5. Technologiebewusstes Design von Web-Anwendungen

Gerhard Austaller, Markus Lauff, Fernando Lyardet, Max Mühlhäuser

Zusammenfassung:

Das Web entstand als extrem simples, aber weltweite Links unterstützendes Hypertextsystem. Dieses »Erfolgsrezept Einfachheit« stellt den Einsatz ausgereifter *Design-Methoden und –Werkzeuge für Hypertext-/Hypermedia* bis heute in Frage. Erst mit XML werden »ausgewachsene« Hypertextsysteme möglich, sind aber noch lange nicht üblich. Die kurze Geschichte des Web rückte überdies schon vor über zehn Jahren Datenbank-Anbindungen und damit *Informationsdesign* ins Zentrum. Letzteres ist aber auch nur bedingt anwendbar. Längst ist im »Schmelztiegel Web« auch die Einbindung umfangreicher Software-Module auf Client- und Serverseite bedeutsam geworden und damit Techniken des *objektorientierten Softwaredesign*. Da jedoch alle genannten Aspekte bedeutsam geblieben sind, kann keine der drei Wurzeln allein befriedigende Lösungen bieten. Ansätze für Web-spezifisches Design sind also gefragt. Doch die »typischen« Charakteristika von Web-Anwendungen sind noch immer so sehr im Fluss und das Gebiet ist noch so jung, dass *best practice* und daraus ableitbare gute Entwurfsmethoden, -prozesse, -notationen und -werkzeuge sich erst langsam herausschälen. Daher kann heute nur folgender Ansatz empfohlen werden: Entwickler von Web-Anwendungen sollten Web-Anwendungen in drei logische Schichten gliedern, die jeweils nach demselben Prinzip zweigeteilt sind: der Unterscheidung zwischen den Bausteinen d.h. *Komponenten* (insbesondere den Knoten der Web-Anwendung d.h. Medien, Softwarekomponenten, Datenbankzugänge) und deren Anordnung in ein *Geflecht*. Folgende drei Ebenen sind dabei zu unterscheiden: 1. Präsentationsdesign, wo das *look and feel* bestimmt wird und multimodale Benutzeroberflächen ins Spiel kommen; 2. Interaktions-Design, wo die Navigation durch Geflechte und der konkrete Dialog mit Komponenten entworfen werden; 3. Funktionales Design, das den »Kern« der Web-Anwendung festlegt. Bei zunehmender Konkretisierung des Entwurfs auf jeder Ebene und für beide Teile (Knoten, Geflecht) muss auf Werkzeuge des Hypertext-, Informations- und Softwaredesign zurückgegriffen werden. Nach heutigem Stand steht für den integrativen Teil des Entwurfs (wie zuerst erläutert) noch keine befriedigende technische Unterstützung bereit.

5.1. Einleitung

Überblick: Im vorliegenden Kapitel soll in das Design von Web-Anwendungen eingeführt werden. Dazu muss zunächst die gängige Praxis beleuchtet werden. Weil diese insbesondere aus historischen Gründen von den wenig aufeinander abgestimmten Teilbereichen Informationsdesign, Hypertextdesign und Software-Design beeinflusst wird – die je nach Web-Anwendung sehr unterschiedlich stark ausgeprägt sein können –, wird in Unterkapitel 5.2 ein evolutorischer Ansatz gewählt, um diese Teilbereiche vorzustellen und Möglichkeiten und Probleme von deren Integration zu diskutieren. Das Thema Informationsdesign wird dabei nur gestreift, weil es in anderen Kapiteln dieses Buches ausführlich behandelt wird. Die dann folgenden drei Unterkapitel versuchen, das unkoordinierte Aufeinandertreffen mehrerer »Kulturen« zu überwinden durch eine strukturierte Dreiteilung nach neuartigen Gesichtspunkten– jedes der Unterkapitel entspricht dabei einem der drei Teile Präsentationsdesign, Interaktions-Design und Funktionales Design. Das letzte Unterkapitel gibt einen Ausblick auf aktuelle Problemstellungen unter dem übergeordneten Design-Ziel Nachhaltigkeit. Der Rest des hier vorliegenden Unterkapitels soll die in 5.3 bis 5.5detaillierten Teilbereiche als zusammenhängenden Ansatz einführen. Dazu müssen einige Ergebnisse aus Unterkapitel 5.2 vorweggenommen werden.

Bezüge zu anderen Kapiteln dieses Buches: das vorliegende Kapitel trägt der Tatsache Rechnung, dass das Design von Web-Anwendungen heute noch sehr stark von Spezifika bestimmter (bzw. Grenzen verfügbarer) Implementierungstechnologien bestimmt ist. Es hat daher vielfältige Berührungspunkte mit Kapitel 6, wobei versucht wird, hier nur die entwurfsrelevanten Technologieaspekte anzusprechen. Krasser sind die Bezüge zu Kapitel 3, mit dem das vorliegende Kapitel auf den ersten Blick teilweise in Widerspruch zu stehen scheint. Die Autoren hoffen jedoch, auf den zweiten Blick dem Leser einen ergänzenden Blickwinkel zu liefern. Das betrifft einerseits die Einordnung in den Lebenszyklus von Web-Anwendungen, wo das aktuelle Kapitel eher später anzusiedeln ist. Andererseits wird der – in Kapitel 3 bereits gewürdigten – Tendenz zu komplexeren und »software-lastigeren« Web-Anwendungen hier noch stärker Rechnung getragen. Kapitel 3 nimmt, wie schon dessen erste Abbildung zeigt, eine Schichtung nach Präsentation, Hypertext und Information vor. Die erste Abbildung des vorliegenden Kapitels macht deutlich, dass die »unterste Schicht« von Web-Anwendungen hier noch stärker vom Softwaretechnik-Aspekt geprägt ist (Funktion) als in Kapitel 3 vom Datenbank-Aspekt (Information). Das bringt neben mehr Komplexität den Vorteil, dass Modularisierung und grundlegende Designgesichtspunkte jetzt alle Schichten durchziehen können. Entsprechend ist der Hypertext-Aspekt hier erstens zweigeteilt (nach Geflecht und Komponenten) und zweitens orthogonal zu den drei Schichten angeordnet. Schließlich wird der in Kapitel 3 Präsentation genannte Teil weiter untergliedert einerseits in den »reinen« Präsentationsteil, der sich auf die Vermittlung von Sachverhalten (Medien, Inhalten, Interaktions*möglichkeiten,* etc.) an den Benutzer konzentriert und durch die Berücksichtigung multimodaler Aspekte an

Komplexität zunimmt, andererseits in den Interaktionsteil, der durch systematischen Entwurf ineinander greifende Initiativen von Mensch und Maschine besser zu modellieren gestattet. Das vorliegende Kapitel erweitert also die in Kapitel 3 vorgestellte Vorgehensweise in Richtung noch komplexerer und software-lastigerer Web-Anwendungen sowie Berücksichtigung von Technologien. Entsprechend wird auf die in Kapitel 3 hervorragend vorgestellten Modellierungstechniken nicht erneut eingegangen. Da darüber hinaus für die hier vorgestellte erweiterte Vorgehensweise erst rudimentär zusätzliche Methoden und Werkzeuge existieren, bleibt das Kapitel notgedrungen hinsichtlich Design-Methoden etwas »theoretisch«; dieses unvermeidliche Manko soll durch Nähe zu Implementierungstechnologien etwas ausgeglichen werden.

Kurzfassung der Historie: Die ersten Gehversuche des World-Wide Web waren von Dokument-zentrierten, sogar textbasierten HTML-Seiten geprägt, für die der Begriff Web-Anwendung übertrieben scheint. Die Wurzeln der Dokument-Beschreibungssprache HTML liegen in der Druck- und Verlagsbranche; HTML fördert die Erstellung größerer strukturierter Dokumente, die mit Verweisen angereichert sind; Erfolgsgeheimnis des WWW war dabei neben der kompromisslosen Einfachheit die volle Integration globaler Verweise, der URLs. »Größere Dokumente« soll heißen, dass HTML die *Verletzung* einer Maxime der Hypertext-Forschung befördert, wonach Knoten eines Hypertextes kleinstmögliche sinntragende Einheiten darstellen sollten. Dies und weitere Widersprüche zum Stand der Wissenschaft werden in Unterkapitel 5.2.1 näher erläutert. Erst mit XML-basierten Standards des W3C werden diese Defizite heute behebbar (s. http://www.w3c.org), teilweise auch nur »im Prinzip«, weil Verbesserungen nur verhalten in Browser und Web-Server Eingang finden.

Der Übergang zu »echten« Web-Anwendungen mit der Entwicklung von Skriptsprachen wie JavaScript (Browser-seitig) und CGI (serverseitig) brachte Interaktivität und Datenbank-Aspekte sowie die dynamische Erzeugung von HTML-Dokumenten ins Spiel, mit Java bekamen Web-Anwendungen zunehmend auch Software-Charakter – Kapitel 1 geht mit seiner Klassifizierung hierauf bereits ein.

Um diese vielfältigen Aspekte des Designs von Web-Anwendungen zu reflektieren, werden für die weitergehende Betrachtung Teilaufgaben des Designs von Web-Anwendungen gemäß Abb. 5-1 zugrunde gelegt. Dabei werden grundsätzlich die *Komponenten* von Web-Anwendungen, d.h. Knoten nebst der sie verbindenden Links und deren Start- bzw. Zielpunkte innerhalb von Knoten (»Anker«), unterschieden vom *Geflecht* aus solchen Komponenten, d.h. von der gesamten Web-Anwendung. »Gesamt« muss dabei ein schwammiger Begriff bleiben, weil Verweise auf Teile des WWW möglich bleiben sollen, die nicht im Verantwortungsbereich der Web-Anwendungsentwickler sind. Nur so bleibt das Web *world wide* (übrigens ist das Web *kein* weltweit zusammenhängendes Geflecht – es ist willkürlich partitioniert in Bereiche, zwischen denen »noch niemand einen Link erstellt hat«, während nachfolgend unter einer Web-Anwendung ein planmäßig zusammenhängendes aber nicht vollständig isoliertes Geflecht von Komponenten verstanden wird).

Abb. 5-1 legt neben der erwähnten Zweiteilung eine dreiteilige Schichtung nahe. Darin wird zunächst die klare Unterscheidung zwischen *Präsentation und Interaktion* vorgeschlagen, vergleichbar der Unterscheidung zwischen View und Controller im Model-View-Controller-Konzept MVC und seinen vielen Weiterentwicklungen [BMR+96]. Präsentation betrifft einerseits das *Geflecht* unter Berücksichtigung des aktuell vom Benutzer besuchten Knoten (dem »Cursor« auf dem Weg durch das Geflecht; ggf. gibt es davon mehrere parallel). Andererseits ist insbesondere die Präsentation von *Komponenten* d.h. Knoteninhalten eine zentrale Entwurfsaufgabe, siehe Kapitel 1: schon dort wird auf die betreffenden Charakteristika von Web-Anwendungen hingewiesen, insbesondere Content-bezogene wie Dokumentencharakter und Multimedialität sowie präsentationsbezogene wie Ästhetik und Selbsterklärbarkeit. Wichtig ist daher die Einbeziehung entsprechender Fachleute (hier Mediendesigner genannt) und deren Koordination mit anderen Web-Anwendungsentwicklern (hier als Ingenieure gekennzeichnet).

