beliebige Webangebote erstellen kann. Die Teilnehmer wurden gebeten, während des Tests laut zu denken, also zu kommentieren, warum sie etwas tun und mitzuteilen, wo sie nicht weiterkommen.

Den Teilnehmern wurde die Aufgabe gegeben, mit W2MS die Struktur eines Webangebots aufzubauen, das aus Artikeln besteht, die kommentierbar sind und in Kategorien eingeteilt werden können. Letzterer Teil der Anforderung wurde dabei zunächst verschwiegen und nach Erledigung der anderen Anforderungen hinzugefügt. Während der Bedienung wurden von dem Mitarbeiter sporadisch Events erzeugt, wie z.B. das Ein- und Ausloggen eines Benutzers, das Senden einer Chat-Nachricht, sowie die Bearbeitung eines Objekts, an dem gleichzeitig auch die Testteilnehmer arbeiteten. In einem Testlauf wurde die Aufgabe von zwei Teilnehmern gleichzeitig ausgeführt, drei weitere Testläufe wurden als Einzeltests durchgeführt – so konnte genauer auf die Einzelperson eingegangen werden. Der Mitarbeiter wirkte in jedem Testlauf als ein zusätzlicher Benutzer von W2MS mit.

Abstraktionsprobleme

Es war zu beobachten, dass alle Teilnehmer zunächst Schwierigkeiten hatten, die informelle Aufgabenstellung in das Konzept von Komponenten zu transferieren. Da man in W2MS jedoch direkt nach der Neuinstallation nichts anderes tun kann, als eine neue (d.h. erste) Komponente zu erzeugen, wurde diese Aktion natürlich auch von allen wie gewünscht durchgeführt. Drei der Teilnehmer versuchten darauf hin, *Inhalte* in die Komponente einzufügen, da sie die gerade erzeugte Komponente schon als Inhaltsobjekt verstanden und nicht als Container für Inhalte. So verstanden sie das Hinzufügen von neuen Datenfeldern als Möglichkeit zur Erweiterung des Inhaltsobjekts. Die editierbare Bezeichnung eines Feldes wurde dabei benutzt, um konkrete Inhalte einzugeben. Zwei Teilnehmer verstanden hingegen nach einer kurzen Orientierung von sich aus, dass es bei einer Komponente nur um die Struktur von Inhalten geht. Nach einer Wiederholung der Aufgabenstellung mit einer Betonung auf dem Wort „Struktur“, waren schließlich zwei weitere Teilnehmer in der Lage, Struktur und Inhalt gedanklich zu trennen. Ein Teilnehmer hatte dabei immer noch große Schwierigkeiten, seine Vorstellungen in eine Struktur zu abstrahieren, bzw. sah die Notwendigkeit dafür nicht. Erwartungsgemäß handelte es sich hier um den technisch uninteressierten Teilnehmer. In Bezug auf diese Benutzergruppe wurde in 4.1 schon dieses Ergebnis vermutet.

Ein Teilnehmer fand sich besonders schnell in das Definieren der Komponenten hinein. Es bereitete ihm zudem sichtbar Genugtuung, die Struktur eines Webangebots auf eine so freie Weise erstellen zu können. Es sei an dieser Stelle ein Zitat dieses Teilnehmers wiedergegeben, dass er während der Benutzung von W2MS geäußert hat: „Mit diesem Tool kann sogar ich Webangebote erstellen.“

Auffallend war, dass alle Teilnehmer sehr sicher in der Vergabe von Zugriffsrechten auf eine Komponente waren. Der Benutzergruppe der „Besucher“ wurden dabei immer die richtigen Rechte für die Komponenten „Artikel“ und „Kommentar“ zugewiesen. Ein Teilnehmer wies richtigerweise darauf hin, dass es noch einer weiteren Einstellungsmöglichkeit bedarf, um für

den Besitzer eines Objekts gesonderte Rechte vergeben zu können. Dies ist z.B. von Bedeutung, wenn man dem Schreiber eines Kommentars das Recht geben will, seinen Kommentar nachträglich noch ändern zu können. Diese Möglichkeit sollte in W2MS noch integriert werden.

Es lässt sich an dieser Stelle festhalten, dass es nicht für jeden Benutzer leicht ersichtlich ist, was er mit einer „Komponente“ assoziieren soll. Außerdem sind es Benutzer gewöhnt, inhaltsorientiert zu arbeiten. Das Prinzip einer getrennten Bearbeitung von Struktur und Inhalt ist also für viele nicht sofort einleuchtend. Die Oberfläche muss also hier in der Hinsicht verbessert werden, dass für den Benutzer klar ersichtlich ist, was er gerade bearbeitet und warum.

Erlernbarkeit

Die Erweiterung der nach dem ersten Teil der Aufgabenstellung bestehenden Struktur um die Anforderung des letzten Teils der Aufgabe (Hinzufügen von Kategorien) wurde von den Teilnehmern nicht als besonderes Problem wahrgenommen. Das ist eine unscheinbare aber bedeutsame Beobachtung, da die Änderung oder Erweiterung des Aufbaus eines Webangebots unter normalen Umständen kein triviales Problem ist. Nachdem die Teilnehmer nun schon die beiden Komponenten „Artikel“ und „Kommentar“ erzeugt und verknüpft hatten, zum Teil auch mit Hilfen und Erklärungen, lag es für sie auf der Hand, dass die Integration von Kategorien „einfach“ durch das Hinzufügen einer neuen Komponente erledigt werden kann. Man kann also sagen, dass Benutzer schon nach einer kurzen Eingewöhnung mit W2MS in der Lage sind, die Struktur eines Webangebots aufzubauen, zu verändern und zu erweitern. Sobald die „Abstraktionshürde“ überwunden ist, machen sich auch dem Benutzer schnell die Vorteile einer gesonderten Definition der Struktur bemerkbar.

Wahrnehmung der Oberfläche

Benutzer sind es von den meisten Anwendungen gewöhnt, dass jede Änderung durch eine gesonderte Speichern-Aktion gesichert werden muss. Wie schon in 4.2.3 erläutert, gibt es dieses Prinzip in W2MS nicht. Von daher war es eine wichtige Frage, ob dies in den Teilnehmern ein Gefühl der Unsicherheit bewirkt. Es konnte zunächst festgestellt werden, dass keiner der Teilnehmer in Bezug auf diese Sache eine Unsicherheit gezeigt hat, indem er etwa nach einem Speichern-Button gesucht hätte, oder das Thema der Speicherung in einem Kommentar in irgendeiner Weise erwähnt hätte. Der kleine grüne Hinweis über die erfolgte Speicherung (Abbildung 4.4) wurde jedoch ebenfalls in keinem Kommentar erwähnt; es konnte außerdem keine Reaktion auf diesen Hinweis beobachtet werden. Schließlich wurden allen Benutzer gefragt, ob sie eine Unsicherheit darüber hätten, ob ihre Änderungen auch wirklich gespeichert werden, bzw. wurden. Dabei wurde deutlich, dass jeder der Teilnehmer sich sicher war, dass alle seine Änderungen auch gespeichert wurden. Einer der Teilnehmer konnte sich auf die Schnelle gar nicht erklären warum, von den anderen wurde geäußert, dass dies ja klar wäre, weil immer der grüne Hinweis erschienen ist. Es läßt sich hier also sagen, dass eine kleine unscheinbare Rückmeldung auf der Oberfläche ausgereicht hat, um allen Teilnehmern in Bezug auf die korrekte Speicherung ihrer Änderungen ein Gefühl der Sicherheit zu geben. Obwohl man den Arbeitsvorgang des Speicherns aus vielen anderen Web- und Desktop-Anwendungen gewöhnt ist, wurde er hier überhaupt nicht vermisst, es wurde sogar noch nicht einmal bewusst darüber nachgedacht.

„Edit-in-place“

Eine weitere interessante Beobachtung war, dass alle Teilnehmer die „edit-in-place“-Funktion von Texten intuitiv benutzten. Auf die Aufforderung hin, eine Komponente oder ein Feld umzubenennen, wurde ohne Zögern das umrandete Textobjekt angeklickt und bearbeitet. Obwohl dieses Verhalten in Webanwendungen noch nicht sehr oft zu finden ist, wurde es hier wie

selbstverständlich behandelt.

Benutzung der Vor- und Zurück-Buttons

Die Vor- und Zurück-Buttons des Browsers wurden in den Fällen intuitiv benutzt, wo es um den Wechsel einer Ansicht ging. Es lässt sich daraus schließen, dass die Teilnehmer hier erwarteten, dass die Browser-Navigation in einer SPA automatisch die gleiche Funktionalität hat wie bei anderen Webangeboten. Die Teilnehmer machten jedoch einen Unterschied zwischen Interaktionen, die einen Wechsel der Ansicht zur Folge hatten und Interaktionen, bei denen nur ein Wert innerhalb einer Ansicht geändert wurde. Ein Teilnehmer äußerte, dass er *nicht* erwarte, dass eine Betätigung des Zurück-Buttons eine Änderung rückgängig macht, da der Browser ja keinen Einfluss auf die Webanwendung habe. Der Wechsel zu der vorherigen Ansicht der Anwendung war für ihn jedoch selbstverständlich. Hier erwies es sich als gut, dass W2MS den Benutzer warnt, falls die Betätigung des Zurück-Buttons eine Rücksetzung seiner letzten Änderung zur Folge hätte. Auch wenn der Benutzer ab diesem Zeitpunkt zwar weiß, welchen Effekt die Browser-Navigation bei W2MS hat, bleibt doch die Frage, ob hier nicht eine grundsätzliche Änderung erfolgen sollte. Es scheint sinnvoller zu sein, nur die Interaktionen, die die Ansicht ändern, über die Browser-Navigation zu steuern. Änderungen innerhalb einer Ansicht könnten dann über gesonderte Buttons innerhalb der Anwendung rückgängig gemacht werden.

Eine generelle Möglichkeit, Änderungen rückgängig machen zu können, ist aber ungeachtet dessen trotzdem wichtig. Es konnte beobachtet werden, dass alle Teilnehmer ein gewisses „Trial & Error"-Verhalten zeigten, um W2MS kennen zu lernen.

Mehrbenutzerbetrieb

Eine weitere wichtige Fragestellung des Benutzertests war, wie die Teilnehmer mit dem Mehrbenutzerbetrieb von W2MS zurechtkommen würden. Es zeigte sich, dass alle Teilnehmer das von Instant-Messenger-Programmen gewohnte Nutzungsmuster auf W2MS übertrugen. Der Kasten „Online Users" in der Seitenleiste (vgl. Abbildung 4.6) wurde von allen richtigerweise als eine Auflistung der momentan eingeloggten Benutzer verstanden. Interessanterweise führte die Existenz des genannten Kastens dazu, dass die meisten Teilnehmer während der Benutzung von W2MS von sich aus einen Chat mit einem der eingeloggten Benutzer anfingen. Der Start eines Chats wurde dabei intuitiv durch das Klicken auf den Namen des jeweiligen Benutzers durchgeführt. Der Kasten „Users with the same view" (vgl. Abbildung 4.6) wurde ebenso von allen Teilnehmern richtig verstanden. Es lässt sich also sagen, dass bei den Teilnehmer ein Bewusstsein für die „Anwesenheit" anderen Benutzer zu beobachten war, sowohl auf die ganze Anwendung, als auch auf die aktuelle Ansicht bezogen.

Ein kritischer Punkt war jedoch erwartungsgemäß die gleichzeitige Bearbeitung einer Komponente von mehreren Benutzern. Die Sperrung eines Textobjekts wurde von den meisten Teilnehmern als ein von dem anderen Benutzer ausgehendes Zugriffsverbot interpretiert. Als Ursache hierfür erwies sich eine missverständliche Wortwahl. Die Sperre wird ja, wie in Abbildung 4.6 zu sehen, durch eine transparente Box in der Farbe des bearbeitenden Benutzers über dem entsprechenden Textobjekt angezeigt. Diese Box ist zudem mit dem Text „locked by *username*" versehen, mit dem man in der Tat auch ein Verbot assoziieren kann. Hier sollte ein klarerer Text verwendet werden.

Bei einem Besucher ließ sich außerdem ein leichter Unmut darüber feststellen, dass er nicht die volle Kontrolle über die Anwendung hat, wenn andere Benutzer gleichzeitig daran arbeiten. Er stellte dabei aber gleichzeitig positiv hervor, dass es immer ersichtlich ist, wer sich gerade auf der gleichen Ansicht befindet und dass man mit diesem Benutzer direkt durch einen Chat in Kontakt treten kann. Es sei an dieser Stelle nochmal darauf hingewiesen, dass gleichzeitige Be-

arbeitungen auch stattfinden, wenn ein System diese nicht explizit unterstützt. Allerdings treten in diesem Fall Inkonsistenzen oder Überschreibungen auf. Daher wird der hier verwendete Ansatz auch beibehalten. Unterstützt wird dies außerdem dadurch, dass andere Teilnehmer das Gefühl des „Zusammenseins“ und die Möglichkeit, auf einer Ansicht gemeinsam zu arbeiten, als spaßig und motivierend empfanden. Der Mehrbenutzerbetrieb bedeutete für einige Teilnehmer also ein deutliches Plus in der Qualität der Benutzungserfahrung.

4.3 Erweitertes Konzept

Das bis hierher beschriebene und in dem funktionalen Prototypen umgesetzte Konzept wurde um fehlende Funktionen erweitert und anhand der Benutzertests überarbeitet. Die folgenden Ausführungen sind dabei nun komplementär zu dem vorherigen Abschnitt zu verstehen. Das erweiterte Konzept wurde nicht implementiert, jedoch wurden die wichtigsten Ansichten von W2MS grafisch in Beispielansichten umgesetzt. Diese sind in Anhang B zu finden und werden in diesem Abschnitt an relevanten Stellen referenziert, bzw. in Ausschnitten angezeigt, um Details hervorzuheben.