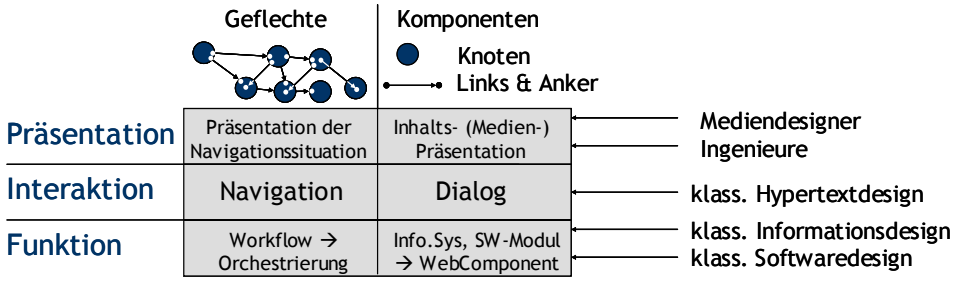

***Abb. 5-1** Grundelemente von Hypertext-Dokumenten*

Auf der Interaktionsebene setzt sich die Trennung zwischen Geflecht und Komponenten fort. *Interaktion im Geflecht* wird meist als Navigation bezeichnet. Diese kann explorativ erfolgen, wie dies im Ausdruck Browsing (Stöbern) zum Ausdruck kommt: der Benutzer verfolgt Links, die ihm beim Lesen interessant erscheinen, und bewegt sich auf einer scheinbar zufälligen Bahn durch das Geflecht. Sie kann aber auch durch möglicherweise sehr anspruchsvolle Software (mit-)gesteuert werden, welche in jeder Navigationssituation (bestimmt durch die »Cursor«) die Wahlmöglichkeiten anpasst - beispielsweise in Form einer dynamisch veränderlichen Navigationsleiste oder eines Graphenlayouts mit Graufärbung für aktuell deaktivierte Teile. Ausgefeilte Benutzermodelle, Domänenmodelle oder instruktionelle Strategien (bei eLearning) bestimmen ggf. diese Dynamik. *Interaktion mit Komponenten* sei als *Dialog* im Sinne einer Benutzeroberfläche bezeichnet. In der ersten Interaktionsstufe von Web-Anwendungen erfolgt der Dialog über Formulare (auch: Masken). Mit der Möglichkeit, Java-Software als Komponenten (Applets) einzusetzen, wurde erstmals die Fülle von Möglichkeiten graphischer Benutzeroberflächen auch für Web-Anwendungen verfügbar. Ubiquitäre Web-Anwendungen unterstützen zunehmend mobile Klienten bis hin zu sprachbasierten Endgeräten, multimodale Dialoge usw. Beim Entwurf sol-

cher Systeme stößt der oben erwähnte MVC-Ansatz auf Komponenten-Seite an Grenzen.

Als kleines Indiz für die Berechtigung der durchgeführten horizontalen (Präsentation / Interaktion) und vertikalen (Geflechte / Komponenten) Trennung sei angemerkt, dass im frühen HTML-basierten Web Geflechte überhaupt nicht präsentiert wurden, Navigation im Geflecht aber sehr wohl unterstützt wurde (durch Anklicken von Links oder über Vorwärts-/Zurück-Schaltflächen im Browser). Umgekehrt wurde die Präsentation der Komponenten (vom Browser ge-»renderte« HTML-Dokumente) seit der Zeit der ersten graphischen Browser sehr ernst genommen, während Dialog mit einer Komponente erst seit der Einführung von »Forms« in HTML überhaupt möglich ist.

Wie beim MVC-Konzept das so genannte »Model« den eigentlichen Kern, also die Funktionalität einer Anwendung beherbergt, so ist auch im Entwurfsprinzip nach Abb. 5-1 eine weitere Schicht erforderlich, nachfolgend *Funktionales Design* genannt. Der Begriff Funktion soll zunächst möglichst neutral alle Entwicklungsstufen des Web umfassen. Dabei waren beim Dokument-zentrierten Frühstadium vor allem statische d.h. durch Autoren vorgegebene Inhalte als Komponenten prägend, so dass als Funktionen im eigentlichen Sinne nur der Aufruf der Präsentations- bzw. Abspielfunktion (bei Medien) in Frage kam, die Schicht also quasi leer war. Im Zusammenhang mit der Zugriffsmöglichkeit auf Datenbanken wurden Web-Anwendungen zunehmend von den Prinzipien der klassischen Informationssysteme und Datenbanken beeinflusst, der Entwurf des funktionalen Teils war also im wesentlichen Informationsdesign. Mit dem Übergang zu den jüngeren in Kapitel 1 genannten Kategorien von Web-Anwendungen (Workflow-basiert, kollaborativ, Portal, ubiquitär) rücken Komponenten mit umfangreichem Funktionsumfang in den Vordergrund; selbstverständlich bieten sich objektorientierte Ansätze an, um funktionale und Datenaspekte integral zu modellieren. Entsprechend spielten Java und konkurrierende Ansätze insbesondere von Microsoft hier zunächst eine dominierende Rolle, in letzter Zeit substantiell ergänzt durch W3C-Standards, die die weltweite Interoperabilität im Web stärken sollen. Die gestiegene Bedeutung aktiver Komponenten drückt sich auch darin aus, dass neben »statische« Links auch Prozedurfernaufrufe treten, z.B. gemäß SOAP-Standard, und dass Zielknoten eines Links ggf. erst zur Laufzeit, z.B. via UDDI, bestimmt werden. Für die Geflecht-Seite waren in der Informationssystem-geprägten Zeit Ansätze des *Workflow Management* prägend, die sich für eher statische Modelle der Komponenten-Verflechtung – im Sinne von Schritten eines Geschäftsprozesses – eignen. Mit *WebServices* und der Idee, unternehmensübergreifende Geschäftsprozesse modellieren zu können, kamen flexiblere Konzepte der Verflechtung ins Spiel unter Begriffen wie *Orchestrierung* oder *Chorographie* [W3Ca], hier ist die Entwicklung noch im Fluss. Der Grad der Dynamik wird bei ubiquitären Web-Anwendungen noch zunehmen, wenn ein offener Dienstemarkt entsteht: Dienste sind dann quasi Web-Components, die sich Anwender bedarfs- und situationsgerecht zu einem Geflecht im Sinne einer ubiquitären hochpersonalisierten und komplexen Web-Anwendung orchestrieren (lassen) können.

Als generelle Vorgehensweise für den Entwurf bzw. das Design von Web-Anwendungen bietet sich nach allem Vorgenannten an, alle sechs in Abb. 5-1 grau hervorgehobenen Teilbereiche abzudecken und folgende Aspekte zu beachten:

- ❑ alle beteiligten Entwickler-Kulturen sind im Team zu besetzen, auf deren Integration ist zu achten.
- ❑ Für jeden der sechs Teilbereiche existieren eigene Modelle, Methoden und Werkzeuge für Entwurf und Implementierung, diese unterscheiden sich je nach Kategorie der Web-Anwendung (s. Kapitel 1) teilweise erheblich; eine auch nur annähernd vollständige Behandlung im vorliegenden Kapitel ist in der gebotenen Kürze unmöglich; an die Weiterentwicklung einer Web-Anwendung in Richtung anspruchsvollerer Kategorien ist zu denken;
- ❑ Die genannte Auswahl muss nicht nur die vorgesehenen Kategorie von Web-Anwendung betrachten, sondern auch die vorgesehenen Technologien; diese Abhängigkeit ist so stark, dass sie im vorliegenden Kapitel eine zentrale Rolle spielt.
- ❑ Ein übergeordneter Entwurfsprozess als Pfad durch die sechs Teilbereiche laut Abb. 5-1 kann nicht generell empfohlen werden. Sinnvoll ist die Abdeckung aller drei horizontalen Schichten erst auf der einen Seite und dann auf der anderen, ggf. iterativ. Während bei weitgehender Neuentwicklung einer Web-Anwendung die Geflecht-Seite zuerst entworfen werden kann, müssen wiederverwendbare Komponenten ohne Kenntnis von Geflechten zuerst entworfen werden. Innerhalb einer Seite empfiehlt sich aus Softwaretechnik-Sicht der Beginn auf der untersten Schicht, weil sich Interaktion an der Funktion orientieren kann und Präsentation an Interaktion. Stehen Ästhetik und andere benutzernahe Charakteristika von Web-Anwendungen im Vordergrund, empfiehlt sich ggf. der umgekehrte Weg.

5.2. Grundlagen: Web-Design aus evolutorischer Sicht

5.2.1. Anfänge

Wie schon zu Beginn des ersten Kapitels und oben erwähnt ist ein zentrales Merkmal von Web-Anwendungen die *Dokument-zentrierte Sichtweise*, d.h. die gegenüber herkömmlicher Software weit größere Bedeutung menschenlesbarer Information: *content is king.* Tim Berners-Lee wollte mit dem Web anfänglich ein simples, aber via Internet weltweit verfügbares Hypertext-System entwickeln und hatte dabei vor allem textuelle Information im Auge. Entsprechend treffen im heutigen Web die Tätigkeit eines *Autors* und eines *Programmierers* aufeinander, also krass ausgedrückt die Welt der Künstler und die der Ingenieure. Im vorliegenden Unterkapitel 5.2 wird dieser Gegensatz durch die geschichtliche Brille betrachtet, um das grundsätzliche Verständnis für Web-Anwendungen zu erhöhen und um wesentliche Fragen von deren

Design zu erörtern. In 5.2.2 bis 5.2.5 wird zunächst der Autorenaspekt behandelt, dann der softwaretechnische Aspekt, anschließend die Gemeinsamkeiten und Vorteile der Integration beider Aspekte und schließlich die Probleme, die bei dieser Integration verbleiben bzw. neu entstehen.

5.2.2. Informations-Design: eine Autorentätigkeit

Historisch wollen wir in diesem Unterkapitel die *Ära vor dem Web,* die *HTML-Ära* (vom Anfang des Web bis vor wenigen Jahren) und die im Fluss befindliche *XML-Ära* unterscheiden. Der Beginn der HTML-Ära war ausschließlich auf *Autoren* fixiert: nur Hypertext-Dokumente wurden unterstützt, wie der Name der sog. Web-Programmiersprache HTML ausdrückt: *Hypertext Markup Language,* eine Sprache für in Textdokumente einzustreuende Anweisungen (sog. *Tags*). Zunehmend unterstützte HTML weitere Medien: Graphik, zeitabhängige Medien wie Video und Audio usw., was sich im Begriff Hyper*media* spiegelt, der bisweilen im Unterschied, bisweilen synonym zu Hypertext verwendet wird; wir werden nachstehend Hypertext als Oberbegriff verwenden.

Das Konzept *Hypertext* ist älter als HTML, seine Grundideen wurden Ende des zweiten Weltkrieges von Vannevar Bush formuliert angesichts der entstehenden Fülle technischer Informationsträger, der Begriff selbst wurde später von Ted Nelson geprägt. Hypertext-Dokumente sind zusammengesetzt aus:

- Knoten, Links und Ankern wie schon in Kapitel 1 eingeführt,
- Geflechten und anderen Aggregaten. Geflechte bezeichnen zusammenhängende Knoten und Links, in der Ära vor dem Web *Hypertext-Dokumente* genannt. Beispiele für Aggregate sind Sichten (z.B. für Experten und Laien unter den Lesern), Pfade (vorgeschriebene Lesereihenfolgen) und Metaknoten (Geflechte, die sich wie ein Knoten in umgebende Geflechte einbetten lassen). Das simple Web der HTML-Ära unterstützte sie nicht; mit fortgeschrittenen Modularisierungskonzepten steigt ihre Bedeutung aber stark – Navigationsstrukturen wie »star-shaped naivgation«, vgl. 5.4.5, seien als ein Beispiel aus dem Interaktions-Design genannt.

Obwohl HTML zunächst nur den Autorenaspekt berücksichtigte, stellte es selbst diesbezüglich einen Rückschritt gegenüber verbreiteten Hypertextsystemen dar, sogar hinsichtlich der grundlegenden Vision nichtlinearer explorativer Dokumente (vgl. Kapitel 1). Nur aufgrund seiner Einfachheit sowie kostenloser weltweiter Verfügbarkeit konnte das Web seinen Siegeszug antreten. Wichtige Schwächen werden nachfolgend kurz zusammengefasst, soweit aus Design-Sicht relevant:

- HTML kann als (klassische) Dokumentbeschreibungssprache mit wenigen aufgepfropften Hypertext-Tags verstanden werden. Das verführt zur Missachtung des Atomaritätsprinzips von Knoten, »HTML-Dokumente« (eigentlich Knoten)

sind häufig mehrere Seiten lang, die Hypertext-Grundidee des nichtsequentiellen Lesens ist nur noch ansatzweise oder in Ausnahmefällen ausgeprägt

- HTML vermischt orthogonale Aspekte wie Hypertextstruktur (via Tags für Links und Anker), Dokumentstruktur (Überschriften, Listen etc.) und Layout (Hintergrundfarbe, Kursivschrift etc.)
- Das Web kennt zwar die in Kapitel 4 vorgestellte verteilte Softwarearchitektur mit Browsern und Servern, es fehlt aber eine »horizontale« Software-Architektur abstrakter Maschinen. Beispiele wären die klassische Dexter-Architektur, wo Inhalts- und Geflecht-Verwaltung sowie Präsentation getrennt werden, oder die im aktuellen Kapitel bzw. in Kapitel 3 vorgeschlagenen Schichtungen.
- HTML ist textzentriert: andere Medien kommen häufig nur als Ziele von Links vor (Sackgassen), viele Medientypen werden als Link-Quellen gar nicht oder erst seit jüngerer Zeit unterstützt;
- Die Weiterentwicklung verstärkte das erstgenannte Manko eher noch: Strukturierung und Formatierung innerhalb von Knoten wurden immer besser unterstützt, doch fehlen weiterhin wichtige Hypertext-Aspekte wie benutzerdefinierbare Knoten- und Linktypen, rückwärts-verfolgbare Links, getrennte Speicherung von Links (in Datenbanken) und Knoten, nichttriviale Zielanker usw.

Um im Gegensatz hierzu den Autorenaspekt von XML zu verstehen, muss zunächst noch aus der Entstehungsgeschichte von HTML berichtet werden. Letzteres geht zurück auf SGML, eine standardisierte generische »*markup language*« für die Welt von Druckereien und Verlagen. Generisch heißt, dass in SGML für eine ganze Klasse von Dokumenten (d.h. einen Anwendungsbereich und die darin üblichen Dokumente) zunächst die zulässigen Tags und die Regeln zu deren Verwendung definiert werden: es entstehen *document type definitions (DTDs)*. Ein *SGML-Parser* kann DTDs einlesen und Dokumente dahingehend überprüfen, ob sie einer DTD entsprechen. Für die Interpretation und Ausführung der via Tags gegebenen Anweisungen muss allerdings spezielle Software geschrieben werden. Verlage unterscheiden per DTD verschiedene Buch-, Zeitschriften- und Broschüren-Formate, Formulare und vieles mehr. HTML ist zunächst nichts anderes als eine SGML-DTD für das Format »Bildschirm«, ergänzt um Tags für Links und Anker als »aufgepfopfte« Hypertextfunktionalität. Neue HTML-Versionen entsprechen neuen DTDs. Browser der HTML-Ära sind keine SGML-Parser, sondern haben wenige DTDs (die unterstützten HTML-Versionen) fest einprogrammiert einschließlich der Art, wie sie Tags interpretieren und Befehle umsetzen. Auch das »Rendering« ist fest codiert, mit CSS wurden aber wieder verwendbare Layouts und ansatzweise die Trennung von Layout und Struktur möglich.

Die XML-Ära brach an, als Standard-PCs SGML-Parser »verdauen« konnten. Es war fast naheliegend, eine vereinfachte Version von SGML zu standardisieren und so die Fülle der Möglichkeiten einer generischen Markup-Sprache nutzbar zu machen. Mit

XML wurden Unmengen »einfacher Programmiersprachen« definiert als XML-DTDs (oder aktueller als XML Schemata), darunter eine Sprache zur Beschreibung von entfernten Prozeduraufrufen (SOAP), eine zur Beschreibung von Finanztransaktionen (XML-EDI), eine Entsprechung von HTML (XHTML) und vieles andere. Da via XML die Syntax, nicht aber die Semantik formal beschrieben wird, können moderne Browser zwar beliebige XML-Schemata und Dokumente *parsen*, aber (im wesentlichen) nur XHTML *ausführen*. Fast alle oben genannten Schwächen von HTML wurden inzwischen in verschiedenen XML-Standards adressiert, ob und wie sich diese teilweise konkurrierenden Standards verbreiten werden, bleibt offen.

Für das *Design* dokumentbasierter Web-Anwendungen, also den Autorenaspekt, sind aus dem vorgenannten einige Grundregeln festzuhalten:

- Im Zentrum des Informations-Designs sollten Geflechte stehen.
- Herkömmliche Dokumente sollten in atomare Knoten zerlegt werden.
- Separate Aspekte wie Layout und Inhalt, Knoten und Geflecht usw. sind konzeptuell zu trennen, auch wenn die Technologie dies nicht unterstützt.
- Die Technologie-Auswahl sollte fortgeschrittene Konzepte wie zentrale Link-Verwaltung zumindest im Entwurf unterstützen, möglichst auch im (dem Endanwender verborgenen) Content-Management-System, künftig bzw. im Intranet sogar in der Implementierungstechnologie selbst (Intranet deshalb, weil zentrale Linkverwaltung nach Intranets oder Anwendungen getrennt erfolgen sollte); XML-basierte Lösungen sind proprietären Ansätzen möglichst vorzuziehen.

5.2.3. Software Design: eine Programmiertätigkeit

Auch dieses Unterkapitel kann sehr gut durch die historische Brille betrachtet werden. Dazu sind die schrittweise Entwicklung zum »programmierbaren Web« und die Entwicklung des (verteilten) Programmierens zu unterscheiden. Nachfolgend werden Themen berührt, die in den Kapiteln 4 und 6 ausführlich behandelt werden, weshalb nur die im Design-Zusammenhang wichtigen Aspekte eingegangen wird.

Programmierbares Web: Der erste Schritt zur »Dynamik« waren Formulare in HTML. Mit ihrer Einführung nahm die Bedeutung von Skriptsprachen dramatisch zu, weil diese auf die Bedürfnisse Browser- oder Server-seitiger Verarbeitung und auf einfache Handhabung speziell zugeschnitten werden konnten. Browser-seitige Skripts werden i.a. verwendet, um in Abhängigkeit von HTML-Formulareingaben neue HTML-Seiten unmittelbar zu erzeugen; Browser unterstützen aber auch den Transfer der Eingaben zum Server, wo sie von Skripts anfänglich fast immer in Datenbank-Zugriffe verwandelt wurden, deren Ergebnis dann dynamisch in eine HTML-Seite verwandelt wurde. Datenbankzugriff ist noch immer häufig, aber auch viele andere serverseitige Anwendungen. JavaScript war die erste erfolgreiche Browser-seitige Skriptsprache. Server-seitig wurde von der »offenen Unix-Gemeinde« CGI und später

PHP vorgezogen, Microsoft favorisiert JScript und VBScript für die ActiveServerPages (ASP). Sun empfiehlt statt der einfach zu programmierenden, aber interpretierten und damit langsamen Skriptsprachen die Verwendung »ihrer« Programmiersprache Java in einer speziellen Variante für JavaServer Pages (JSP).

Egal wo und mit welcher Sprache neue HTML-Seiten erzeugt werden, das Skript oder Programm sollte vordefinierte Datenstrukturen und Operationen anbieten, um typische Elemente einer HTML-Seite wie Überschriften verschiedener Tiefe, Paragraphen, Listen und anderes zu erzeugen, mit Inhalt zu füllen und (als baumartige Struktur von Elementen) zu einer Seite zusammenzufügen. Grundlage hierfür ist praktisch immer das *Document Object Model DOM,* welches seit Jahren konsistent mit den aktuellen HTML-Versionen definiert und in den Skript- bzw. Programmiersprachen zur Verfügung gestellt wird.

Die Entwickler von Java waren ursprünglich ausgezogen, um »die Sprache des Web« einzuführen. Browser sollten nicht nur HTML *darstellen* können, sondern auch Java *ausführen.* Genau wie HTML-Dokumente sollte man sog. *Java-Applets* von Servern laden, statt eines dargestellten Dokumentes sollte die Benutzeroberfläche des Programms (Applets) am Browser-Fenster erscheinen. Entsprechend lag ein Hauptaugenmerk auf Sicherheitsaspekten, denn fremde Applets dürfen beim Endanwender keine unerwünschten Operationen ausführen.

Sun verband mit Java die Vision eines *»Pay-per-Use«*-Softwaremarktes, die bis heute kaum realisiert ist; Java selbst ist aber heute weit verbreitet – allerdings nur in moderatem Umfang als Applet-Programmiersprache, dafür als »normale« sowie als Server- und verteilte Programmiersprache. Außer Skripts und Applets werden in Browsern insbesondere Programme (oder Skripts) zur dynamischen Darstellung multimedialer Präsentationen ausgeführt, entwickelt z.B. mit Macromedia Flash.