4.3.1 Allgemeines

Die Interaktionen von Benutzern werden in W2MS in zwei Kategorien eingeteilt:

- Sobald eine Interaktion einen Wechsel der Ansicht der Anwendung zur Folge hat, handelt es sich um eine **Navigations-Interaktion.**
- **Objekt-Interaktionen** sind hingegen ausschließlich mit der Bearbeitung eines Objekts verbunden. Der Benutzer verbleibt dabei auf der gleichen Ansicht der Anwendung.

Auch wenn technisch alle Interaktionen gleich behandelt werden, so macht es für den Benutzer doch einen großen Unterschied, ob er ein Textfeld bearbeitet oder ober er von einer Programmansicht zur nächsten wechselt. Der Benutzer erwartet, dass letzere Interaktionen mittels der Browsernavigation gesteuert werden können, da er dies so von der Bedienung anderer Webangebote gewöhnt ist. Für Objekt-Interaktionen bedarf es allerdings einer gesonderten Steuerung, die in W2MS durch einen Undo-, sowie einen Redo-Button links oben auf der Oberfläche repräsentiert ist (Abbildung 4.9).

Abbildung 4.9: Undo- und Redo-Buttons in W2MS

Diese Art Buttons sind aus anderen Anwendungen gut bekannt und dürften daher vom Benutzer intuitiv benutzt werden. Sie beziehen sich immer auf die aktuelle Ansicht der Anwendungen und ermöglichen, dort gemachte Änderungen rückgängig zu machen, bzw. zu wiederholen.

4.3.2 Verwaltung von Komponenten

Das Herzstück von W2MS besteht in der Komponentenverwaltung. Auf der Übersichtsansicht (vgl. Abbildung B.1 im Anhang) werden alle angelegten Komponenten in einer Kurzdarstellung mit den wichtigsten Funktionen angezeigt. Diese Ansicht ist gleichzeitig die Startansicht von W2MS. Die Kurzdarstellung einer Komponente besteht aus folgenden Elementen (vgl. Abbildung 4.10):

- Die Anzahl neuer Inhaltsobjekte dieser Komponente seit dem letzten Einloggen des Benutzers. Die Zahl kann dabei angeklickt werden, was zu einer Liste der betreffenden Inhaltsobjekte führt.
- Die Gesamtanzahl der Inhaltsobjekte dieser Komponente.
- Eine Suchfunktion, bei der alle textuellen Datenfelder der Komponente durchsucht werden. Passende Inhaltsobjekte werden daraufhin in einer Liste angezeigt.
- Jeweils ein Button, um ein neues Inhaltsobjekt einzufügen, die Struktur der Komponente zu bearbeiten, bzw. die Ansichten der Komponente zu bearbeiten.

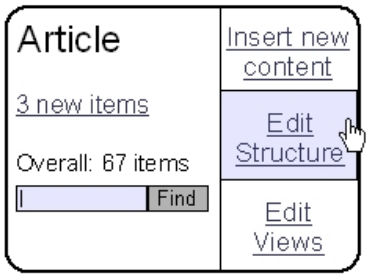


Abbildung 4.10: Kurzdarstellung einer Komponente in W2MS

Der Benutzer erhält hier also einen schnellen Überblick über den Inhalt des Webangebots und kann außerdem auch schnell auf bestimmte Inhaltsobjekte zugreifen. Ein weiterer wichtiger Aspekt ist, dass in der Kurzdarstellung der Komponenten die Aufteilung in Struktur, Ansichten und Inhalt schon enthalten ist. Neben dem bequemen und schnellen Zugriff auf die gewünschte Funktion, soll dadurch verhindert werden, dass der Benutzer versehentlich die Struktur bearbeitet, obwohl er eigentlich neuen Inhalt hinzufügen wollte.

4.3.3 Bearbeitung der Komponentenstruktur

Die Programmansicht bei der Bearbeitung der Komponentenstruktur ist in Abbildung B.2 im Anhang dargestellt. Eine Komponente kann aus folgenden Elementen zusammengesetzt werden, die jeweils durch Parameter näher bestimmt werden können:

- **Textzeile:** Eine Textzeile besteht aus textuellem Inhalt ohne Zeilenumbruch. Durch einen Parameter kann die Anzahl der Zeichen eingeschränkt werden. Für fortgeschrittene Anforderungen besteht die Möglichkeit, die Beschaffenheit des Textes durch einen regulären Ausdruck zu bestimmen.
- **Textblock:** Ein Textblock ist ein beliebiger Text. Durch Parameter kann hier die maximale Textlänge in Worten, sowie in Abschnitten angegeben werden.
- **Aufzählung:** Eine Aufzählung besteht aus einer vordefinierten Menge von Werten, die beim Einfügen oder Bearbeiten eines Inhaltsobjekts in einem Auswahlfeld ausgewählt werden können. Die einzelnen Werte werden an dieser Stelle angegeben.
- **Bild:** Hier kann optional eine fixe Breite oder Höhe des Bildes angegeben werden mit der Option, das Seitenverhältnis beizubehalten. Hochgeladene Bilder werden dann von W2MS automatisch in das gewünschte Format gebracht. In den Komponentenansichten kann die Bildgröße unabhängig davon für jede Ansicht gesondert angegeben werden.
- **Audio:** Als optionale Parameter können hier Bitrate und Frequenz angegeben werden.

W2MS konvertiert die Datei dann automatisch in eine MP3-Datei mit der gewünschten Qualität.

- **Video:** Als optionale Parameter können hier Bitrate, Auflösung und Bildrate des Videoteils, sowie Bitrate und Frequenz des Audioteils angegeben werden. W2MS wandelt die Videodatei dann in eine entsprechende FLV-Datei. Flash-Video ist hier das Format der Wahl, da es auf Browsern mit Abstand die meiste Verbreitung im Vergleich zu den anderen Videoformaten hat. W2MS stellt hierfür auch eine entsprechende Abspielkomponente bereit.
- **Link:** Ein Link kann hier optional nur auf die Domain des Webangebots beschränkt werden.
- **Verknüpfung zu einer anderen Komponente:** Wird eine neues Inhaltsobjekt erzeugt, so kann bei diesem Feld ein bestehendes Inhaltsobjekt der angegebenen Komponente ausgewählt werden. Durch einen Parameter kann hier angegeben werden, welches Feld der anderen Komponente dabei als Auswahlwert für das Auswahlfeld dienen soll.
- **Einbindung einer anderen Komponente:** Wird eine neues Inhaltsobjekt erzeugt, so wird bei diesem Feld die Struktur der angegebenen Komponente eingebunden. So können einmal definierte Strukturen in mehreren Komponenten wiederverwendet werden. Einzelne Felder der angegebenen Komponente können hierbei deaktiviert werden.

Durch die Verknüpfungsmöglichkeit bietet W2MS die Möglichkeit, ein Inhaltsobjekt in weiteren Inhaltsobjekten wiederzuverwenden. Auf diese Weise können mehrere Inhaltsobjekte von verschiedenen Komponenten ein gemeinsames Inhaltselement enthalten, das zentral geändert werden kann.

Zusätzlich besteht die Möglichkeit einer Komponente Subkomponenten hinzuzufügen. Diese haben – im Unterschied zu einer einfachen Verknüpfung zu einer anderen Komponente – den Zweck, einer Instanz der Komponente beliebig viele Instanzen der Subkomponente zuweisen zu können. In der Praxis kommt dies oft vor, so auch bei dem nun schon mehrfach angeführten Beispiel von Artikeln und Kommentaren. Beim Hinzufügen einer Subkomponente, lassen sich alle im System angelegten Komponenten auswählen – auch die, die gerade bearbeitet wird. Ein Beispiel hierfür wären Kategorien, die wiederum weitere (Unter-)Kategorien enthalten.

Die Reihenfolge der Felder und Subkomponenten kann auf der Oberfläche per Drag&Drop verändert werden. Dies hat allerdings keine Auswirkungen auf das Layout des Webangebots (dafür sind die Ansichten zuständig). Bei der Bearbeitung von Inhalten, bzw. bei der Inhaltssuche wird die Reihenfolge der Felder jedoch berücksichtigt.

W2MS bietet die Möglichkeit, auf die Inhaltsobjekte einer Komponente durch einen RSS-Feed zuzugreifen. In dem Kasten „Component-Properties“ auf der Seitenleiste lässt sich der RSS-Feed für die jeweilige Komponente konfigurieren. Entscheidend ist dabei, den Elementen des RSS-Feeds die passenden Felder der Komponente zuzuweisen. Der Zugriff auf den RSS-Feed einer Komponente geschieht durch Links, die in die globalen Ansichten des Webangebots integriert werden können. Durch Aktivierung der Checkbox „Integrate feedlink into HTML-Source“, kann W2MS außerdem dazu veranlasst werden, einen maschinenlesbare Verweis auf den jeweiligen Feed in den HTML-Code zu integrieren. Dieser wird von Browsern automatisch erkannt.

4.3.4 Verwaltung von Zugriffsrechten

Die Einstellung der Zugriffsrechte von Komponenten hat einen entscheidenden Einfluss auf die Funktionsweise des Webangebots. Dennoch bedarf es hierzu nur einer recht einfachen Ansicht (vgl. Abbildung B.3 im Anhang). Das Prinzip von Zugriffsrechten an sich wurde in 4.2 schon

näher erläutert. Hinzu kommt hier, dass dem Besitzer eines Inhaltsobjekts der betreffenden Komponente gesonderte Rechte zugewiesen werden können. Als Besitzer wird derjenige bezeichnet, der ein Inhaltsobjekt erzeugt hat. Die zuweisbaren Rechte sind hier „Read“, „Edit“ und „Delete“ und beziehen sich nur auf die von dem jeweiligen Benutzer erzeugten Inhaltsobjekte. Es handelt sich also um Rechte von einzelnen Benutzern auf bestimmten Inhaltsobjekten – dies steht im Gegensatz zu den anderen Rechten, die sich auf Benutzer*gruppen* und Komponenten beziehen.

4.3.5 Gestaltung von Komponentenansichten

Für eine Komponente können beliebig viele Ansichten (Views) definiert werden. Ansichten dienen dazu, die Inhaltsobjekte einer Komponente auf verschiedene Art und Weise präsentieren zu können. Für jede Ansicht kann dabei getrennt festgelegt werden, welche Felder der Komponente angezeigt, sowie wie diese angeordnet und formatiert werden sollen. Dabei können auch statische Daten wie Bilder und Texte in eine Ansicht integriert werden. In Abbildung B.4 im Anhang ist die Programmansicht zur Gestaltung von Komponentenansichten zu sehen.

Erstellt man eine neue Ansicht, so ist diese zunächst leer und muss mit Datenfeldern und sonstigen Elementen befüllt werden. Dazu gibt es am unteren Rand der Ansicht ein Auswahlfeld mit der Bezeichnung „Insert new item“. Hier gibt es folgende Auswahlmöglichkeiten:

- **Fields (Felder):** Hier wird ein weiteres Auswahlfeld mit allen Feldern der Komponente angezeigt. Klickt man auf eines dieser Felder, so wird es der Ansicht hinzugefügt und ist im Layout-Bereich zu sehen.
- **Subcomponents (Subkomponenten):** Hier wird zunächst ein weiteres Auswahlfeld mit allen Subkomponenten der Komponente angezeigt. Wählt man eine davon aus, kann man in einem weiteren Auswahlfeld zwischen verschiedenen Möglichkeiten wählen. Man kann zum einen eine Ansicht der Subkomponente auswählen, um die Anzeige von Einträgen auf der Ansicht zu erreichen. Dies schließt auch die Eingabeansichten mit ein, um ein Eingabeformular der Subkomponente in der Ansicht anzuzeigen. Zum Anderen lässt sich die Variable „Number of items“, also die Anzahl der Einträge der Subkomponente, auswählen. Zusammen mit einem statischen Text (etwa „Kommentare:“) ließe sich so eine Kurzinformation über die Subkomponente in die Ansicht integrieren.
- **View-Link (Ansichten-Link):** Es kommt oft vor, dass von der einen Ansicht einer Komponente zu einer anderen verlinkt wird, in der Regel, um von einer Übersichtsansicht zu einer Vollansicht eines Inhaltsobjekts zu wechseln. Hier wählt man also aus dem erscheinenden Auswahlfeld die gewünschte Ansicht aus und ein entsprechender Link wird mit dem Standardtext „more ...“ der Ansicht hinzugefügt. Dieser Text kann später im Layout-Bereich geändert werden.
- **Rating:** Soll ein Inhaltsobjekt bewertet werden können, so muss in einer der Ansichten dieses Rating-Element integriert werden. Dieses lässt sich über die „Item-Properties“ sowie über CSS anpassen.
- **Text:** Hier wird ein statischer Text der Ansicht hinzugefügt. Dieser kann dann im Layout-Bereich in einem Richtext-Editor bearbeitet werden.
- **Image:** Hier wird ein Dialog zum Hochladen einer Datei sowie ein Textfeld für den Alternativtext des Bildes angezeigt. Wird beides korrekt ausgefüllt, so wird das Bild der Ansicht hinzugefügt.

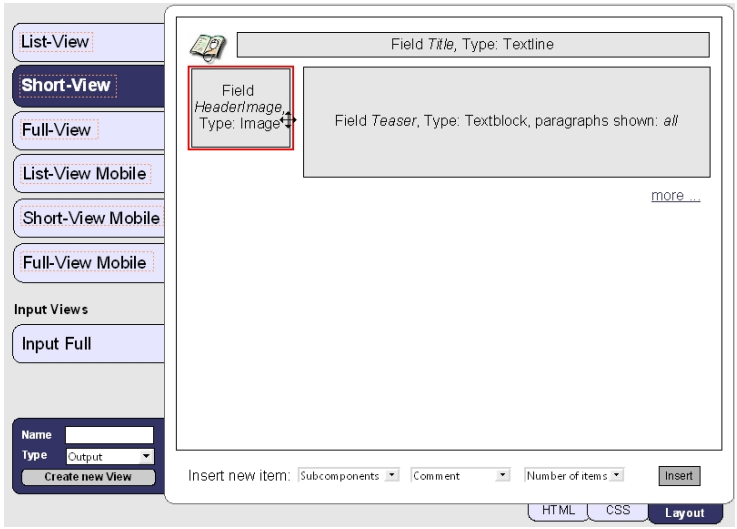


Abbildung 4.11: Gestaltung von Ansichten einer Komponente in W2MS

Hat man nun einige Elemente der Ansicht hinzugefügt, so sind sie im Layout-Bereich sichtbar und können dort mit Drag&Drop verschoben werden (vgl. Abbildung 4.11). Dabei kann ein Element rechts, links, überhalb oder unterhalb jedes anderen Elements positioniert werden. Standardmäßig erstrecken sich alle Elemente, die nicht von sich aus eine feste Breite haben, über die ganze Breite der Ansicht. Werden zwei Elemente nebeneinander angeordnet, so wird der Platz gleichmäßig aufgeteilt.