Verteiltes Programmieren: Verteilte Programme im Internet setzten ursprünglich direkt auf TCP-Verbindungen auf; *Interprozess-Kommunikation* (kurz *IPC*), d.h. Botschaften-Austausch zwischen je zwei gleichberechtigten Partnern, dominierte. Für Multimedia gewinnt IPC (erweitert um Dienstgütegarantien zu »Streams«) wieder an Bedeutung, aber zunächst wurde es abgelöst durch entfernten Prozeduraufruf (*remote procedure call, RPC*) und damit einhergehend Client-Server-Architekturen. Der nächste Entwicklungsschritt, nämlich die Anpassung von RPC an objektorientierte Programmiersprachen, wurde irreführend als »verteiltes objektorientiertes Programmieren« bezeichnet und führte mit Technologien wie CORBA und Java's *»remote method invocation« (RMI)* endgültig zur weiten Verbreitung verteilter Programme. Irreführend ist obige Bezeichnung deshalb, weil objektorientierte Prinzipien wie Modularität und Flexibilität im Widerspruch zur RPC-Welt der monolithischen Klienten und Server steht, welche in CORBA und RMI weiterlebt. Verteiltes Objektorientiertes Programmieren, das den Namen verdienen würde, ist allerdings bis heute im akademischen Stadium steckengeblieben. Stattdessen gewinnt inzwischen *ereignisbasierte Kommunikation* (kurz *EBC*) an Bedeutung und damit einhergehend Publish-/Subscribe-Architekturen. Dabei bestimmt der Informationserzeuger, wann Kommu-

nikation erfolgt (»wenn ein neues Ereignis auftritt« → Push-Prinzip) und nicht mehr der Nachfragende (Pull-Prinzip). Nachfragende registrieren interessierende Ereignistypen via *subscription*. Diese Typen bestimmen den Empfängerkreis, nicht Verbindungen. Ursprünglich nicht vorgesehene Empfänger können problemlos hinzugefügt werden. In JavaBeans und sog. *message oriented middleware* ist EBC ansatzweise zu finden.

Der starke Trend zu Software-zentrierten Web-Anwendungen führte dazu, dass im Web die genannten Entwicklungen im Zeitraffer nachvollzogen wurden (vgl. WebServices). Da davon vor allem das funktionale Design betroffen ist, wird das Vorgenannte nicht nur nachfolgend, sondern auch in 5.5 aufgegriffen.

5.2.4. Verschmelzen von Informations- und Software-Design

Objektorientierte Softwareentwicklung kapselt sinnvoll zusammengehörige Daten gemeinsam mit den unmittelbar zugehörigen Operationen: den Methoden; puristische Ansätze verbieten sogar direkten Zugriff auf die Daten von außen: nur die Methoden sind für Objekt-Nutzer »sichtbar«. Offensichtlich haben Geflechte aus Objekten und den zwischen ihnen möglichen Aufrufbeziehungen große Ähnlichkeit mit Knoten-Link-Geflechten bei Hypertext. Einerseits könnte man behaupten: ein Hypertextknoten ist ein Objekt mit den Methoden »präsentiere menschenlesbare Daten«, »selektiere Anker« und »folge Link zu selektiertem Anker«. Andererseits verbergen sich hinter den Schaltflächen von HTML-Formularen beispielsweise JavaScript-Methoden, mit etwas Mühe lassen sich also viele allgemeine objektorientierte Software-Entwürfe als HTML-Dokumente realisieren; Applets schließlich sind per Definition Java-Objekte und Hypertext-Knoten gleichzeitig. Dass Objektorientierung und Hypertext verschmelzen, kann man gut anhand eines Knotens vom Typ »computergeneriertes Video« illustrieren – man denke an einen Link von einer Web-Seite auf eine Comic-Animation. Folgt man diesem Link, so wird die Animation abgespielt, aber man kann ggf. nicht erkennen, ob die Bilder wie ein Video von Datei abgespielt werden oder in Realzeit von einem Programm berechnet – ob man also ein Video-Dokument oder ein Video-Programm sieht. Macht es überhaupt noch Sinn, die Rolle von Autoren und Programmierern zu unterscheiden? Man ist geneigt *nein* zu sagen, doch gibt es bislang wohl noch sehr wenige Web-Entwickler, die beide »Kulturen« repräsentieren.

Offensichtlich ist weder die Technologie von Web und verteiltem Programmieren völlig verschmolzen, noch sind es die »Kulturen« der Menschen, die an der Entwicklung von Web-Anwendungen beteiligt sind. Für den Entwurf kann es aber über weite Strecken ratsam sein, eine künstliche, »technologiegetriebene« Trennung von Objekt- und Hypertext-Welt einfach zu ignorieren. Man entwirft dann Web-Anwendungen aus Elementen (Objekt-Knoten-Zwitter) und Referenzen (Links, die auch Methodenaufruf-Beziehungen sein können). Die zu verwendende Technologie kann man bei diesem Vorgehen später, ggf. von Fall zu Fall, unterscheiden. Dafür gibt es – stark verkürzt – folgende Hinweise bzw. Alternativen:

- *Elemente* können als statische, clientseitig (z.B. JavaScript) oder serverseitig (ASP, JSP) generierte Seiten realisiert werden, als Applets, Benutzeroberflächen verteilter objektorientierter Programme (Java), statische oder generierte Medien; was Benutzer im »Browser-Fenster« sehen, kann je nach Ausprägung präsentiertes (»gerendertes«) HTML, Medieninhalt, Formular oder Software-Benutzeroberfläche sein. Was er auswählen / anklicken / ausführen kann, sind *Referenzen.*
- *Referenzen* stehen für URLs in HTML bzw. XLinks in XML, wenn das Ziel eher Information als Programm ist und Inhalt und Adresse zum Programmierzeitpunkt (reines HTML) oder Präsentationszeitpunkt (dynamisches HTML) feststehen. Andernfalls repräsentieren Referenzen Fernaufrufe, z.B. entfernter Skripts (wenn Information zu »berechnen« ist) bzw. Methoden (wenn eher algorithmische Berechnungen erforderlich sind). Soll die Ziel*adresse* erst beim Anklicken berechnet werden, muss verwendete Softwaretechnik dynamisch berechnete Objektreferenzen unterstützen (die ist sogar in HTML über Umwege möglich, wenn Proxies zwischen Browser und Server geschaltet werden und das Konzept des *pointer redirect* realisiert wird).

5.2.5. Probleme und Grenzen des integrierten Web-Design

Der Weg zur Integration von Hypertext-, Informations- und Software-Design zum Web-Design ist insbesondere durch drei Problembereiche behindert: »kulturelle«, technologische und konzeptuelle. Auf der »kulturellen« Seite ist zu bemängeln, dass das wechselseitige Verständnis für die aufeinander treffenden Kulturen nicht nur bei den Web-Entwicklern in der Praxis unzureichend ist, sondern auch bei Forschern, Werkzeug- und Methoden-Entwicklern sowie Ausbildern. Auf der Technologie-Seite sind die verfügbaren Werkzeuge und Methoden unzureichend, außerdem spiegeln sie noch keineswegs die in 5.2.4 beschriebene Verschmelzung wider. Und auf der konzeptuellen Seite müsste das in 5.2.4 skizzierte integrierte Konzept schon an neueste Entwicklungen und Erkenntnisse angepasst werden, bevor es überhaupt ausgereift, in Werkzeuge und Methoden umgesetzt oder gar weit verbreitet ist. Das sollte aber einen guten Web-Entwickler nicht abgehalten, das skizzierte Modell – angepasst an seine Bedürfnisse – zu verinnerlichen und beim Web-Design zumindest konzeptuell anzuwenden sowie Werkzeuge auszusuchen, die es wenigstens ansatzweise auf ein maschinenlesbares Design abzubilden gestatten.

Aufgrund der gebotenen Kürze wird nachfolgend nur zum besseren Verständnis auf die genannten drei Problembereiche beispielhaft und stichwortartig eingegangen. Der Bereich der kulturellen Hürden ist dabei so vielschichtig, dass nur ein Beispiel herausgegriffen werden soll: während die Spezifika des graphischen Entwurfs von Web-Seiten im Kontrast zu Printmedien in Lehrbüchern und Bildungsangeboten behandelt werden, sind Spezifika des Informationsdesigns für das Web im Kontrast zu

anderen Medien kaum Gegenstand der Ausbildung. Etwas ausführlicher können beispiele technologischer Hürden genannt werden:

- Webseitengestaltung im Sinne von Informationsdesign und Softwaredesign sowie die jeweils anschließenden Entwicklungsphasen werden in marktgängiger Technologie üblicherweise *nicht* in integrierten Konzepten, Methoden und Werkzeugen unterstützt, schon gar nicht in der in 5.2.4 diskutierten Vielfalt.
- Der Übergang von formularbasierten interaktiven Benutzerschnittstellen zu Software mit graphischen Benutzeroberflächen stellt üblicherweise einen drastischen Technologiewechsel dar, weil Java-Applets (als Technologie für weiche Übergänge) in die Welt industrieller Softwarefertigung nur wenig Eingang fanden – und das, obwohl Java z.B. in Form von Java Server Pages (JSP) ansonsten viel allgemeiner zur »Programmiersprache des Web« wurde als geplant.
- Zu einigen wünschenswerten Varianten der in 5.2.4 skizzierten Design-Konzepte *Element* und *Referenz* gibt es keine entsprechende Technologie: beispielsweise gibt es in HTML keine Möglichkeit, das Ziel eines Links zum Zeitpunkt der Navigation (des Anklickens) zu berechnen (außer umständlich über Proxies).
- Die feingranulare Mischung des Autoren- und Programmier-Aspektes innerhalb eines Elements oder einer Referenz wird technologisch nicht hinreichend unterstützt; will man beispielsweise die Navigation durch ein Geflecht in anspruchsvoller Weise unterstützen, müsste man eine Software-Ebene für Navigation oberhalb des in HTML oder XML realisierten Geflechts ansiedeln. Das ist heute höchstens mit viel Eigenentwicklung unter Verwendung von wenig verbreiteten XML-basierten Spezialtechnologien zu erreichen.

Was den dritten und letzten Problembereich angeht, die konzeptuelle Seite, so ist insbesondere die Frage der Nachhaltigkeit und Wiederverwendbarkeit hervorzuheben, die in Unterkapitel 5.6 angerissen wird. Für näheres wird daher auf dieses Unterkapitel verwiesen.