Für jedes Element gibt es weitere Einstellungen, die über den Kasten „Item-Properties“ in der Seitenleiste (Abbildung 4.12) verändert werden können. So kann für ein Bild-Feld der Komponente Breite und Höhe angegeben werden, oder für ein Text-Feld die Anzahl der Abschnitte oder Wörter, die davon angezeigt werden sollen. Bei eine Subkomponente lässt sich einstellen, wie viele Einträge angezeigt werden sollen und welche Ansicht der Subkomponente dabei verwendet werden soll.

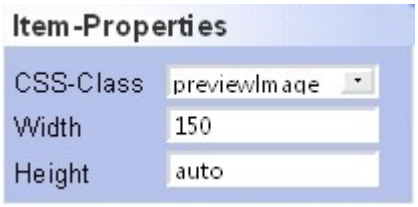


Abbildung 4.12: Eigenschaften eines Datenfeldes in einer Ansicht in der Seitenleiste von W2MS

Wenn es sich um eine stark verkürzte Ansicht handelt, die z.B. nur aus dem Titel eines Artikels besteht, möchte man in der Regel, dass die gesamte Ansicht ein Link auf die Vollansicht ist. Dies kann man auf der Seitenleiste in dem Kasten „View-Properties“ einstellen.

W2MS bietet zur Gestaltung der Ansicht noch einige fortgeschrittenere Möglichkeiten an. So gibt es zunächst für jedes Element der Ansicht den Kasten „Item-CSS“ in der Seitenleiste. Hier können dem Element beliebige CSS-Attribute zugewiesen werden. So könnte man einem Bild-Feld das „float“-Attribut zuweisen, damit nachfolgender Text um das Bild herumfließt. Es lassen sich hier auch CSS-Klassen definieren, um etwa alle Links innerhalb eines Elements einzufärben.

Darüber hinaus besteht die Möglichkeit, den HTML- sowie CSS-Code der Ansicht direkt zu bearbeiten. Dazu klickt man auf den entsprechenden Karteireiter unten rechts.

Neben den normalen Ansichten, gibt es noch die Eingabeansichten (Input Views). Mit diesen Ansichten können Eingabeformulare für die betreffende Komponente erstellt und gestaltet werden. Dies ist von Bedeutung für Komponenten, die von den Besuchern des Webangebots befüllt werden können, wie es z.B. für Kommentare üblich ist.

4.3.6 Verwaltung von Inhalten

Die Verwaltung von Inhalten ist ein zentrales Problem von Webangeboten, zumindest von inhaltsorientierten Webangeboten, die ja im Fokus von W2MS stehen. Im Allgemeinen ist das der Bereich, der sich am häufigsten ändert und mit dem Benutzer am meisten in Berührung kommen.

Ein wichtiger und häufiger Arbeitsschritt ist dabei das Auffinden von bereits vorhandenen Inhalten, z.B. um diese bearbeiten zu können. Hier bietet W2MS eine Reihe von Möglichkeiten, um möglichst schnell zu dem gewünschten Inhaltsobjekt zu gelangen. Dies beginnt mit der Kurzdarstellung von Komponenten auf der Komponentenübersicht, bzw. Startseite von W2MS (Abbildung 4.10). Von hier aus können die wichtigsten Aktionen zum Auffinden eines Inhaltsobjekts ausgeführt werden, ohne dass dabei zusätzliche Navigationsschritte nötig wären. Neben dem Link auf eine Liste der neuesten Inhaltsobjekte kann von dort aus sofort eine Schnellsuche durchgeführt werden. Diese führt auf die Ansicht der Inhaltssuche, die in Abbildung B.5 im Anhang gezeigt ist. In den meisten Fällen wird die Schnellsuche schon zum Ziel führen, da sie alle textuellen Informationen der Komponente in die Suche mit einbezieht. Falls die Suchergebnisse jedoch nicht zufriedenstellend sind, kann der Suchdialog erweitert werden (vgl. Abbildung 4.13) und eine feinere Eingrenzung vorgenommen werden. Alle Interaktionen mit dem Suchdialog haben dabei sofortige Auswirkung auf die Ergebnisliste. Auf diese Weise kann man sich inkrementell dem gewünschten Ergebnis annähern und muss dabei nur so viele Schritte durchführen wie nötig sind, um ein bestimmtes Inhaltsobjekt in der Masse aufzufinden.

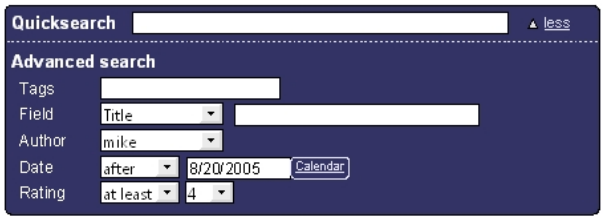


Abbildung 4.13: Suchdialog zum Auffinden von Inhaltsobjekten einer Komponente in W2MS

Ein entscheidender Arbeitsschritt ist schließlich natürlich das Einfügen, bzw. Bearbeiten eines Inhaltsobjekts. Die Ansicht bei der Bearbeitung eines Inhaltsobjekts ist in Abbildung B.6 im Anhang dargestellt. Jedes Feld der Komponente des Inhaltsobjekts wird dabei speziell bearbeitet, je nach dem von welchem Typ es ist. Handelt es sich bei einem Feld um eine eingebundene Komponente, so werden an dieser Stelle wiederum alle Felder dieser Komponente angezeigt. Handelt es sich um eine verknüpfte Komponente, so werden alle Inhaltsobjekte dieser Komponente in einem Auswahlfeld angezeigt. Dabei dient das in der Komponentenstruktur angegebene Feld als Auswahlwert. Bei einem Feld vom Typ Textblock, wird ein Richtext-Editor (im Folgenden RTE genannt) für die Bearbeitung verwendet.

Wenn mehrere Benutzer das gleiche Inhaltsobjekt bearbeiten, kann ein Feld immer nur von einem Benutzer gleichzeitig bearbeitet werden. Wie in 4.2.3 schon gezeigt, wird dieses dann für die anderen Benutzer für die Zeit der Bearbeitung (sichtbar) gesperrt. Felder mit textuellem In-

halt werden daher zunächst nicht als Eingabefeld angezeigt, sondern müssen durch einen einfachen Klick aktiviert werden. Für den Benutzer bedeutet dies jedoch keinen höheren Navigationsaufwand, da ein Eingabefeld auch erst durch einen Klick aktiviert werden muss, damit es den Fokus für die Tastatureingabe bekommt. Klickt der Benutzer schließlich auf „Save“, so ist das Feld wieder frei zur Bearbeitung für andere Benutzer.

W2MS überwacht die Inhalte schon während der Eingabe auf Fehler oder Probleme und meldet diese umgehend dem Benutzer. Versieht der Benutzer im RTE einen Text mit einem Link, so überprüft W2MS sofort im Hintergrund, ob dieser erreichbar ist. Wenn nicht, wird unter dem Absatz eine Warnmeldung angezeigt (Abbildung 4.14).

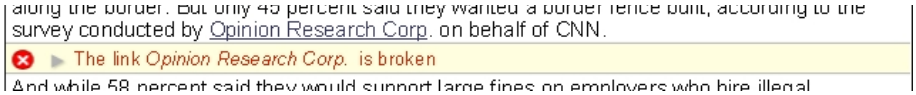


Abbildung 4.14: Hinweis auf einen ungültigen Link im Richtext-Editor in W2MS

W2MS meldet außerdem, wenn die Eingabe des Benutzers nicht den in der Komponentendefinition angegebenen Parametern des entsprechenden Feldes entspricht. Wenn ein Textblock auf einen einzigen Abschnitt beschränkt ist, der Benutzer aber mehr als einen Abschnitt im RTE eingibt, so erscheint unter dem Feld eine entsprechende Warnmeldung und das Speichern des Feldes wird verhindert. W2MS überprüft außerdem, ob ein Bild mit einem Alternativtext versehen ist. Wird dieser nicht eingegeben, so verweigert W2MS hier ebenso das Speichern und zeigt unter dem Feld eine erklärende Meldung an (vgl. jeweils Abbildung B.6).

Mit W2MS ist die Zurücksetzung eines Inhaltsobjekts auf einen früheren Stand möglich. In der Seitenleiste wird dazu der Kasten „Content-History“ angezeigt (Abbildung 4.15), in der alle Versionen des Inhaltsobjekts aufgelistet sind. Durch einen Klick auf den Link „revert“ werden alle Änderungen bis zu der betreffenden Version rückgängig gemacht. Hierbei führt jedoch nicht jede einzelne Änderung zu einer neuen Version. Werden von einem Benutzer hintereinander mehrere Änderungen in kurzem Abstand durchgeführt, so werden diese in einer Version zusammengefasst.

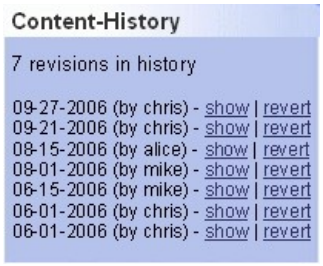


Abbildung 4.15: History eines Inhaltsobjekts in der Seitenleiste von W2MS

4.3.7 Bearbeiten von Menüs

W2MS ermöglicht die Erstellung von beliebig vielen Menüs, die dann in den globalen Ansichten des Webangebots eingebunden werden können. Menüs sind Zusammenstellungen von Links auf Inhalte, Komponenten oder Funktionen des Webangebots. Die Menüs in W2MS ermöglichen dabei sowohl eine statische als auch eine dynamische Navigation. Abbildung B.7 im Anhang zeigt die Ansicht bei der Bearbeitung eines Menüs.

Folgenden Arten von Einträgen können in Menüs enthalten sein:

- **System-Links** sind die Links, die nicht direkt mit einer Komponente zusammenhängen. Das ist z.B. ein Link auf die Startansicht des Webangebots (der klassische „Home“-Link) oder ein Link auf das Registrierungsformular.
- **Komponenten-Links** sind Links, die direkt auf eine Komponente verweisen. Es kann dabei auf ein einzelnes Inhaltsobjekt verwiesen werden, aber auch auf eine durch Parameter und optionale Filter definierte Übersicht von mehreren Inhaltsobjekten.
- **Link-Container** können weitere Links enthalten, fassen also mehrere Links unter einem Oberthema zusammen.
- **Automatische Link-Container** werden automatisch mit Links zu Inhaltsobjekten einer Komponente gefüllt. Dabei kann durch Filter eine Einschränkung auf bestimmte Inhaltsobjekte vorgenommen werden.

Bei allen Komponenten-Links muss zusätzlich noch die Ansicht der Komponente angegeben werden, in der die Inhaltsobjekte angezeigt werden sollen. Die Einstellungen zu einem bestimmten Menüeintrag werden auf der Seitenleiste in dem Kasten „Item-Properties“ vorgenommen. Menüeinträge können durch Drag&Drop beliebig im Menü verschoben werden. Die grafische Umsetzung eines Menüs sowie eventuell DHTML-Effekte für Link-Container werden in den globalen Ansichten des Webangebots definiert. So kann ein Menü in verschiedenen Ansichten wiederverwendet werden.

4.3.8 Gestaltung von globalen Ansichten

Ein Webangebot kann in W2MS mehrere globale Ansichten haben. Eine globale Ansicht steuert auf oberster Ebene, was die Besucher wie von dem Webangebot zu Gesicht bekommen. Auf diese Weise lässt sich ein Webangebot für verschiedene Zielgruppen (wie z.B. Geschäftskunden und Privatkunden) und für verschiedene Endgeräte (z.B. Mobiltelefon und PC mit vollwertigem Browser) anpassen. Im Anhang sind zwei Programmansichten zu finden, die die Bearbeitung einer globalen Ansicht einmal für einen normalen Browser (Abbildung B.8) und einmal für ein Mobiltelefon zeigen (Abbildung B.9).

Die Gestaltung von globalen Ansichten läuft ähnlich ab wie bei den schon in 4.3.5 beschriebenen Komponentenansichten. Elemente können der Ansicht hinzugefügt und per Drag&Drop angeordnet werden. Jedes Element hat bestimmte Eigenschaften, die in dem Seitenleistenkasten „Item-Properties“ geändert werden können. Außerdem können die Elemente auch hier über den Kasten „Item-CSS“ mit CSS-Attributen und -Klassen versehen werden. In Abbildung 4.16 wird so z.B. ein „tag cloud“-Element, wie für tag clouds üblich, mit Blocksatz und nicht-unterstrichenen Links ausgestattet.

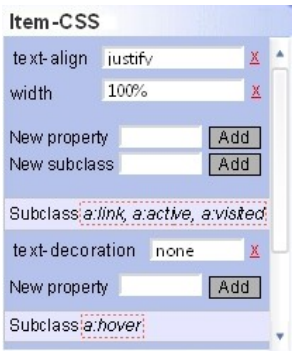


Abbildung 4.16: Bearbeitung von CSS-Eigenschaften eines Elements in der Seitenleiste von W2MS

In der globalen Ansicht gibt es jedoch eine größere Anzahl an Elementen, die hinzugefügt werden können. Außerdem ist eine globale Ansicht in neun Bereiche aufgeteilt, in denen die Elemente angeordnet werden können. Diese Bereiche ergeben sich aus der horizontalen Aufteilung in „left“, „center“ und „right“ sowie der vertikalen Aufteilung in „top“, „center“ und „bottom“. Es müssen nicht zwingend alle neun Bereiche verwendet werden. Solange ein Bereich leer bleibt, wird er in der Ansicht nicht verwendet. Zu Beginn befindet sich nur ein Element namens „content area“ in der Ansicht. Dieses Element kann nicht gelöscht werden und dient dazu, den Bereich der Ansicht zu definieren, in dem die Inhalte angezeigt werden. Alle Elemente, die nicht in der „content area“ angeordnet sind, sind immer sichtbar, egal welcher Inhalt des Webangebots gerade angezeigt wird.