5.2.6. Vorgeschlagenes strukturiertes Vorgehen

Die Unterscheidung zwischen Hypertext-, Informations- und Software-Design wird in den nachfolgenden Abschnitten immer wieder erforderlich sein, weil sie in Entwurfsmethoden und -werkzeugen sowie in der Praxis leider noch üblich ist. Soweit möglich wird sie aber vermieden zugunsten der integrierten Betrachtung des Designs von Web-Anwendungen. Für strukturierten Entwurf wird der in 5.1 eingeführte und in seinen Teilbereichen in 5.3 bis 5.5 beschriebene Ansatz gemäß Abb. 5-1 vorgeschlagen. Die drei Bereiche mit jeweils einer Komponentenseite und Geflechtseite sollen hier nochmals kompakt aufgelistet werden:

- *Präsentationsdesign:* hier steht auf der Komponentenseite die Ausgabe von Dokumenten, Medien und Daten (im Sinne eines Informationssystems oder im Sinne von Anwendungsdaten einer Softwarekomponente) im Vordergrund. Auf der Geflechtseite muss auf die Visualisierung, auditive oder multimodale Ausgabe von Geflechten sowie der aktuell vom Benutzer besuchten Komponente(n) eingegangen werden; Da diese Seite noch weitgehend im Forschungsstadium ist – abgesehen von der üblichen Darstellung von Navigationsleisten in Browsern –, fällt ihre Behandlung in 5.3 knapp aus.
- *Interaktions-Design:* die Einflussnahme des Benutzers auf die Web-Anwendung und der Kontrollfluss zwischen Mensch und Maschine sind Gegenstand dieser Ebene. Auf der Geflechtseite hat sich der Begriff *Navigation* eingebürgert, auf der Komponentenseite soll der verbreitete Begriff *Dialog* verwendet werden. Beide werden in Unterkapitel 5.4 behandelt.
- *Funktionales Design:* Unterkapitel 5.5 gibt eine Einführung in den Entwurf des Kerns von Komponenten und Geflechten und legt dabei einen Schwerpunkt auf die Sichtweise des Softwareentwicklers, weil diese bei den neuesten Kategorien von Web-Anwendungen (siehe Kapitel 1) stark in den Vordergrund tritt. Entsprechend wird die Komponentenseite weniger als Informationsdesign beschrieben denn als Entwurf von Softwarekomponenten. Analog wird auf der Geflechtseite die Komposition aktiver Komponenten zu Geschäftsprozessen und weitergehend personalisierten ubiquitären Web-Anwendungen verstärkt behandelt.

5.3. Präsentationsdesign

Im Rahmen des Präsentationsdesigns wird von »Mediendesignern« (vgl. 5.1) das Aussehen und ggf. die Struktur der Präsentation multimedialer Inhalte definiert. Basierend auf der ursprünglichen Idee *content is king* wurden im klassischen HTML Inhalte zusammen mit Formatanweisungen, Verknüpfungen und Programmen (Skripten) eng miteinander verzahnt spezifiziert. Modernes Präsentationsdesign verfolgt hingegen die konzeptuelle Trennung vom Inhalt der Web-Anwendung und dessen Präsentation. Der Inhalt der Web-Anwendung ergibt sich dabei aus der Komposition der explizit entwickelten multimedialen Inhalte auf der Komponenten-Seite und der implizit definierten Inhalte auf der Geflecht-Seite. Gutes Präsentationsdesign ermöglicht damit die flexible Anpassung der Präsentation an unterschiedliche kulturelle, technische und kontextuelle Anforderungen.

Webseiten, -anwendungen und ganze Websites werden darüber hinaus häufig neu strukturiert oder mit einem neuen visuellen Design versehen (vgl. Kapitel 1). In der traditionellen Webentwicklung bedeutete dies sehr häufig die manuelle Anpassung von hunderten oder gar tausenden von einzelnen HTML-Dokumenten. Die dazu notwendige Modifikation der HTML Dokumente setzte bei beteiligten Personen oft de-

taillierte HTML-Kenntnisse voraus. Zwar lassen sich einige Probleme durch geeignete Werkzeuge etwas abmildern, doch bleibt häufig ein beträchtlicher Teil manueller Änderungen in HTML erforderlich. Eine konsistente Modellierung aller Inhalte ist damit in größeren Entwicklungsteams meist nicht oder nur sehr aufwendig möglich.

Werkzeuge zur Erstellung von Web-Anwendungen können unter dem Blickwinkel ihrer Unterstützung von Präsentationsdesign in zwei Kategorien eingeteilt werden: herkömmliche *Seiten-Editoren* und eher fortgeschrittene *Content-Management-Systeme.*

Seiten-Editoren werden in der Regel für die ad-hoc-Erstellung von kleineren Internetauftritten verwendet. Vorteile sind die Ähnlichkeit mit Standardsoftware, was den Benutzer eine gewohnte Arbeitsumgebung empfinden lässt, und die Möglichkeit, Inhalt unmittelbar zu formatieren; andererseits sind für nicht-triviale Aufgaben

HTML-Kenntnisse notwendig, der Entwickler arbeitet auf Seitenebene d.h. verliert den Blick für die konzeptionellen Zusammenhänge, außerdem werden Layout, Navigation und Interaktion vermischt, was nur für triviale Anwendungen als Vereinfachung zu bezeichnen ist.

Im Vergleich dazu erlauben Content-Management-Systeme die Trennung der redaktionellen Tätigkeiten von der Layoutgestaltung. Sie dienen damit der leichten Pflege eines Internetauftritts. Grundlage dafür ist jedoch, dass die Struktur einer Internet-Präsentation abgebildet wird. Die Besonderheiten von Content-Management-Systemen sind, dass Spezielle Werkzeuge für verschiedene beteiligte Rollen, z.B. Grafiker, Redakteur u.s.w. angeboten werden, HTML-Kenntnisse i.a. nicht benötigt werden, Inhalt, Layout und Navigation getrennt werden, Inhalte in einzelnen Informationseinheiten spezifiziert werden und Workflows abgebildet werden können. Die vorgestellte Unterteilung von Seiten-Editoren und Content-Management-Systemen verschwimmt zunehmend, weil viele Seiten-Editoren in neueren Versionen auch einfache Funktionen von Content-Management-Systemen integrieren.

5.3.1. Präsentation von Knoten und Geflechten

Der Inhalt einer Webseite ergibt sich aus der Komposition explizit entwickelter multimedialen Inhalte auf der Komponenten-Seite und implizit definierter Inhalte auf der Geflecht-Seite (z.B. Navigationsmöglichkeiten).

Bei der Erstellung der multimedialen Inhalte steht dem Entwickler eine Anzahl von Gestaltungsmöglichkeiten zur Verfügung. Im Hinblick auf das gewünschte Konzept zur Trennung von Inhalt und Darstellung stehen diese Gestaltungsmöglichkeiten jedoch häufig in Konkurrenz. So ergibt sich in der Regel mit einer wachsenden Anzahl von Formatierungsmöglichkeiten eine abnehmende Flexibilität zur Adaptation des Inhalts an den jeweiligen Präsentationskontext. Als einfaches Beispiel hierfür seien die HTML-Elemente <b> und zur Formatierung von Text in einer fetten (Bold) Darstellung genannt. Auf Geräten, die keine fette Darstellung unterstützen, geht die Formatierung mit <b> meistens verloren, da keine Alternative spezifiziert

wurde. In XHTML 2.0 wird an Stelle des Elements <b> nur noch das Element unterstützt. Die Darstellung des Elements wird über das Präsentationsdesign gesteuert und kann damit an die technischen Möglichkeiten des Geräts angepasst werden, z.B. unterstreichen, wenn keine fette Darstellung möglich ist.

Während das grundlegende Aussehen der multimedialen Inhalte auf der Komponentenseite durch den Entwickler spezifiziert wird, ergeben sich auf der Geflecht-Ebene formatfreie Inhalte implizit aus dem Interaktions-Design und dem funktionalen Design. Die Darstellung dieser Inhalte obliegt damit vollständig dem Präsentationsdesign.

Als Beispiel für Aufgaben des Navigationsdesign wird nachfolgend das Präsentationsdesign für Navigationsoberflächen verwendet. Solche Navigationsoberflächen müssen helfen, die drei wesentlichen Fragen der Orientierung zu beantworten: (1) Wo bin ich? (2) Wo war ich? (3) Wohin kann ich gehen?

Präsentationsdesign ist ein recht typisches Beispiel einer Aufgabe, für deren Bewältigung der Rückgriff auf Erfahrungswissen wichtiger und praktikabler ist als die Anwendung formaler Methoden Für solche Aufgaben bieten sich Entwurfsmuster *(Patterns)* an. Zur Präsentation der Antworten auf die genannten Fragen kann beispielsweise auf das Konzept der Use-Patterns zurückgegriffen werden.

Die Frage »Wo bin ich?« wird dabei häufig mit Hilfe des Navigationsschemas »breadcrumbs« (Brotkrumen) nach dem Märchen von Hänsel und Gretel beantwortet. Das Konzept listet dazu alle übergeordneten Ebenen der Website auf (z.B. Startseite > Kontakt > Telefonverzeichnis).

Die Beantwortung der Frage »Wo war ich?« ist auf Grund der zustandlosen Webtechnologie und der primitiven Möglichkeiten der Verlinkung nicht ohne weiteres zu beantworten. Allgemein bekannt ist die in Browsern übliche Schaltfläche »Zurück« sowie Listen besuchter Seiten. Dieses einfache Beispiel zeigt die Notwendigkeit der Abstimmung zwischen Präsentations- und Interaktions-Design: bei Einkäufen im Web kann mit der Schaltfläche »Zurück« beispielsweise ein Einkauf nicht rückgängig gemacht werden, die entsprechende Irreführung der Benutzer wird nur in wenigen, gut »designten« Web-Anwendungen vermieden. Ein anderes wichtiges Beispiel ist Konsistenz, idealerweise im gesamten Web: die unterschiedliche Darstellung von bereits besuchten und noch nicht besuchten Links ist ein übliches Konzept des Präsentationsdesign. Jakob Nielsen empfiehlt dazu, die üblichen Linkfarben nicht zu verändern, weil Konsistenz hier vor Ästhetik geht.

Ein gebräuchlicher Ansatz zur Beantwortung der Frage »Wohin kann ich gehen?« besteht darin, alle Top-Level der Website aufzulisten. In Verbindung mit der »breadcrumbs«-Navigation und geeigneter Auszeichnung der von der aktuellen Seite aus erreichbaren Ziele erhält der Benutzer im Prinzip hinreichende Informationen über seine Position im Geflecht. Diese Auszeichnung erfolgt mindestens über das Hervorheben von Links im Text (oder anderen Medium), darüber hinaus empfiehlt sich die Kennzeichnung an ausgezeichneter Stelle (Navigationsleiste). Grafische Repräsentation des Geflechts und der aktuellen Position darin ist zwar wünschenswert, wird aber in der Praxis mangels Standards oder allgemein

in der Praxis mangels Standards oder allgemein anerkannter Methoden und Darstellungen kaum eingesetzt.

5.3.2. Ansätze für geräteunabhängige Entwicklung

Erweiterte Anforderungen an das Präsentationsdesign ergeben sich aus der zunehmenden Anforderung, in das Design von Web-Anwendungen den Trend zu einer Vielfalt von Web-fähigen Endgeräten einzubeziehen.