Um neue Elemente in eine globale Ansicht einzufügen, klickt der Benutzer auf eine Miniaturdarstellung der neun Bereiche (vgl. Abbildung 4.17). Auf diese Weise wird das Element gleich an der entsprechenden Stelle in der Ansicht eingefügt.

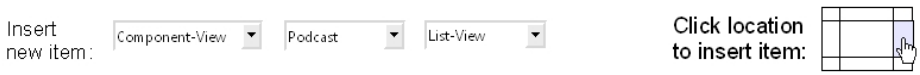


Abbildung 4.17: Dialog zur Einfügung eines Elements in eine globale View in W2MS

Folgende Elemente können dabei ausgewählt werden:

- **Komponenten:** Für jede Komponente gibt es hier verschiedene Optionen zur Auswahl. Neben den verschiedenen Ansichten einer Komponente kann auch ein bestimmtes Inhaltsobjekt einer Komponente auf der globalen Ansicht platziert werden. Alternativ kann man hier auch nur einen Link erzeugen, der dann auf die jeweilige Komponentenansicht führt. Außerdem lässt sich an dieser Stelle ein RSS-Link erzeugen, der in einem Feedreader integriert werden kann. Durch Parameter kann hier noch eingrenzt werden, welche Inhaltsobjekte der Komponente in den RSS-Feed integriert werden sollen.

- **Menüs:** Hier können die vorher erzeugten Menüs ausgewählt und auf der Ansicht platziert werden. In den „Item-Properties“ lässt sich einstellen, ob ein Menü mit mehreren Ebenen als Dropdown-Menü, oder in einer DHTML-freien Variante umgesetzt wird. Für das Menü werden einige für die Gestaltung relevanten CSS-Klassen vorangelegt, die in dem Kasten „Item-CSS“ bearbeitet werden können.

- **Formulare:** Hier können verschiedene Systemformulare, wie das Login-, das Registrierungs- sowie das Suchformular zur Ansicht hinzugefügt werden. Alternativ kann ein Link erzeugt werden, der zu dem entsprechenden Formular führt.

- **Tag cloud:** Hier lässt sich eine „tag cloud“ in die Ansicht einfügen. In den „Item-Properties“ kann dabei eine Eingrenzung auf eine Komponente vorgenommen werden. Außerdem kann die Anzahl der tags dort beschränkt werden.

- **Statischer Text oder Bild:** Texte und Bilder können hier analog zu den Komponentenansichten eingefügt werden.

Über den Karteireiter in der unteren rechten Ecke kann man von der Layout-Ansicht zur Bearbeitung des HTML-Codes sowie der CSS-Definitionen wechseln. Die unter CSS eingegeben Styles betreffen alle Elemente, die innerhalb der globalen Ansicht angezeigt werden, also im Prinzip das ganze Webangebot. Allgemeingültige Styles sollten also an dieser Stelle definiert werden. Werden unter HTML Änderungen am HTML-Code vorgenommen, so sind diese in der Layout-Ansicht gleich sichtbar. Allerdings sind nur DIV-Tags, die mit der CSS-Klasse „W2MS_DnD“ versehen sind, auch per Drag&Drop verschiebbar. Damit ein Element den Sei-

tenleistenkasten „Item-CSS“ nutzen kann, muss es außerdem eine eindeutige ID haben, damit die definierten Styles eindeutig diesem Element zugewiesen werden können. Es sollte außerdem darauf geachtet werden, dass von W2MS erzeugte DIV-Container (erkennbar daran, dass deren ID mit „W2MS_“ beginnt) im HTML-Modus nicht gelöscht werden. Immer wenn der Benutzer den HTML-Code, oder CSS-Code bearbeitet, führt W2MS im Hintergrund eine Validitätsprüfung durch. Wenn der Code nicht validiert, wird unter der Code-Ansicht ein Problembericht angezeigt. Dieser verschwindet erst, wenn der ungültige Code vom Benutzer entfernt wird.

4.3.9 Benutzerverwaltung

W2MS kennt nur einen Typ von Benutzern. Es wird also auf unterster Ebene keine Unterscheidung zwischen einem Besucher des Webangebots und einem Administrator gemacht. Dies ermöglicht in Verbindung mit der Einstellung von Zugriffsrechten auf Komponenten eine einfache Realisierung der unterschiedlichsten Benutzungsstrukturen. Die Programmansicht zur Bearbeitung von Benutzergruppen ist in Abbildung B.10 im Anhang dargestellt. Jeder Gruppe können hier noch weitere Recht zugewiesen werden, die sich nicht auf eine einzelne Komponente beziehen. Diese Rechte sind selbsterklärend und können der Programmansicht entnommen werden. Eine wichtige Einstellung hier ist noch die Angabe der Standardgruppen für Besucher sowie für registrierte Benutzer. Wenn ein Besucher das Webangebot aufruft, wird von W2MS automatisch und unsichtbar ein neuer Benutzer generiert. Dieser erhält eine zufällig generierte ID, die sowohl in der Datenbank als auch in einem Cookie auf dem Client gespeichert wird. Dieser Benutzer wird außerdem der Standardgruppe für Besucher hinzugefügt. Wenn dieser Benutzer sich nun registriert, werden alle bis dahin aufgelaufenen Daten seiner neuen, selbstgewählten ID zugewiesen und er wird der Standardgruppe für registrierte Benutzer hinzugefügt.

Wie in Abbildung B.11 im Anhang zu sehen ist, kann auch die Profilstruktur der Benutzer in W2MS geändert werden. Dabei kann die Profilstruktur um Datenfelder des Typs Text, Bild oder Datum erweitert werden. Es kann darüber hinaus angegeben werden, welche der Datenfelder auf dem Registrierungsformular des Webangebots sichtbar sind.

5 Fazit

5.1 Rückblick

In dieser Arbeit wurde untersucht, wie die Entwicklung und Verwaltung von Webangeboten durch webbasierte Werkzeuge vereinfacht werden kann. Es bestätigte sich die einleitend gemachte Vermutung, dass die mangelnde Umsetzung von Ansätzen aus der Forschung in einer fehlenden Orientierung an den Wünschen und Zielen der Entwickler begründet ist. Webentwickler aus allen Kompetenzklassen äußerten, dass ihnen Ansätze aus der Forschung, so wie sie sind, als nicht praxistauglich erscheinen (vgl. 3.7.2). Wenngleich dabei im professionellen Bereich durchaus alternative, proprietäre formelle Ansätze verwendet werden, so ist im semi- bis nicht-professionellen Bereich ein informeller Entwicklungsstil an der Tagesordnung. Die bestimmenden Parameter im Entwicklungsprozess sind hier zum einen die Forderung nach direkten und schnellen Ergebnissen und zum anderen der Wunsch nach möglichst viel Flexibilität in Gestaltung und Funktionalität.

Die semi- bis nicht-professionellen Entwickler wurden schließlich als Zielgruppe für das browserbasierte Entwicklungswerkzeug „W2MS“ ausgewählt, das im Zuge dieser Arbeit konzeptioniert und zum Teil implementiert wurde (vgl. Kapitel 4). Ein zentrales Ziel für W2MS bestand dabei in der Verknüpfung von Web-Engineering-Ansätzen mit dem informellen Entwicklungsstil der Zielgruppe. In Kapitel 3 wurde unter Anderem die Trennung von Struktur, Inhalt, Funktionalität und Layout als eines der Hauptanliegen des Web-Engineering identifiziert (vgl. 3.3 und 3.2.1). Diese Trennung ermöglicht die Flexibilität, die auf Grund der hohen Änderungsrate, die für Webangebote charakteristisch ist, für eine störungsfreie Aufrechterhaltung (Maintainability) eines Webangebots über längere Zeit unabdingbar ist. In W2MS wurde diese Flexibilität durch eine Komponentenarchitektur realisiert. Struktur, Zugriffsrechte, Inhalt und Layout einer Komponente werden dabei getrennt voneinander bearbeitet. Dies geschieht jedoch für den Benutzer auf eine klar erkennbare Art und Weise. Vor allem kann zwischen den Arbeitsschritten beliebig hin- und hergesprungen werden. Das Webangebot entsteht dabei automatisch im Hintergrund. Jeder Arbeitsschritt kann zudem rückgängig gemacht werden, es kann außerdem zu älteren Versionen zurückgesprungen werden. Somit wird einem informellen Entwicklungsstil entgegengekommen, ohne die Flexibilität aufzugeben. In Benutzertests wurde schon selbst anhand des noch unausgereiften Prototypens deutlich, dass Benutzer nach einer kurzen Eingewöhnung mit W2MS sehr schnell zu Ergebnissen kommen.

Ein durchgängiges Prinzip in W2MS ist die automatisierte Ausführung von Vorgängen im Hintergrund. In 3.7.2 wurde schon herausgefunden, dass dies eine entscheidende Vorbedingung für die erfolgreiche Etablierung von Web-Engineering-Methoden im semi- oder nicht-professionellen Bereich ist. In dem erweiterten Konzept für W2MS ist definiert, dass eine automatische Validitätsprüfung des HTML- und CSS-Codes durchgeführt wird, sobald der Benutzer diesen bearbeitet (vgl. 4.3.8). Bei der Bearbeitung von Inhaltsobjekten überwacht W2MS automatisch, ob alle Links erreichbar sind, ob für alle Bilder Alternativtexte vergeben sind sowie ob die auf den Feldern ggf. definierten Einschränkungen eingehalten wurden (vgl. 4.3.6). Der Benutzer muss sich also nicht aktiv um Tests bemühen und wird nur dann behelligt, wenn wirklich Fehler oder Probleme auftauchen.

In Bezug auf Accessibility bietet W2MS jedoch noch weitere Möglichkeiten. So kann ein Webangebot durch die Erstellung von passenden Ansichten für beliebige Endgeräte angepasst werden.

Bei W2MS wurde ein weiteres Augenmerk darauf gelegt, die technischen Möglichkeiten, die im

Zuge von Web 2.0 aufgekommen sind, konsequent und zielführend auszunutzen. Dies konnte sowohl in Bezug auf die Usability als auch auf die Funktionalität von W2MS erfolgreich umgesetzt werden. Die Oberfläche wurde dabei komplett im Sinne einer SPA realisiert. Die meisten der in 4.2.3 beschriebenen Usability-Features der Oberfläche erwiesen sich bei dem Benutzertest als hilfreich und wurden von den Benutzern gut angenommen (vgl. 4.2.4). Durch die Implementierung einer event-gesteuerten Client/Server-Kommunikation, die auf Comet und AJAX basiert (vgl. 4.2.1), konnte W2MS zudem als ein Mehrbenutzersystem mit kollaborativen Features realisiert werden. Wenngleich es hier sicher Bedarf für Verbesserungen gibt, so ist dies doch ein relativ neuartiger Ansatz für Webanwendungen, die momentan – bedingt durch die Beschränkungen von HTTP – in aller Regel in einem Einzelbenutzermodus laufen.

5.2 Ausblick

Es wurde in dieser Arbeit deutlich, dass noch viel ungenutztes Potential in dem Bereich der browserbasierten Webentwicklungswerkzeuge steckt. Aktuell tauchen fast täglich neue Webanwendungen auf, die mit beeindruckenden Oberflächen ausgestattet sind. Oft wird dabei sogar mittlerweile die Oberfläche einer gewöhnlichen Desktop-Anwendung in Funktionalität sowie allemal in der grafischen Qualität übertroffen. Die Möglichkeiten sind also da, um auch für den Bereich des Web-Engineering qualitativ hochwertige Webanwendungen zu schaffen. Die Erstellung eines Webangebots in einer Webanwendung hat dabei den Vorteil der Unmittelbarkeit – alle Entwicklungsschritte wirken sich unmittelbar auf das Webangebot aus. Eine Tatsache, die dem informellen und ergebnisorientierten Entwicklungsstil vieler Webentwickler entgegenkommt. Der Erfolg solcher Werkzeuge wird sich jedoch letztlich daran entscheiden, ob sie den Zielen und Wünschen der Entwickler entgegenkommen. Standards, Ansätze und Methoden haben nur wenig Wirkung, solange man sich nicht die Mühe macht, diese schlüssig mit den Zielen von Entwicklern in Einklang zu bringen und solange keine benutzbaren Werkzeuge dafür bereitgestellt werden. Es wäre also ein lohnenswerter und wichtiger Forschungsgegenstand, zu untersuchen, wie Web-Engineering *benutzbar* gemacht werden kann. Auch Methoden und Ansätze müssen eine gute Usability haben – zumindest wenn sie angewendet und umgesetzt werden wollen. Um das zu erreichen, werden jedoch Werkzeuge gebraucht, die den Entwickler ganzheitlich betreuen und die die Arbeit selber machen anstatt sie dem Entwickler aufzubürden. *Das* würde wirklich die effektive Umsetzung von Standards und eine Qualitätssteigerung von Webangeboten und damit des ganzen Web bedeuten – so wie es am Anfang dieser Arbeit gefordert wurde (vgl. 1.1). Das Konzept von W2MS soll ein Schritt in diese Richtung darstellen. Eine erweiterte Implementierung des Konzepts mit anschließenden Benutzertests würde weiteren Aufschluss über die Möglichkeiten, Stärken und Schwächen des Ansatzes von W2MS bringen.

Literaturverzeichnis

[ADOBE2006] Adobe. Flex2. Stand: 16. Oktober 2006. http://www.adobe.com/products/flex/whitepapers/pdfflex2wp_technicaloverview.pdf

[ADOBE2006b] Adobe. Adobe Dreamweaver. Stand: 18. Oktober 2006. http://www.adobe.com/products/dreamweaver/

[ATTERER2005] Richard Atterer. Where web engineering tool support ends: building usable websites. In: Proceedings of the 2005 ACM symposium on Applied computing - Web technologies and applications. Santa Fe, New Mexico, USA, März 2005, ACM Press, S. 1684-1688.

[AUTOMATIC2006] Automatic. WordPress. Stand: 25.Oktober 2006. http://www.wordpress.com

[BARRY2001] Chris Barry, Michael Lang. A Survey of Multimedia and Web Development Techniques and Methodology Usage. In: IEEE Multimedia, 8/2, April-June 2001, S. 2-10.

[BERNERSLEE1996] Tim Berners-Lee. The World Wide Web: Past, Present and Future. August 1996. http://www.w3.org/People/Berners-Lee/1996/ppf.html

[BERNERSLEE2006] Tim Berners-Lee. developerWorks Interviews podcast series: Tim Berners-Lee. 28. Juli 2006. http://www-128.ibm.com/developerworks/podcast/dwi/cm-int082206.txt

[BOYINK2005] Michael Boyink. Blog-based Sites vs. Traditional Sites. 15. Juli 2005. http://www.articlehub.com/Business/Blogbased-Sites-Vs-Traditional-Sites.html

[BOZZON2006] Alessandro Bozzon, Sara Comai, Piero Fraternali, Giovanni Toffetti. Conceptual modeling and code generation for rich internet applications. In: Proceedings of the 6th international on Web engineering - Session 14: modeling and tools. Palo Alto, California, USA, 2006, ACM Press, S. 353 - 360.

[CERI2000] S. Ceri, P. Fraternali, A. Bongio. Web Modeling Language (WebML): a modeling language for designing Web sites. In: Computer Networks, 33(1-6), 2000, S. 137-157.

[CERI2001] S. Ceri, P. Fraternali, M. Matera, A. Maurino. Designing multi-role, collaborative Websites with WebML: a CMS case study. 2001. http://www.webml.org/webml/upload/ent5/2/IWWOST01.pdf

[DESHPANDE2002] Y. Deshpande, S. Murugesan, A. Ginige, S. Hansen, D. Schwabe, M.Gaedke. Web Engineering. In: Journal of Web Engineering, 1/1, 2002, S. 3-17.

[DOJO2006] Dojo Foundation. Dojo toolkit. Stand: 15. Oktober 2006. http://www.dojotoolkit.org

[FITZGERALD1997] Brian Fitzgerald. The Use of Systems Development Methodologies in Practice. In: Information Systems Journal, 7/3, Juli 1997, S. 201-212.

[FRASER2006] Neil Fraser. Differential Synchronization. August 2006. http://neil.fraser.name/writing/sync/

[FUCHS2006] Thomas Fuchs. script.aculo.us. Stand: 15. Oktober 2006. http://script.aculo.us/

[GARRETT2005] Jesse James Garrett. Ajax: A New Approach to Web Applications. 18. Februar 2005. http://www.adaptivepath.com/publications/essays/archives/000385.php

[GINIGE2001] Athula Ginige, San Murugesan. WebEngineering:An Introduction. In: IEEE Multimedia, 8/1, January-March 2001, S. 14-18.

[GOOGLE2006] Google. Google Web Toolkit - Build AJAX apps in the Java language. 15. Oktober 2006. http://code.google.com/webtoolkit/

[GRIBBLE2006] Cheryl Gribble. History of the Web. Stand: 3. Oktober 2006. http://www.hitmill.com/internet/web_history.html

[HEWITT2006] Joe Hewitt. FireBug. Stand: 17. Oktober 2006. http://www.joehewitt.com/software/firebug/

[HIGGINS2006] Bill Higgins. Ajax and REST, Part 1. 2. Oktober 2006. http://www-128.ibm.com/developerworks/java/library/wa-ajaxarch/

[JOOMLA2006] Joomla!. Joomla!. Stand: 23. Oktober 2006. http://www.joomla.org/

[KIRDA2001] Engin Kirda, Mehdi Jazayeri, Clemens Kerer, Markus Schranz. Experiences in Engineering Flexible Web Services. In: IEEE Multimedia, 8/1, January-March 2001, S. 58 - 65.

[KNAPP2003] A. Knapp, N. Koch, F. Moser, G. Zhang. ArgoUWE:A Case Tool for Web Applications. First Int. Workshop on Engineering Methods to Supp Information Systems Evolution (EMSISE 2003), September 2003.

[KOCH2002] N. Koch, A. Kraus. The Expressive Power of UML-based Web Engineering. Second International Workshop on Web-oriented Software Technology (IWWOST 2002), Mai 2002.

[LANG2005] Michael Lang, Brian Fitzgerald. Hypermedia Systems Development Practices: A Survey. In: IEEE Software, 22/2, 2005, S. 68-75.

[MCDONALD2001] Andrew McDonald, Ray Welland. Web Engineering in Practice. Proceedings of the 4ht WWW10 Workshop on Web Engineering, Mai 2001.

[MURUGESAN1999] San Murugesan, Yogesh Deshpande, Steve Hansen and Athula Ginige. Web Engineering: A New Discipline for Development of Web-based Systems. Proc. First Int. Conf. Software Engineering (ICSE99) Workshop on Web Engeering, 1999.

[NETCRAFT2006] Netcraft. Web Server Survey. Stand: 29. Oktober 2006. http://news.netcraft.com/archives/web_server_survey.html

[NIELSEN1999] Jakob Nielsen. User Interface Directions for the Web. In: Communications of the ACM, 42/1, Januar 1999, S. 65-72.

[NIELSEN2003] Jakob Nielsen. Usability 101: Introduction to Usability. 25. August 2003. http://www.useit.com/alertbox/20030825.html

[NING2006] Ning. Ning - Create your own social websites!. Stand: 24. Oktober 2006. http://www.ning.com/

[OLASZLO2006] OpenLaszlo. OpenLaszlo. Stand: 16. Oktober 2006.

http://www.openlaszlo.org/

[OREILLY2005] Tim O'Reilly. What Is Web 2.0 - Design Patterns and Business Models for the Next Generation. 30. September 2005. http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

[OREILLY2005b] TimO'Reilly. Web 2.0: Compact Definition?. 1. Oktober 2005. http://radar.oreilly.com/archives/2005/10/web_20_compact_definition.html

[PAULSON2005] Linda Dailey Paulson. Building Rich Web Applications with Ajax. In: IEEE Computer, 38/10, Oktober 2005, S. 14-17.

[PROIETTI2006] Valerio Proietti. mootools - an ultra compact javascript framework. Stand: 15. Oktober 2006. http://mootools.net/

[REDDOT2006] RedDot Solutions AG. RedDot Web Content Management. Stand: 23. Oktober 2006. http://reddot.de/produkte_web_content_management.htm

[RESIG2006] John Resig. jQuery - New Wave Javascript. Stand: 15. Oktober 2006. http://jquery.com/

[RODE2003] Jochen Rode, Mary Beth Rosson. Programming at Runtime: Requirements & Paradigms for Nonprogrammer WebApplication Development. In: IEEE HCC 2003. Auckland, New Zealand, 2003, IEEE Computer Society Press, S. 23-30.

[RODE2004] Jochen Rode, Mary Beth Rosson, Manuel A. Pérez-Quiñones. End-Users Mental Models of Concepts Critical to Web Application Development. In: Proc. of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing (VLHCC'04). 2004, IEEE Computer Society, S. 215-222.

[RODE2005] Jochen Rode, Mary Beth Rosson, Manuel A. Pérez-Quiñones. The Challenges of Web Engineering and Requirements for Better Tool Support. Virginia Polytechnic Institute and State University, Computer Science, 2005.

[ROR2006] Ruby on Rails. Ruby on Rails. Stand: 16. Oktober 2006. http://www.rubyonrails.org/

[RUSSELL2006] Alex Russell. Comet: Low Latency Data for the Browser. 3. März 2006. http://alex.dojotoolkit.org/?p=545

[SCHWABE1998] Daniel Schwabe, Gustavo Rossi. An Object Oriented Approach to Web-Based Applications Design. In: Theory and Practice of Object Systems, 4/4, 1998, S. 207-225.

[SCHWABE2001] Daniel Schwabe, Gustavo Rossi, Luiselena Esmeraldo, Fernando Lyardet. Engineering Web Applications for Reuse. In: IEEE Multimedia, 8/1, January-March 2001, S. 20-31.

[STEPHENSON2006] Sam Stephenson. prototype JavaScript Framework. Stand: 15. Oktober 2006. http://prototype.conio.net/

[TAYLOR2002] M. J. Taylor, J. McWilliam, H. Forsyth and S. Wade. Methodologies and website development: a survey of practice. In: Information and Software Technology, 44, 2002, S. 381-391.

[TYPO32006] TYPO3 Association. TYPO3-CMS. Stand: 23. Oktober 2006. http://typo3.com/

[WEBRATIO2006] WebRatio. WebRatio. Stand: 12. Otkober 2006. http://www.webratio.com

[WHITLEY1998] Edgar A. Whitley. Method-ism in practice: investigating the relationship between method and understanding in Web page design. In: Proceedings of the international conference on inftion systems. Helsinki, Finland, 1998, Association for Information Systems, S. 68 - 75.

[YAHOO2006] Yahoo!. Yahoo! User Interface Library. Stand: 15. Oktober 2006. http://developer.yahoo.com/yui/

Glossar

AJAX: Abk. für „Asynchronous JavaScript and XML“. Steht für eine asynchrone Kommunikation des Clients mit dem Server ohne Neuladen der Browserseite.

Client: Steht im Kontext dieser Arbeit für einen Browser, über den ein Benutzer auf ein Webangebot zugreift.

CMS: Abk. für Content-Management-System(e)

Comet: Steht für eine Technik, die es dem Server ermöglicht, Daten in Echtzeit zum Client zu senden, ohne dass diese durch den Benutzer angefordert worden wären.

RichGUI: Abk. für „Rich Graphical User Interface“. Steht im Kontext dieser Arbeit für eine dynamische Weboberfläche, die reichhaltige Interaktionsmöglichkeiten bietet und von hoher grafischer Qualität ist.

SPA: Abk. für „single page application“. Steht für Webangebote, bei denen während der gesamten Benutzung kein Neuladen der Browserseite stattfindet.

W2MS: Abk. für Web 2.0 Management System, Name des in dieser Arbeit konzeptionierten Webentwicklungswerkzeugs.

Web: Die Bezeichnung Web soll als Kurzbezeichnung für World Wide Web verstanden werden und ist somit synonym mit der Abkürzung *WWW*. Das Web umfasst alles, was durch eine URL referenzierbar ist.

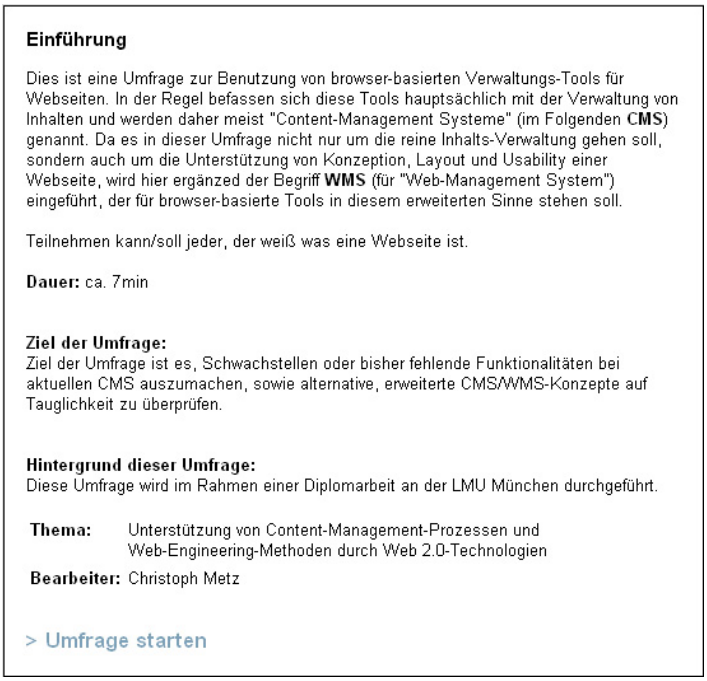
Webangebot: Der Begriff Webangebot soll umfassend für alles stehen, was im Internet von *einem* Anbieter unter *einem* Namen verfügbar gemacht wird. Dieser Name wird in der Regel ein Domain-Name sein, sofern nicht unter einer Domain mehrere Webangebote verfügbar gemacht werden. Somit ist sowohl die Präsentations-Assoziation des Begriffs Web*seite*, als auch die Funktions-Assoziation des Begriffs Web*anwendung* enthalten. Wenn in dieser Arbeit einer der letzteren beiden Begriffe verwendet wird, ist damit dann auch die genannte Assoziation beabsichtigt.

WYSIWYG: Abk. für „What you see is what you get“. Steht für die Eigenschaft von Benutzeroberflächen, dass der Benutzer Daten in einer Präsentationsansicht bearbeitet.

Einige Begriffe, die in dieser Arbeit explizit definiert werden, sind hier nicht gesondert aufgeführt.

A Umfrage zu Management-Tools für Webseiten

A.1 Umfrage



Einführung

Dies ist eine Umfrage zur Benutzung von browser-basierten Verwaltungs-Tools für Webseiten. In der Regel befassen sich diese Tools hauptsächlich mit der Verwaltung von Inhalten und werden daher meist "Content-Management Systeme" (im Folgenden **CMS**) genannt. Da es in dieser Umfrage nicht nur um die reine Inhalts-Verwaltung gehen soll, sondern auch um die Unterstützung von Konzeption, Layout und Usability einer Webseite, wird hier ergänzed der Begriff **WMS** (für "Web-Management System") eingeführt, der für browser-basierte Tools in diesem erweiterten Sinne stehen soll.