Das Spektrum dieser Web-fähigen Endgeräte umfasst insbesondere fast alle denkbaren Klassen mobiler Geräte, von sehr kleinen Mobilfunktelefonen mit WAP-Browser über Kombigeräte aus Mobilfunktelefonen und Organizern, bis hin zu TabletPCs mit drucksensitiven Displays. Derzeit noch weniger praxisrelevant sind kooperative und sehr große Endgeräte. Ausgehend von den technischen Merkmalen mobiler Endgeräte ergeben sich sehr unterschiedliche Darstellungs- und Interaktionsmöglichkeiten zur Verwendung mit Web-Anwendungen.

Im Präsentationsdesign werden diese Anforderungen im Rahmen von speziellen Aktivitäten zur Unterstützung der geräteunabhängigen Entwicklung von Anwendungen berücksichtigt. Im W3C beschäftigt sich damit zum Beispiel die *Device Independent Working Group* (DI WG). Die Aufgaben der DI WG umfassen Aspekte aus dem Bereich der Anwendungsentwicklung, der Adaption von Anwendungen in Bezug auf den Benutzerkontext und der unterschiedlichen Darstellung von Informationen. In 5.6.1 werden relevante Ansätze ausführlicher besprochen.

5.4. Interaktions-Design

Interaktions-Design betrifft den Schnittpunkt der visuellen, dynamischen, funktionalen und technischen Elemente von Web-Anwendungen. Hauptzweck des Interaktions-Design ist es, diese Elemente zu kombinieren und Spannungen zwischen ihnen auszugleichen, um den Benutzern ein einerseits interessantes und ansprechendes, andererseits konsistentes und verständliches Erlebnis zu bieten. Im vorliegenden Unterkapitel werden die Charakteristika des Interaktions-Design von Web-Anwendungen und die beim Entwurf oft zu berücksichtigenden »Design Forces« beschrieben.

Beim so genannten »Web-enabling« schon lange existierende Informationssysteme oder Anwendungen wird häufig der Fehler gemacht, Interaktions-Design auf Dialog-Design zu reduzieren. In diesem Fall werden Anwendungen im Web publiziert, indem einfach ihre Sichten in HTML-Seiten übersetzt werden; lediglich einige zusätzliche Aspekte wie gleichzeitiger Zugriff auf gemeinsam genutzte Datenbanken, werden hinsichtlich der Besonderheiten im Web angepasst. Dieser kurzsichtige Ansatz ignoriert die mächtigste Besonderheit des Web: die Möglichkeit, beliebig auf weitere Seiten zu verweisen.

Nachfolgend wird dagegen eine systematische Vorgehensweise vorgeschlagen, die Interaktion bei Web-Anwendungen in drei Aspekte einteilt: Organisation der Benutzeroberfläche, Navigation sowie Aktivitäten, die der Benutzer ausführen kann.

5.4.1. Organisation der Benutzeroberfläche

Der hier diskutierte Aspekt steht naturgemäß eng mit Präsentationsdesign in Beziehung, wird aber stärker von Interaktions- als von Präsentationsgesichtspunkten bestimmt. Die Benutzeroberfläche von Web-Anwendungen muss häufig eine große Anzahl von Informationen und darauf mögliche Operationen und Beziehungen darstellen. Die Herausforderung besteht nun darin, die Vielzahl von Aspekten geeignet darzustellen. Dazu empfiehlt es sich als erster Schritt, die Elemente nach den verschiedenen Interaktionskanälen zu unterteilen. Diese Unterteilung muss klar und konsistent für die gesamte Oberfläche sein. Die »Eingabe- und Interaktionsgruppe« sollte dabei im Ablauf der Web-Anwendung weitgehend unverändert bleiben, während die »Ausgabegruppe« i.a veränderlich sein sollte.

Ein häufiges Problem besteht darin, dass ein Knoten mehr Informationen enthält, als auf einen Bildschirm passt. Die Abwägung zwischen Gesichtspunkten des Präsentations- und des Interaktions-Design kann dabei von folgenden Fragen geleitet werden: Sollen die Abmessungen des Bildschirms Vorrang vor dem Konzept erhalten, dass Knoten atomare Einheiten (auch der Navigation) sind – soll oder kann ein Knoten in mehrere aufgespaltet werden? Kann zusätzliche Navigation eine Alternative zur übergebührlichen Nutzung von Rollbalken sein? Wie sind komplexes Verhalten der Benutzerschnittstelle und Portabilität abzuwägen?

Im genannten Spannungsfeld sind vier »technologische« Ansätze zu nennen:

1. Der gesamte Knoten wird als HTML zum Anwender geschickt. Die HTML-Seite enthält Skripte für die Auswahl anzuzeigender Elemente durch den Anwender. Die Verwendung von Skripts vermeidet weitere Anfragen an den Server.

2. Der gesamte Knoten wird ohne Skripte als eine umfangreiche HTML-Seite zum Anwender geschickt. Der Anwender navigiert auf der Seite mittels relativer Links innerhalb der Seite.

3. Eine partielle Sicht auf den Knoten wird zum Anwender geschickt. Die gesendete Seite enthält sinnvoll portionierte Teilinformation. Anwender navigieren zu weiteren Seiten, um die vollständige gewünschte Information abzurufen.

4. Der Knoten wird als XML zum Anwender geschickt. Der Browser des Anwenders wandelt das XML je nach Auswahl des Anwenders mittels XSLT in die passende Darstellung um.

Forces / *Implementierungen*	*Navigationssemantik*	*Portabilität*	*Usability*
HTML + Skripting	+	-	+
HTML + relative Links	+	+	-
Miteinander verknüpfte HTML Seiten	-	+	-
XML + XSLT	+	-	+

***Abb. 5-2** Tabelle verschiedener »Design Forces«, Auswirkungen auf die Implementierung.*

Miteinander verknüpfte Seiten vermeiden übergebührlichen Einsatz von Rollbalken (»Scrolling«), führen aber zu zusätzlicher Navigation und dadurch zu größerer Latenz, um die gleiche Information anzusprechen. [Niel97b] empfiehlt, Scrolling gegenüber zusätzlicher Navigation den Vorzug zu geben [Niel97b], allerdings wurde in 5.2.1 schon auf die Verletzung Hypertext-Konzepts atomarer Knoten hingewiesen; mit einem (in HTML als fehlend kritisierten) Aggregat-Konzept könnte beiden Ansprüchen Rechnung getragen werden.

Allgemein gültige Regeln für das Abwägen zwischen »Design Forces« scheitern am starken Einfluss von Spezifika der jeweiligen Web-Anwendung. Intranet-Anwendungen können z.B. Annahmen hinsichtlich verwendeter Browser treffen. Dagegen müssen E-Commmerce-Betreiber ihr Augenmerk auf die Portabilität legen, um jedem potentiellen Kunden den Zugang zu ihren Seiten möglich zu machen.

5.4.2. Navigationsdesign

Resultat des Navigationsdesigns sind einerseits die vom Benutzer zugreifbaren Elemente, andererseits die Navigationsstruktur. Die Elemente werden im einfachsten Fall zu Knoten. Die Navigationsstruktur definiert die Beziehungen der Knoten untereinander, diese Beziehungen werden später als Link-Anker in der Benutzeroberfläche sichtbar. In diesem Szenario definiert das Interaktions-Design die für die Navigation selbst nötigen Aspekte (Anker und URL) und die für die Orientierung des Benutzers nötigen Elemente.

5.4.3. Design der Darstellung eines Links: Der Anker

Als sichtbare Entsprechung von URLs müssen Anker sowohl Motivationen für die Aktivierung durch den Benutzer vermitteln als auch Konsequenzen. Da die HTML-basierte Realisierung des Webs das Konzept des Ankers mit dem des Links verschmelzt zu einem einzigen unidirektionalen Element <a>, verschmilzt auch die Semantik entsprechend. Benutzer können sich deshalb nicht darüber sicher sein, was die möglichen Konsequenzen (siehe Abb. 5-3) der Verfolgung eines Link-Ankers sind.

Link	**Semantik**	**Ungewissheit über die Bedeutung**
	Navigation	Repräsentiert der Link ein einzelnes Ziel oder mehrere Ziele? Liegt das Ziel auf der aktuellen Website oder außerhalb? Wird der neue Knoten im selben Fenster dargestellt oder in einem neuen? Wird anstelle echter Navigation lediglich der Text der aktuellen Seite verändert?
	Download	Welche Art von Dokument wird geholt? Sind die zur Präsentation des Dokuments erforderlichen Werkzeuge (z.B. Plug-Ins) vorhanden?
	Prozess	Wird der Link eine Aktion am Server auslösen? Wird es möglich sein, zurück zu navigieren bzw. die Aktion rückgängig zu machen?

***Abb. 5-3** Tabelle möglicher Konsequenzen der Verfolgung eines Links.*

Der Text eines Ankers sollte möglichst selbsterklärend sein. Es ist auch hilfreich, Links nach Link- oder Ziel-Kategorien zu klassifizieren, ggf. visualisiert über Icons als Teil des Ankers. Während solche Anker und Icons statisch bestimmt werden können, müssen dynamisch veränderliche Eigenschaften (z.B. ob bestimmte Medientypen geöffnet werden können) über in Seiten eingebettete Skripte ausgezeichnet werden.

5.4.4. Design der Interna eines Links: Die URL

Navigationsprozesse werden ausgelöst, indem in der Benutzeroberfläche Anker aktiviert werden. Diese repräsentieren Links (in HTML URLs), welche das Ziel des Navigationsprozesses bestimmen. Der Anker sollte daher selbsterklärend, prägnant, absolut adressiert und langlebig sein.

Die Einführung von XML war ein großer Schritt für Links und Anker. Die XML-Standards XPath, XPointer und XLink stellen in Kombination eine Infrastruktur für allgemeine Hypermedia-Funktionalität zur Verfügung und gehen damit weit über HTML-Links hinaus. Technologische Details findet der Leser in Kapitel 6, hier soll nur auf Design-relevante Aspekte kurz eingegangen werden.

Insgesamt ist es für Web-Anwendungsdesigner wesentlich, die erweiterten Fähigkeiten von Links in XML zu kennen. Beispielsweise können so genannte multidirektionale Links verwendet werden, um ein XML-Dokument mit mehreren Ressourcen über einen einzigen integrierten Link zu verknüpfen. Die verknüpften Ressourcen können dabei in beliebiger Reihenfolge angesprungen werden. So könnte etwa in einem Index ein Link auf alle Referenzen des Worts »Hypertext« zeigen. Der Anwender könnte dann diese Referenzen nach Belieben durchforsten und am Ende mithilfe des gleichen Links zum Ursprungstext zurückspringen. XML-Links sind außerdem selbstbeschreibend; das heißt, dass ein Link auch beliebigen Text enthalten kann, der die Ressourcen beschreibt, auf die er zeigt.