Teilnehmen kann/soll jeder, der weiß was eine Webseite ist.

Dauer: ca. 7min

Ziel der Umfrage:
Ziel der Umfrage ist es, Schwachstellen oder bisher fehlende Funktionalitäten bei aktuellen CMS auszumachen, sowie alternative, erweiterte CMS/WMS-Konzepte auf Tauglichkeit zu überprüfen.

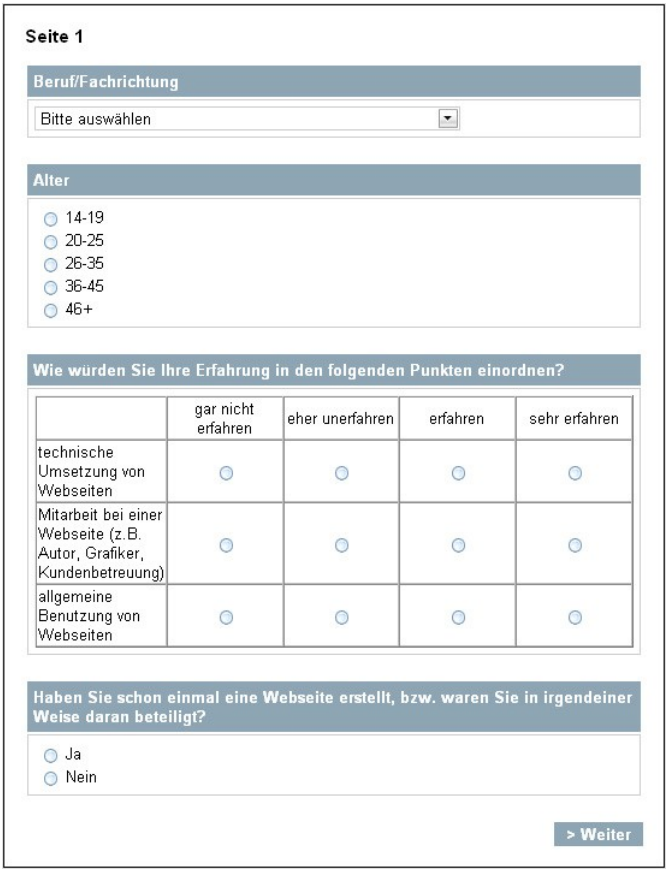
Hintergrund dieser Umfrage:
Diese Umfrage wird im Rahmen einer Diplomarbeit an der LMU München durchgeführt.

Thema: Unterstützung von Content-Management-Prozessen und Web-Engineering-Methoden durch Web 2.0-Technologien

Bearbeiter: Christoph Metz

> Umfrage starten

Abbildung A.1: Umfrage Einleitungsseite



Seite 1

Beruf/Fachrichtung

Bitte auswählen

Alter

- 14-19
- 20-25
- 26-35
- 36-45
- 46+

Wie würden Sie Ihre Erfahrung in den folgenden Punkten einordnen?

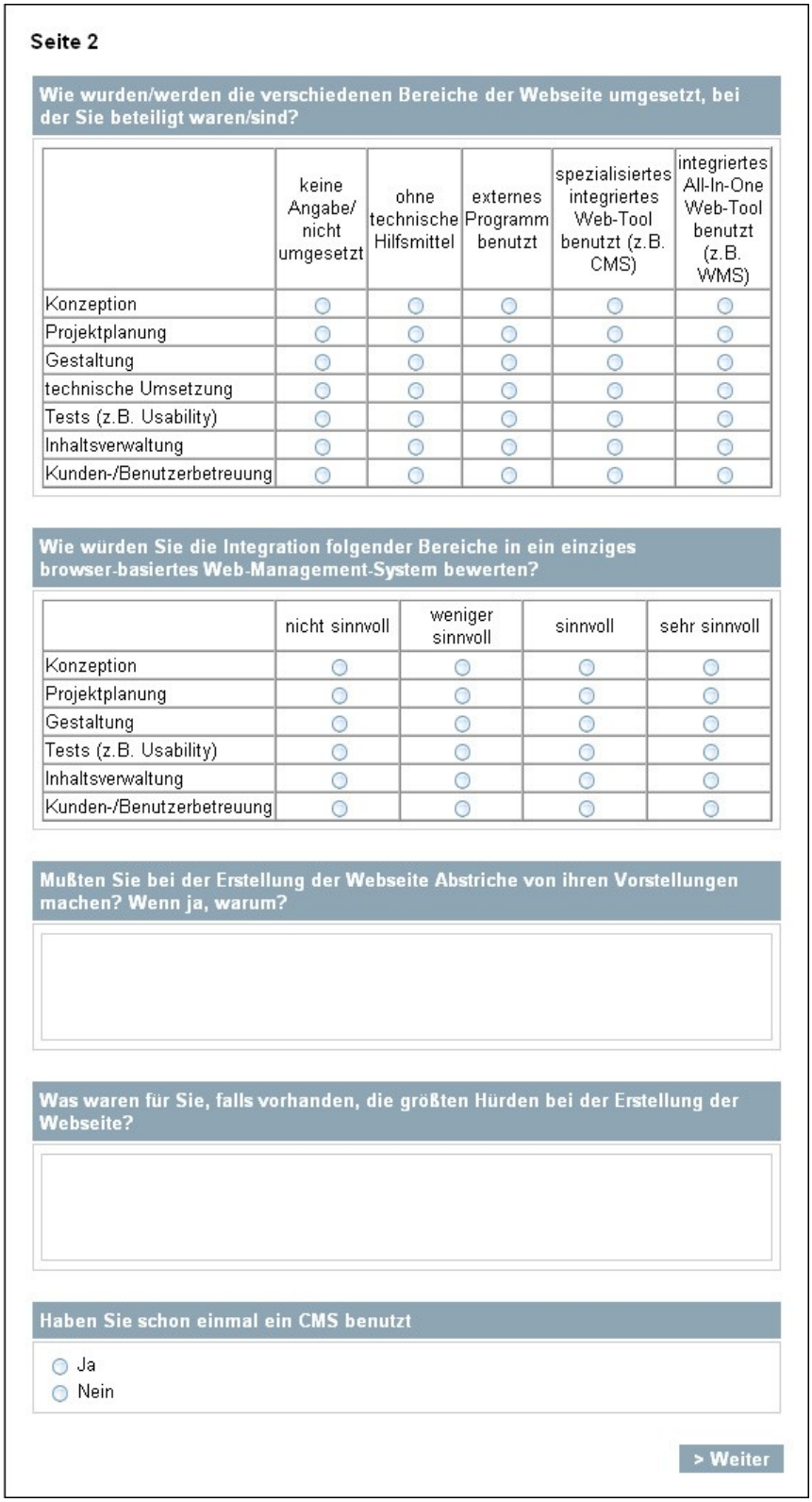
	gar nicht erfahren	eher unerfahren	erfahren	sehr erfahren
technische Umsetzung von Webseiten				
Mitarbeit bei einer Webseite (z.B. Autor, Grafiker, Kundenbetreuung)				
allgemeine Benutzung von Webseiten				

Haben Sie schon einmal eine Webseite erstellt, bzw. waren Sie in irgendeiner Weise daran beteiligt?

- Ja
- Nein

> Weiter

Abbildung A.2: Umfrage Seite 1



Seite 2

Wie wurden/werden die verschiedenen Bereiche der Webseite umgesetzt, bei der Sie beteiligt waren/sind?

	keine Angabe/ nicht umgesetzt	ohne technische Hilfsmittel	externes Programm benutzt	spezialisiertes integriertes Web-Tool benutzt (z.B. CMS)	integriertes All-In-One Web-Tool benutzt (z.B. WMS)
Konzeption	○	○	○	○	○
Projektplanung	○	○	○	○	○
Gestaltung	○	○	○	○	○
technische Umsetzung	○	○	○	○	○
Tests (z.B. Usability)	○	○	○	○	○
Inhaltsverwaltung	○	○	○	○	○
Kunden-/Benutzerbetreuung	○	○	○	○	○

Wie würden Sie die Integration folgender Bereiche in ein einziges browser-basiertes Web-Management-System bewerten?

	nicht sinnvoll	weniger sinnvoll	sinnvoll	sehr sinnvoll
Konzeption	○	○	○	○
Projektplanung	○	○	○	○
Gestaltung	○	○	○	○
Tests (z.B. Usability)	○	○	○	○
Inhaltsverwaltung	○	○	○	○
Kunden-/Benutzerbetreuung	○	○	○	○

Mußten Sie bei der Erstellung der Webseite Abstriche von ihren Vorstellungen machen? Wenn ja, warum?

Was waren für Sie, falls vorhanden, die größten Hürden bei der Erstellung der Webseite?

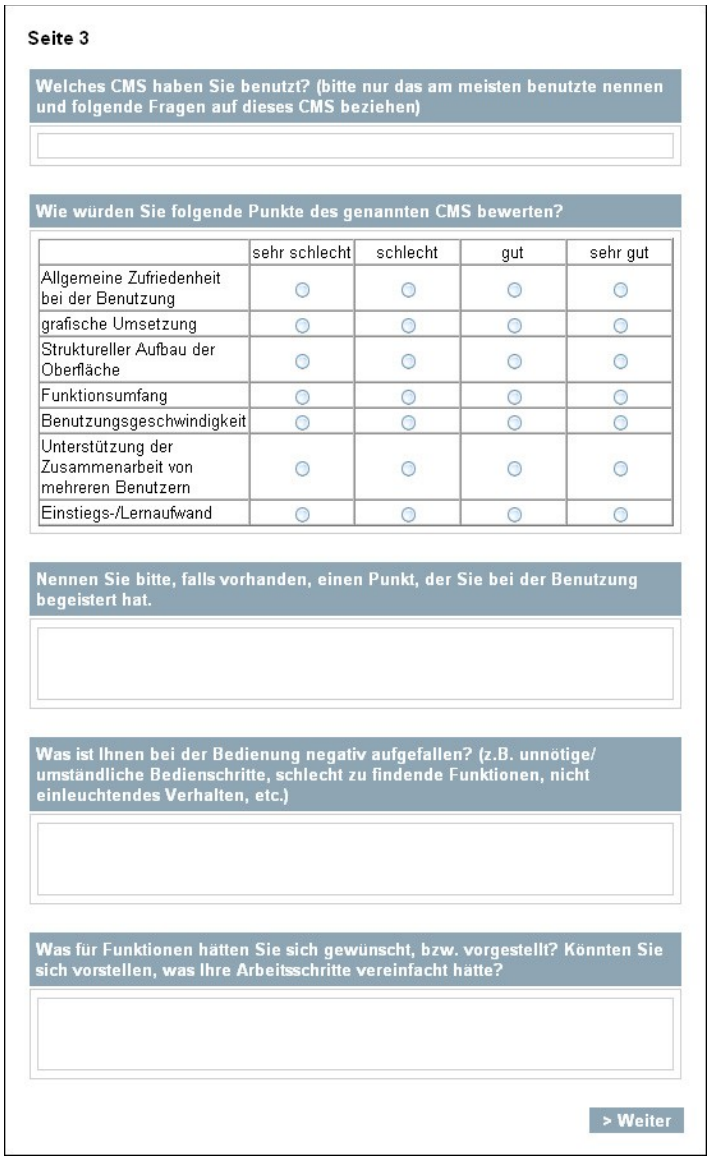
Haben Sie schon einmal ein CMS benutzt

○ Ja
○ Nein

> Weiter

Abbildung A.3: Umfrage Seite 2

- *Seite 3 erscheint nur, wenn die letzte Frage auf Seite 2 bejaht wurde.*
- *Die erste Frage auf Seite 4 erscheint nur, wenn die letzte Frage auf Seite 1 verneint wurde. Wird diese bejaht, so werden die Seiten 2 und 3 übersprungen.*



Seite 3

Welches CMS haben Sie benutzt? (bitte nur das am meisten benutzte nennen und folgende Fragen auf dieses CMS beziehen)

Wie würden Sie folgende Punkte des genannten CMS bewerten?

	sehr schlecht	schlecht	gut	sehr gut
Allgemeine Zufriedenheit bei der Benutzung	○	○	○	○
grafische Umsetzung	○	○	○	○
Struktureller Aufbau der Oberfläche	○	○	○	○
Funktionsumfang	○	○	○	○
Benutzungsgeschwindigkeit	○	○	○	○
Unterstützung der Zusammenarbeit von mehreren Benutzern	○	○	○	○
Einstiegs-/Lernaufwand	○	○	○	○

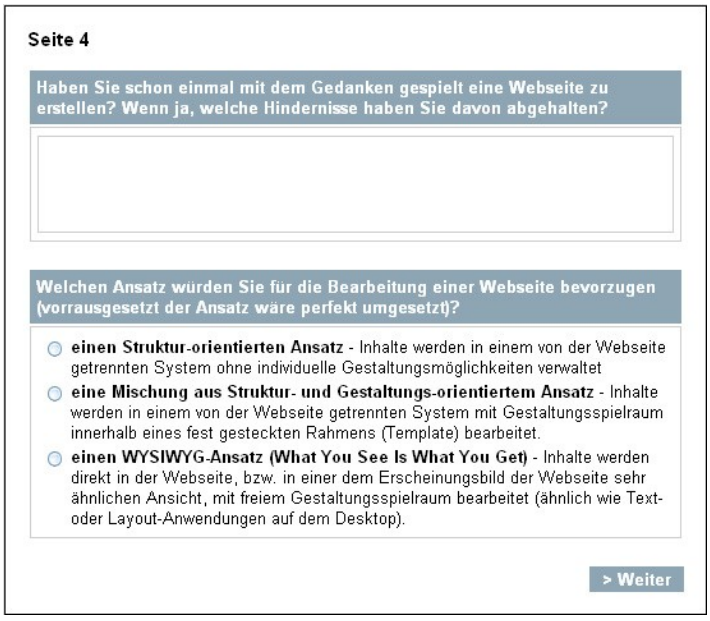
Nennen Sie bitte, falls vorhanden, einen Punkt, der Sie bei der Benutzung begeistert hat.

Was ist Ihnen bei der Bedienung negativ aufgefallen? (z.B. unnötige/ umständliche Bedienschritte, schlecht zu findende Funktionen, nicht einleuchtendes Verhalten, etc.)

Was für Funktionen hätten Sie sich gewünscht, bzw. vorgestellt? Könnten Sie sich vorstellen, was Ihre Arbeitsschritte vereinfacht hätte?

> Weiter

Abbildung A.4: Umfrage Seite 3



Seite 4

Haben Sie schon einmal mit dem Gedanken gespielt eine Webseite zu erstellen? Wenn ja, welche Hindernisse haben Sie davon abgehalten?

Welchen Ansatz würden Sie für die Bearbeitung einer Webseite bevorzugen (vorrausgesetzt der Ansatz wäre perfekt umgesetzt)?

- ○ **einen Struktur-orientierten Ansatz** - Inhalte werden in einem von der Webseite getrennten System ohne individuelle Gestaltungsmöglichkeiten verwaltet
- ○ **eine Mischung aus Struktur- und Gestaltungs-orientiertem Ansatz** - Inhalte werden in einem von der Webseite getrennten System mit Gestaltungsspielraum innerhalb eines fest gesteckten Rahmens (Template) bearbeitet.
- ○ **einen WYSIWYG-Ansatz (What You See Is What You Get)** - Inhalte werden direkt in der Webseite, bzw. in einer dem Erscheinungsbild der Webseite sehr ähnlichen Ansicht, mit freiem Gestaltungsspielraum bearbeitet (ähnlich wie Text- oder Layout-Anwendungen auf dem Desktop).

> Weiter

Abbildung A.5: Umfrage Seite 4

A.2 Ausgewählte Ergebnisse

Teilnehmeranzahl: 27

Kundenbetreuung	2	7%
Programmierer	12	44%
sonstige(r) Beruf/Fachrichtung mit Bezug zum Internet	11	41%
sonstige(r) Beruf/Fachrichtung ohne Bezug zum Internet	2	7%

Tabelle A.1: Frage nach Beruf/Fachrichtung (N=27)

14-19	1	4%
20-25	12	44%
26-35	13	48%
36-45	1	4%

Tabelle A.2: Frage nach Alter (N=27)

struktur-orientiert	3	11%
Mischung aus struktur- und gestaltungs-orientiert	11	41%
WYSIWYG	13	48%

Tabelle A.3: Frage: Welchen Ansatz würde Sie für die Bearbeitung einer Webseite bevorzugen?(N=27)

	nicht sinnvoll	weniger sinnvoll	sinnvoll	sehr sinnvoll
Konzeption	6	12	7	1
Projektplanung	6	8	11	1
Gestaltung	3	5	11	7
Tests (z.B. Usability)	2	4	10	10
Inhaltsverwaltung	0	0	6	20
Kunden-/Benutzerbetreuung	1	3	9	13

Tabelle A.4: Frage: Wie würden Sie die Integration folgender Bereiche in ein einziges, browser-basiertes Web-Management-System bewerten (N=26)

	keine Angabe/ nicht umgesetzt	ohne technische Hilfsmittel	externes Programm benutzt	spezialisiertes Web-Tool benutzt (z.B. CMS)	All-In-One Web-Tool benutzt (z.B. WMS)
Konzeption	1	16	5	3	1
Projektplanung	6	13	6	1	0
Gestaltung	1	1	21	1	2
technische Umsetzung	0	2	10	12	2
Tests (z.B. Usability)	6	12	4	3	1
Inhaltsverwaltung	1	5	3	17	0
Kunden-/Benutzerbetreuung	8	7	3	7	1

Tabelle A.5: Frage: Wie wurden/werden die verschiedenen Bereiche der Webseite umgesetzt, bei der sie beteiligt waren/sind? (N=26)

	sehr schlecht	schlecht	gut	sehr gut
Lernaufwand	1	7		
Nutzungsgeschwindigkeit	1	4	2	1
Struktureller Aufbau		4	4	
Funktionsumfang		1	4	3

Tabelle A.6: Bewertung von TYPO3 durch Benutzer dieses CMS (N=8)

Die gesamten Ergebnisse sind in der Excel-Datei „umfrageergebnisse.xls“ im Verzeichnis „Umfrage“ auf der dieser Arbeit beigelegten CD zu finden.