Da XLink sowohl externe als auch interne Links erlaubt, erleichtert diese Technologie die Einführung und kooperative Nutzung von Link-Datenbanken. Die immense Zahl von Links im globalen Internet führt zum Wunsch, diese Links effizient gemeinsam zu nutzen. XLink verfügt über einen Weg, um für Dokumente relevante Link-Datenbanken zu definieren. Ergänzende Möglichkeiten, um Links allgemein zur Verfügung zu stellen (Subscriptions oder Profile) werden zunehmend angeboten.

5.4.5. Navigation und Orientierung

Navigationshilfsmittel sollen helfen, die kognitive Belastung von Benutzern zu begrenzen. Dazu gibt es drei wesentliche Gruppen von Strategien:

- **Navigationsorganisation:** diese bestimmt die Gesamtstruktur der Navigation
- **Orientierungshilfen:** hier werden u.a. die im Präsentationsdesign in 5.3.1 behandelten Fragestellungen »Wo bin ich?« und »Wie bin ich hierher gekommen?« unter Interaktions-Gesichtspunkten organisiert
- **Link-Wahrnehmung:** hierunter sind insbesondere die in 5.4.3 behandelten Themen der Assoziation von Links mit Motivation und Konsequenz zu verstehen.

Offensichtlich ist im vorliegenden Abschnitt insbesondere der erste Punkt zu besprechen, der bisher noch nicht behandelt wurde. Dabei steht die sinnvolle Organisation des Navigationsraums im Zentrum des Designs inklusive der Unterstützung sinnvoller Navigationsaktivitäten, beispielsweise unter Vermeidung von Redundanz in der Navigation. Ein Beispiel für Redundanz sind separate Indizes (als Liste von Suchresultaten oder als Zugangsstrukturen), deren Elemente die Anwender nur dadurch ansprechen können, dass sie immer wieder zum Index zurück navigieren. Eine derartige Navigationsstruktur wird als »star-shaped navigation« bezeichnet. Eine leichte Verbesserung bringt schon die Möglichkeit, jeweils zum nächsten und zum vorigen Element zu navigieren, ohne zuvor zum Index zurückgehen zu müssen; schon dadurch wird die Zahl der durchschnittlichen Navigationsschritte (Zahl zu verfolgender Links) ungefähr halbiert werden.

Mit zunehmender Größe von Web-Anwendungen profitieren Anwender immer stärker von zusätzlichen Orientierungs- und Navigationshilfsmitteln. Als Beispiel sei ein Objekt genannt, das immer aktiv und wahrnehmbar ist und das als Index für weitere Navigationsobjekte (Knoten oder Unterindizes) dient. Dieses »Active Reference Objekt« bleibt inklusive Repräsentationen der Zielobjekte sichtbar und hilft dem Anwender entweder beim Sichten der Zielobjekte oder dabei, verwandte Objekte auszuwählen. Hinsichtlich Rückwärts-Navigation werden nicht nur Informationen über den Pfad zur aktuellen Position zur Verfügung gestellt, sondern auch Abkürzungen zu Indizes sowie Knoten auf dem Weg.

5.4.6. Strukturierter Dialog für komplexe Aktivitäten

Beim Design komplexer Web-Anwendungen sind häufig umfangreiche Abläufe sichtbar zu machen, welche die Benutzer durchführen sollen. Umfangreich kann dabei heißen, dass sich eine Aktivität über mehrere Seiten erstreckt. In diesem Fall sind drei Kategorien von Vorwärts-Navigation zu unterscheiden: (1) als Ergebnis der Navigation wird eine Aktion im Prozess-Ablauf ausgelöst; (2) die Navigation ruft »nur« eine weitere Seite auf, z.B. Seite 2 eines Formulars; (3) die Navigation führt zu einem Knoten, der nicht unmittelbar mit der Aktivität zu tun hat (Zusatzinformation o.ä.). Diese Alternativen bedrohen sowohl die Übersichtlichkeit für den Benutzer als auch die Konsistenz unterstützer Abläufe. Ein Beispiel ist der letzte Schritt einer Aktivität, welche verbindliche Absprachen zwischen Nutzer und Web-Anwendungen wiederspiegelt, der so genannte Checkout. Entscheidet sich der Nutzer, direkt vor Abschluss des Checkout auf eine unabhängige Seite zu wechseln, so ist zweifelsfrei zu klären, welchen Zustand die Aktivität hat. Ähnliches gilt wie schon erwähnt, wenn der Benutzer die Schaltfläche »zurück« betätigt, bevor der Checkout abgeschlossen ist.

Die angesprochenen Probleme zeigen, dass sich aus Sicht der Interaktion ein Geschäftsprozess in seinen Charakteristika stark von denen einer Hypertext-Anwendung unterscheidet. Abb. 5-4 fasst diese Charakteristika zusammen.

	Hypertext	**Geschäftsprozess**
Steuerung	Der Anwender steuert die Reihenfolge der besuchten Seiten. Die nächste Seite wird aufgerufen, indem ein Link verfolgt wird.	Der Prozess definiert, welche Aktivität als nächste ausgeführt wird. Aktivitäten werden oft sequentiell ausgeführt, der Kontrollfluss kann allerdings auch komplexer sein.
Verlassen der Seite / Aktivität	Eine Seite wird verlassen, indem ein Anker selektiert wird. Der Zustand der Seite ändert sich nicht. Eine Seite kann nicht beendet werden.	Wird eine Aktivität verlassen, so muss klar sein, ob sie vollständig beendet ist oder unter- bzw. abgebrochen wird. Das Konzept des Beendens von Aktivitäten ist elementarer Bestandteil von Geschäftsprozessen
Wiederaufnahme / Rückgängig machen	Zurückgehen auf eine Seite (z.B. über »Zurück« im Browser) bedeutet lediglich, dass die Seite erneut aufgerufen wird.	Zurückkehren zur Aktivität bedeutet, dass der bei Unterbrechung erreichte Zustand wieder aufgenommen wird. Rückgängigmachen einer Aktivität muss explizit aufgerufen werden.

Abb. 5-4 *Tabelle der Unterschiede zwischen Hypertext und Geschäftsprozessen*

Ansätze für die Realisierung von Geschäftsprozessen in Web-Anwendungen reichen von einfachem HTML (Prozesse sind Nebenprodukt der Navigation) bis zu Me-

thoden und Werkzeuge des Workflow-Management (Stichwort »workflow driven hypertext«). Im zweiten Fall erfolgt die Abbildung von Workflow-Interaktionen auf die Navigation in einer Web-Anwendung i.a. nicht automatisiert: Prozessdefinitionen beschreiben dort i.a. die Einbettung von Prozessen in Workflows, nicht aber die Benutzer-Interaktion mit Prozessen. Interaktions-Design muss daher versuchen, komplexe Aufgaben unter geschickter Ausnutzung der Navigationsmöglichkeiten abzubilden.

5.4.7. Zusammenhang mit Technologie und Architektur

Wie immer wieder betont, stehen Design, Architektur und Technologie bei der Entwicklung von Web-Anwendungen in engem Bezug. Dies trifft insbesondere auf den Übergang von einfachen zu komplexen Aktivitäten zu, der sich auf Technologieauswahl und Softwarearchitektur auswirkt und in der Evolution von Web-Anwendung bisweilen einen harten Übergang zu komplexeren Architekturen und leistungsfähigerer Technologie darstellt, welcher auch das Design nachhaltig beeinflusst. Entsprechend steht der vorliegende Abschnitt in engem Bezug zu den Kapiteln 4 und 6.

Simple Aktivitäten zum Abruf von Informationen können durch einfache »Three-Tier«-Architekturen realisiert werden, die mittels Vorlagen *(templates)* die passende HTML-Ausgabe zu den Anfragen der *clients* generieren (etwa auf Basis von ASP.NET, JSP, PHP, …). Bei solchen einfachen Architekturen sind Anwendungskontrolle und –logik in den Skript-Quelltext der Vorlagen eingebettet.

Sobald die darzustellenden Informationen komplexer werden, z.B. weil sie aus mehreren Quellen kombiniert werden, können Skripte äußerst umfangreich werden. Dann bietet sich an, Skripsprachen durch benutzerdefinierte *server-side tags* zu ersetzen. Diese machen es möglich, den für die Darstellung nötigen Code von der HTML-Seite zu trennen und zu verstecken. Allerdings ist bei benutzerdefinierten Tags trotz Trennung von HTML und Code die Kontrolllogik immer noch separat pro Knoten realisiert. Jeder für sich bestimmt die Folge-Sicht und leitet sie an die entsprechende Vorlage weiter. Dieses Konzept ist mit der Verwendung von Go-To-Befehlen bei der traditionellen Programmierung zu vergleichen, deren Nachteile für Wartbarkeit, Verständlichkeit und Modularisierung schon in den 60er Jahren kritisiert wurden. Entsprechend kann auch aus der Softwaretechnik eine Grundidee zur Überwindung von »Go-To-Programmierung« für komplexere interaktive Software übernommen werden, nämlich die Verwendung der, bereits in Kapitel 4 erläuterten, *Model-View-Controller*-Architektur *(MVC)* [KrPo88].

Bei komplexen Geschäftsprozessen stößt das reine MVC-Konzept aber an Grenzen. Es sieht keine Möglichkeit für Transaktionen vor, die mehr als eine Eingabe der Benutzer benötigen. Solche Transaktionen lassen sich folglich nicht über mehr als einen Knoten (eine Seite, Aktion) spannen. Das Konzept muss dann durchbrochen werden, komplexe Transaktionen werden in die Anwendungslogik (model) »verbannt«. Allgemein ist die Rolle des controller in »model 2«-Web-Anwendungen eher unterentwickelt: er empfängt Ereignisse vom Benutzer und bildet sie in die passenden Aufrufe ab. Technologie-Beispiele für die Abbildung von Events auf

rufe ab. Technologie-Beispiele für die Abbildung von Events auf Aufrufe sind *Apache Struts* [Apac03] als XML-basierte Konfigurationsdateien bzw. *DIE*-Einstellungen bei ASP.NET.

Auch die Hersteller brandmarken zu simple *controller*-Konzepte als Ursprung zusätzlicher Komplexität und Redundanz im Code [Sun03, Micr03b] – als Beispiel wird häufig das Problem angeführt, bei Anfragen wiederkehrende Parameter pro Seite (Knoten) getrennt authentifizieren, validieren und verarbeiten zu müssen. Allerdings werden unterschiedliche Schlüsse hieraus abgeleitet. In den J2EE Architecture Blueprints spricht sich Sun für Controller aus, die stärker spezialisiert sind, d.h. in FrontController und ViewDispatcher aufgeteilt. .NET nutzt das Vererbungs-Konzept in ASP.NET aus, um das allen Controllern gemeinsame Verhalten in einem gemeinsamen PageController zusammenzufassen, der in der Hierarchie weit oben steht. Angemerkt sei, dass der zweite Ansatz programmiertechnisch elegant (objektorientiert) erscheint, dass aber infolge Vererbung alle Knoten Kopien der selben Logik teilen. Letztere ist also in der Realisierung wieder in allen Knoten repliziert.