B W2MS-Programmansichten

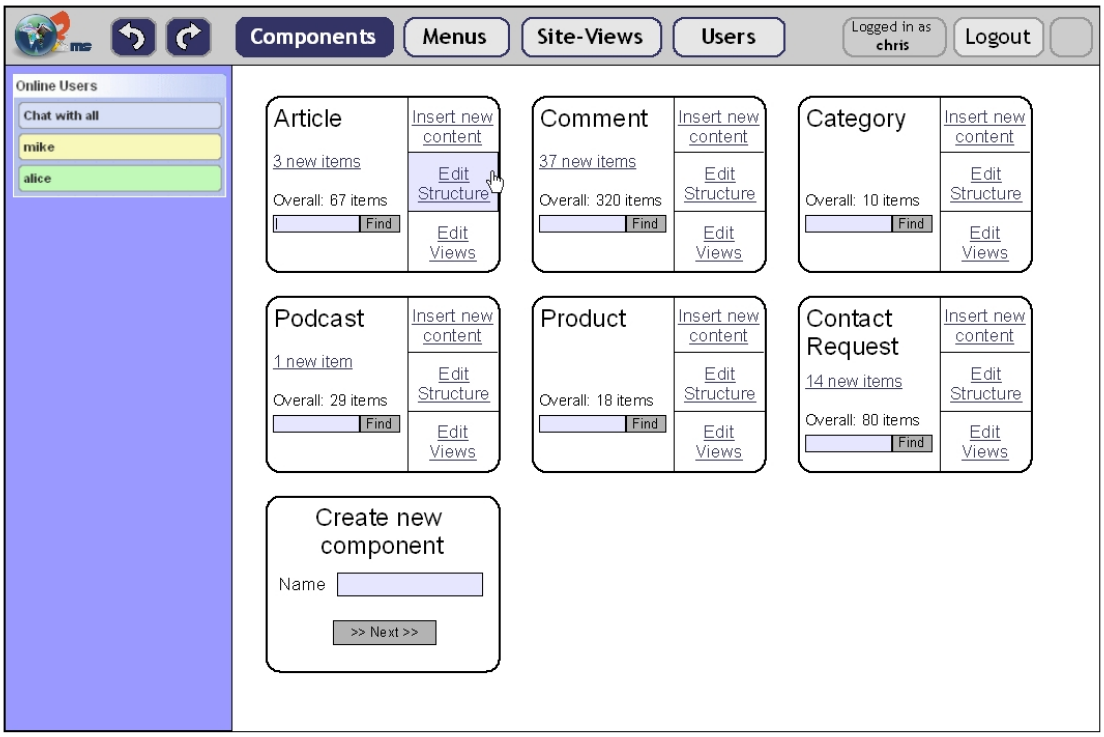


Abbildung B.1: Komponentenübersicht

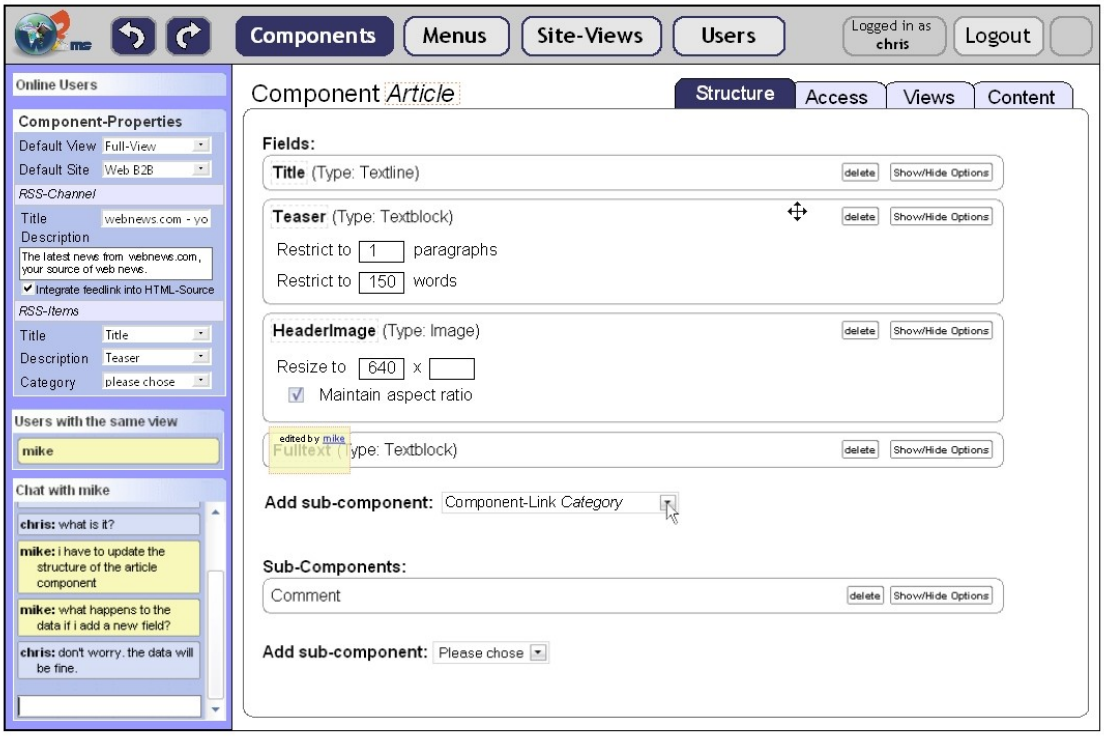


Abbildung B.2: Struktur einer Komponente bearbeiten

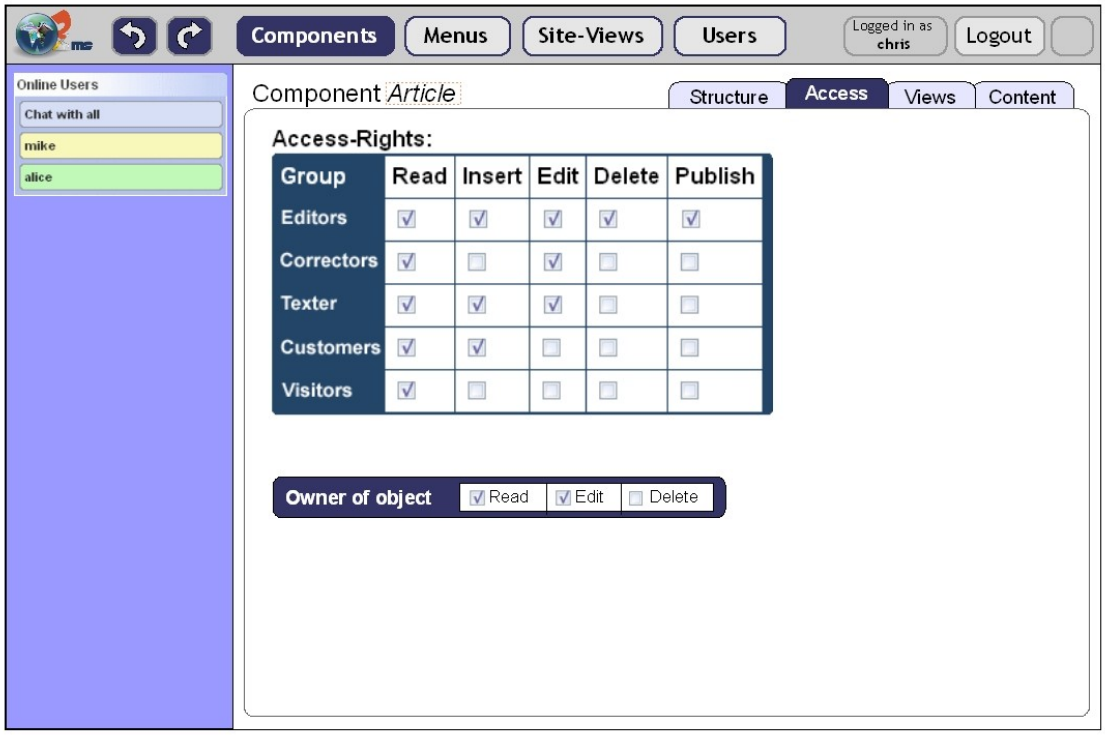


Abbildung B.3: Zugriffsrechte einer Komponente bearbeiten

Abbildung B.4: Ansichten einer Komponente bearbeiten

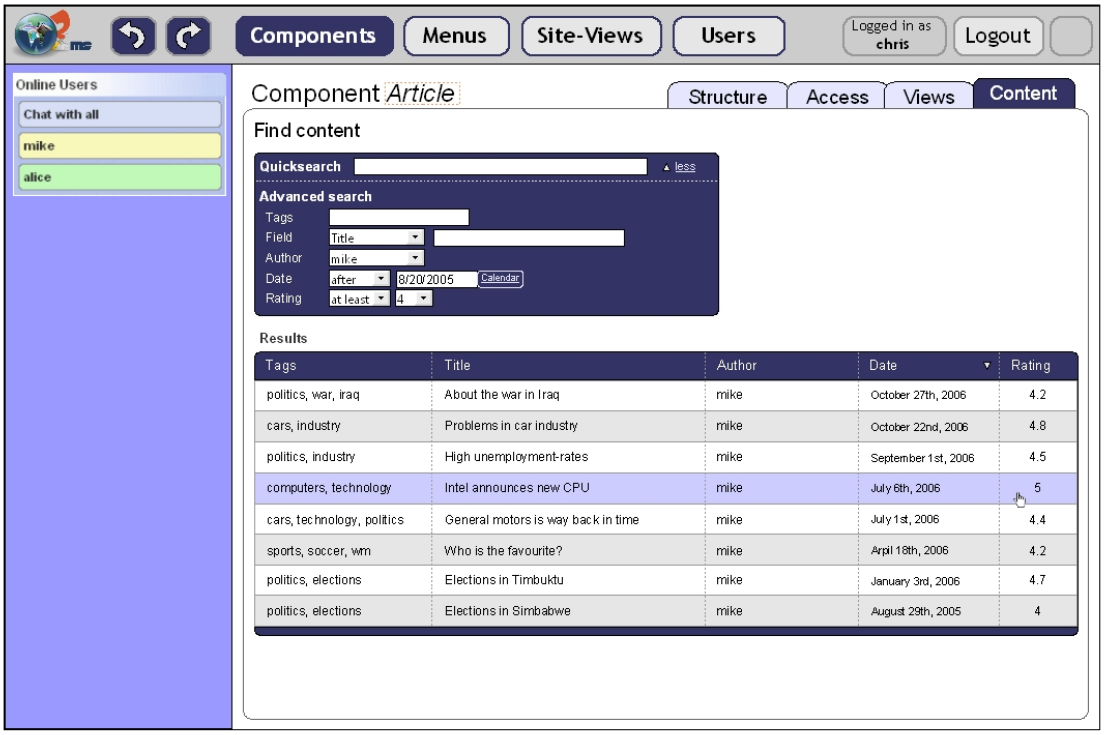


Abbildung B.5: Inhaltsobjekte einer Komponente finden

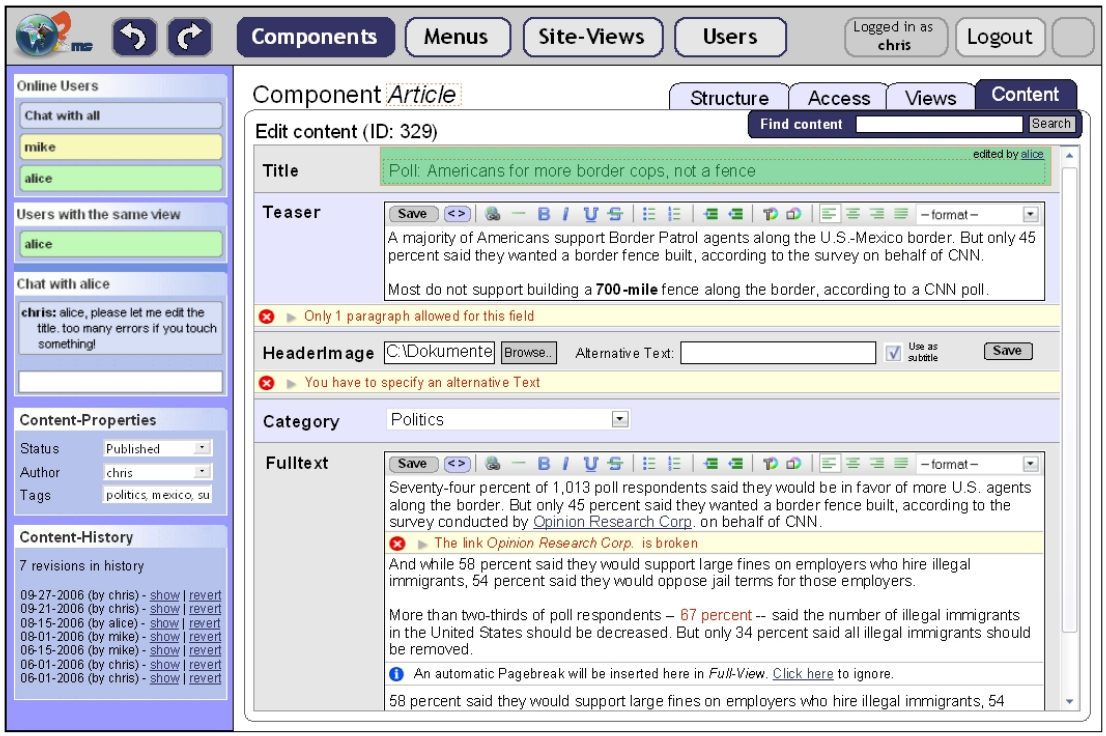


Abbildung B.6: Inhaltsobjekt einer Komponente bearbeiten (alle Felder bis auf das erste sind im Bearbeitungsmodus)

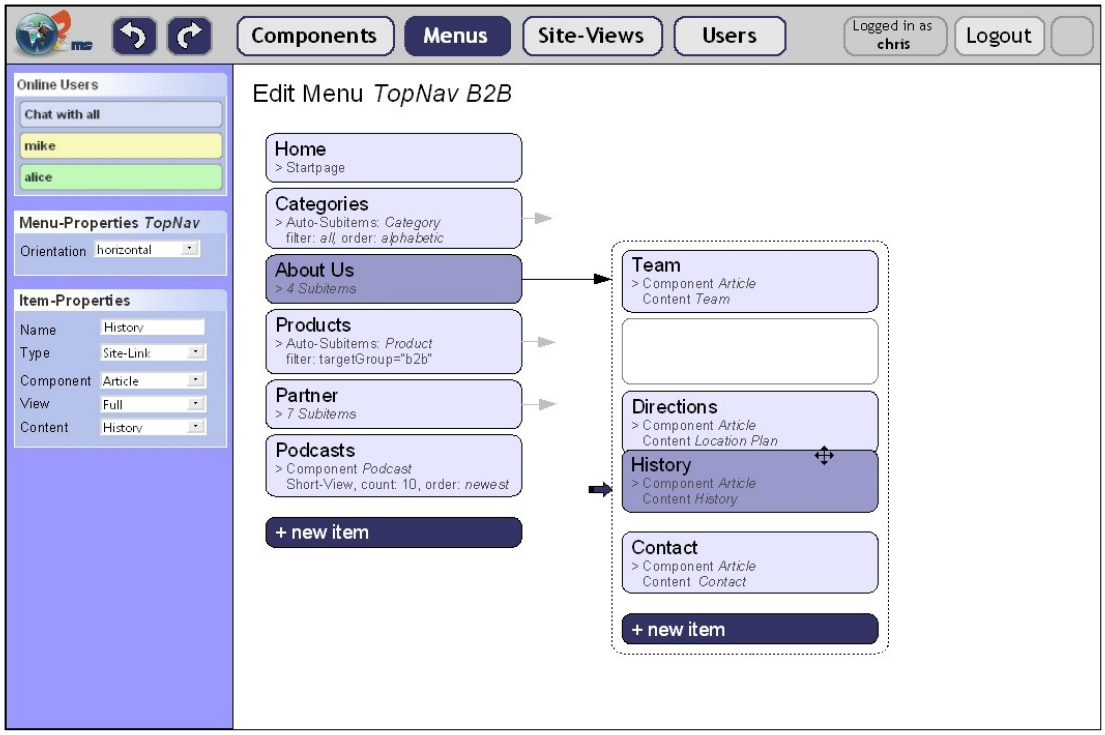


Abbildung B.7: Menü bearbeiten

Abbildung B.8: Globale Ansicht des Webangebots bearbeiten (#1)

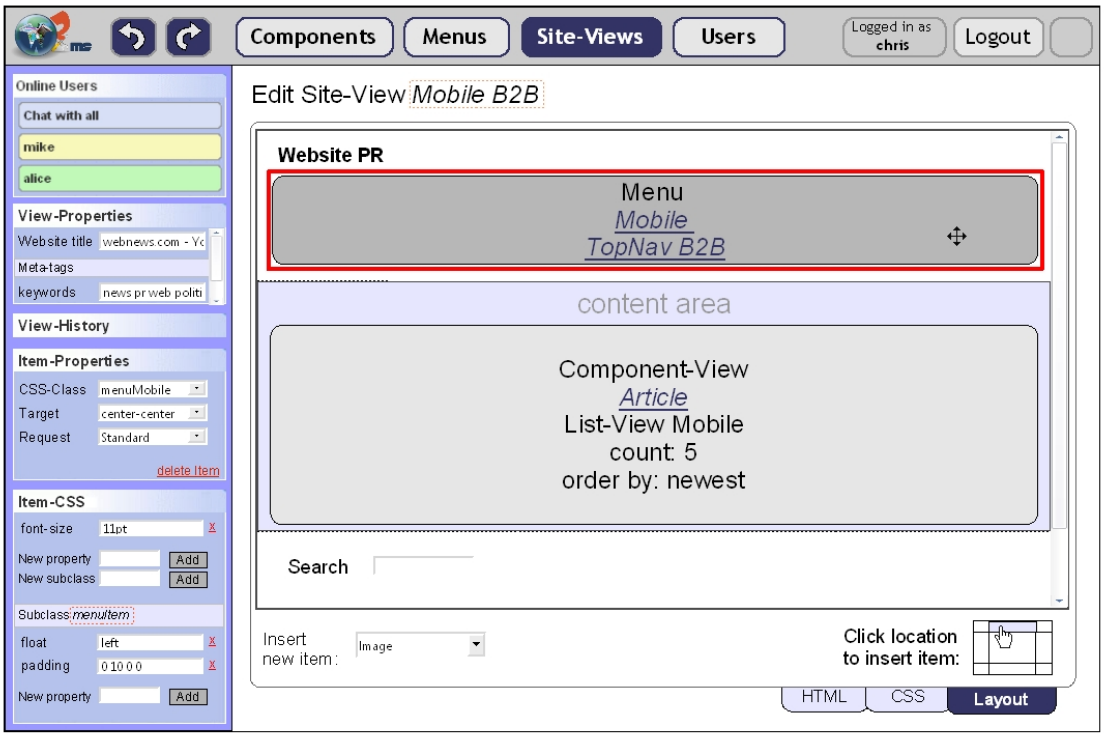


Abbildung B.9: Globale Ansicht des Webangebots bearbeiten (#2)

Abbildung B.10: Benutzergruppen bearbeiten

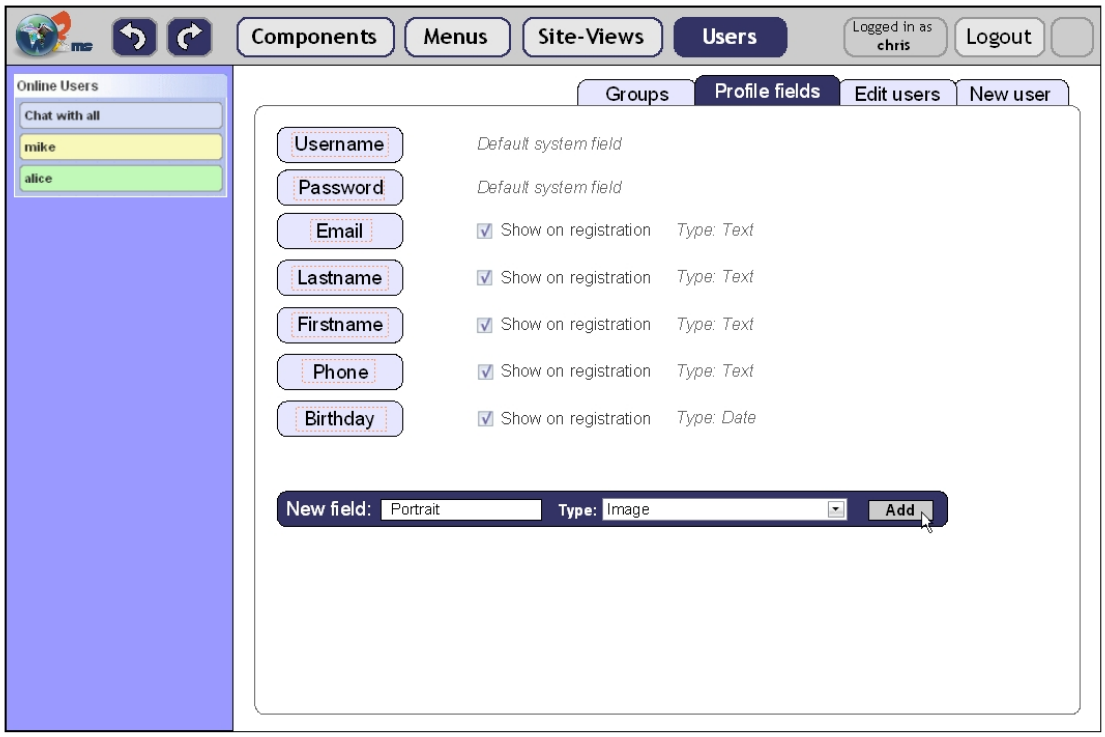


Abbildung B.11: Profilschema der Benutzer bearbeiten

Inhalt der CD

Name	Typ	Beschreibung
Ausarbeitung	Verzeichnis	*Enthält diese Arbeit im OpenOffice- sowie im PDF-Format.*
Bilder	Verzeichnis	*Enthält die im Rahmen dieser Arbeit erzeugten Grafiken im OpenOffice- sowie im JPG-Format.*
Quellen	Verzeichnis	*Enthält die in dieser Arbeit benutzten Quellen, soweit als PDF verfügbar.*
Umfrage	Verzeichnis	*Enthält Screenshots der im Rahmen dieser Arbeit durchgeführten Umfrage sowie die Ergebnisse der Umfrage im Excel-Format.*
W2MS	Verzeichnis	*Enthält den Quellcode des Prototypen von W2MS. Installationsinformationen sind in der Datei install.txt enthalten.*
Zwischenbericht	Verzeichnis	*Enthält den Zwischenbericht zu dieser Diplomarbeit im Powerpoint-Format.*
readme.txt	Datei	*Dieses Inhaltsverzeichnis*

Ich danke meinem Gott, Retter und Herrn

Jesus Christus.

Er gibt mir Freude, die ich nicht fassen kann.
Er gibt mir Frieden, den ich nicht verstehen kann.

Er war meine Kraft, als ich keine mehr hatte.
Er war meine Hoffnung, als ich keine mehr sah.
Er war mein Willen, als ich aufgeben wollte.

Er gab mir den Verstand.
Er ermöglichte mir ein Studium.
Er half mir zahllose Male.
Er griff ein durch Wunder.
Er hörte alle meine Gebete.

Er brachte mich sicher bis zum Ende.

SOLI DEO GLORIA