5.5. Funktionales Design

Auch beim funktionalen Design sind Technologieaspekte abzuwägen, die starken Einfluss auf die zu entwickelnde Web-Anwendung haben. Dabei ist die Verhältnismäßigkeit der Mittel zu wahren, andererseits sollten Anwendungen erweiterbar, skalierbar, wartbar etc. sein. Besondere Schwierigkeiten liegen im Zusammenspiel der Komponenten. Web-Anwendungen wie *news ticker* kommen meist ohne Transaktionsunterstützung aus, während *online shops* womöglich viele Produktphasen abbilden von der Konfiguration über die Bestellung bis zur Reparatur. Das erfordert Transaktions- und Workflow-Unterstützung sowie die Anbindung existierender *(legacy)* Datenbanken und Softwaresysteme; Ansätze finden sich unter dem in Kapitel 4 besprochenen Begriff »*Enterprise Application Integration*« (EAI).

5.5.1. Integration

Integration von Systemen kann auf drei Ebenen erfolgen, die als Unterebenen des funktionalen Design zu verstehen sind: Dabei unterscheidet man nach Daten-, Anwendungs- und Prozeß-Ebene.

Bei der Integration auf Datenebene wird sichergestellt, dass die Daten zwischen den Darstellungen der einzelnen Anwendungen transformiert und kopiert werden. Beispiele hierfür sind primitive Transformationsschritte zwischen dem Datenexport aus einer Anwendung und dem –import in eine andere oder die Verwendung von JDBC zur entsprechenden Kopplung von Datenbanken. Diese Vorgehensweise bezieht die Anwendungen selbst nicht ein und erlaubt z.B. keine Validation der Daten.

Bei der Integration auf Anwendungsebene (auch: Objektebene) erfolgt das Zusammenspiel über APIs und damit zeitlich wie semantisch enger verzahnt. Viele Details hängen allerdings von der zur Kopplung verwendeten Middleware ab, worauf im nächsten Unterkapitel noch eingegangen wird.

Als höchste Stufe wird oft die Integration auf Prozessebene angesehen, weil hier Geschäftsmodelle unabhängig von der verwendeten Infrastruktur modelliert werden.

5.5.2. Kommunikationsparadigmen und Middleware

Middleware wurde oben als Technologie zur Verknüpfung von Anwendungen erwähnt. Existierende Ansätze unterscheiden sich stark in Komplexität und Zielsetzung, wie schon aus der kurzen Beschreibung in Kapitel 4 und aus Unterkapitel 5.2.3 hervorgeht, wo *Interprozesskommunikation* (IPC), *Remote Procedure Call* (RPC), *Event-Based Communication* (EBC) inklusive *Message-Oriented Middleware* (MOM) sowie *verteilte objektorientierte Ansätze* kurz beschrieben sind.

Die an verschiedenen Stellen des Buches erwähnten XML-basierten Ansätze sollen hier in Vorbereitung auf die nachfolgenden Seiten nochmals in einem Abschnitt zusammengefasst werden: XML als aufkommende *lingua franca* des Internet ist nicht nur die Basis eines »besseren Web/HTML« und der portablen Spezifikation semistrukturierter Daten, sondern auch Basis neuer Standards für verteilte Anwendungen. Zu nennen sind insbesondere Simple Object Access Protocol (SOAP), Web-Service Description Language (WSDL) und Universal Description, Discovery, and Integration (UDDI). SOAP vermittelt Nachrichten und Aufrufe über verschiedene Internet-Protokolle wie HTTP, SMTP usw. WSDL dient der Schnittstellenbeschreibung und Adressierung von Web-Services und UDDI stellt eine Art Datenbank für das Publizieren und Suchen von Web-Services zur Verfügung (Details siehe Kapitel 6).

5.5.3. Verteilte firmenübergreifende Web-Anwendungen

Bei der softwareseitigen Realisierung von Web-Anwendungen gewinnt der *Verteilungs*-Aspekt zunehmend Bedeutung: wie heute Links auf entfernte Web-Seiten Gang und Gäbe sind, entsteht künftig verteilte Software aus dem verflochtenen Zugriff auf entfernte Web-Anwendungen. Dieser kann als Dienst-zu-Dienst-Kommunikation interpretiert werden, wobei hier der Begriff *Dienst* Funktionalität kennzeichnet, die über eine wohldefinierte Schnittstelle zur Verfügung gestellt wird. Zunehmend kommt Technologie für so genannte Web-Services auf Basis von XML zum Einsatz. So stellt *Ebay* nicht nur ein eigenes Authentifizierungssystem zur Verfügung, sondern unterstützt auch Microsofts Passport; google erlaubt das Einbinden seiner Suchfunktion in externe Anwendungen über SOAP. Die Konsequenzen dieses »grobgranularen offenen Komponenten-Marktes« sind für Systemdesigner gravierend. Der Einsatz extern entwickelter Funktionalität spart Entwicklungskosten bei ggf. höherer Qualität der Komponenten (Dienste), allerdings auf Kosten der »Kontrolle« über diese

Dienste. So haben Sicherheitslücken in Passport die Anfangseuphorie bereits gedämpft, bei sicherheitskritischen Anwendungen ist die Akzeptanzschwelle für externe Dienste sehr hoch. Andererseits kann gerade ein komponentenbasierter Ansatz helfen, durch den hohen Verwendungsgrad den Aufwand für qualitativ hochwertige Software zu rechtfertigen und Vertrauen in die Qualität von Komponenten zu etablieren. Daher ist mittelfristig mit einem Markt für Web-Services zu rechnen, der den im alltäglichen Leben angebotenen vielfältigen Dienstleistungen vergleichbar ist. Aufbauend auf XML und die Basistechnologien SOAP, WSDL und UDDI entstehen zurzeit weitere sich ergänzende, zum Teil aber auch konkurrierende Protokolle, wie sie für die Abwicklung Geschäften über Firmengrenzen hinweg unerlässlich sind. Abb. 5-5 gibt einen Überblick über die Abhängigkeit der einzelnen Protokolle zueinander.

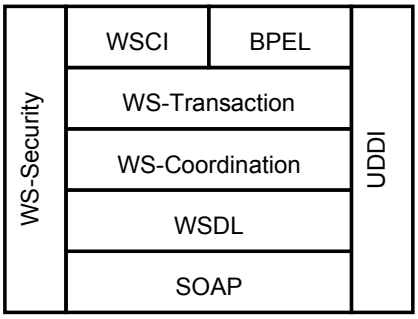

Abb. 5-5 Protokollstack für Web-Services

WS-Coordination [IBM02b] ist ein erweiterbares Framework zur Koordination von Aktionen verteilter Anwendungen, es stellt aber selbst keine Protokolle und Typen zur Verfügung. Diese werden in weiteren Spezifikationen festgelegt, wie z.B. in [IBM02c]. WS-Transaction stellt verschiedene Typen für WS-Coordination zur Verfügung. »Atomic Transactions« erlaubt die Koordination kurzer Aktionen auf Basis des 2-Phase-Commit-Protokolls. Der Ansatz eignet sich insbesondere zur Kapselung proprietärer Formate bisheriger transaktionsorientierter Systeme. Davon zu unterscheiden sind *Business Activities*, die für langlebige Aktionen vorgesehen sind und Ressourcen nicht längerfristig blockieren.
Das Web Service Choreography Interface [W3CaNielsen] und die konkurrierende Web Services Conversation Language [W3Cb] bieten die Möglichkeit, die an einem Service beteiligten Nachrichten und deren Struktur zu spezifizieren sowie die Reihenfolge, in der sie ausgetauscht werden müssen. BPEL4WS (Business Process Execution Language for Web Services) [IBM03] oder kurz BPEL ermöglicht aufbauend darauf die Beschreibung komplexer Geschäftsprozesse. Mit BPEL können Kontrollflüsse sowie Abhängigkeiten zwischen beteiligten Prozessen beschrieben werden.

Vor allem im Bereich der Beschreibung von Geschäftsprozessen in XML gibt es neben dem erwähnten BPEL4WS und (den nicht nur hierfür geeigneten) WSCI/WSCL eine Reihe weiterer herstellerspezifischer Protokolle. [KRKB03] gibt einen genaueren Vergleich der hier genannten sowie weiterer, nicht auf dem Web-Service-Protokollstack aufbauender Protokolle.

Ein weiterer nicht zu vernachlässigender Punkt bei Geschäften über das Internet sind die Sicherheitsaspekte, angedeutet in Abb. 5-5 durch den Block WS-Security. Authentizität, Vertraulichkeit, Integrität und Verbindlichkeit spielen dabei eine zentrale Rolle. In Kapitel 13 wird darauf unter besonderer Berücksichtigung von Web-Services genauer eingegangen.

Unternehmensübergreifend *(business-to-business, B2B)* entwickeln sich Web-Anwendungen zu riesigen, über viele Rechner verteilte Systeme. Dabei werden nicht nur unternehmensinterne Anwendungen integriert, sondern auch Anwendungen (Dienste) Dritter. Teilweise haben Unternehmen heute schon umfangreichen Zugriff auf Anwendungen (Dienste) von Zulieferern unter dem Stichwort *supply chain management (SCM)*. Mit Web-Services soll diese Möglichkeit standardisiert im Web angeboten werden. Forschungsansätze gehen schon dahin, die Auswahl der benötigten Dienste situationsabhängig und somit dynamisch vorzunehmen. Mit der Service Provisioning Markup Language (SPML), welche die Abrechnung von in Anspruch genommenen Diensten erlaubt, ist bereits ein erster Schritt in diese Richtung gemacht.

5.6. Ausblick: Nachhaltigkeit

Das so genannte *Post-PC* Zeitalter wird nicht mehr von einer einzigen Klasse von Endgerät (dem PC) dominiert, sondern ist gerade durch eine Vielzahl verwendeter Endgeräte gekennzeichnet. In den nächsten Jahren sind vor allem mobile Endgeräte für Web-Anwendungen von zentraler Bedeutung, wie schon in 5.3.2 erwähnt. *Nachhaltige* Web-Anwendungen müssen daher heute schon auf diese aktuelle Entwicklung vorbereitet werden, und zwar durch die Einbeziehung der Konzepte *Situationsbezogenheit* und *Geräteunabhängigkeit,* die in 5.6.1 und 5.6.2 behandelt werden. Da das erstgenannte Konzept noch eher im Forschungsstadium ist, beschränkt sich das entsprechende Unterkapitel darauf, die im Design situationsbezogener Web-Anwendungen zu beachtenden Aspekte zu erläutern. Unterkapitel 5.6.3 gibt schließlich einen Ausblick auf neue bzw. im Web-Engineering noch fehlende Konzepte, die Nachhaltigkeit von Web-Anwendungen zukünftig allgemein befördern könnten.

5.6.1. Situationsbezogene Anwendungen

Situationsbezogenheit *(Context-Awareness)* von Anwendungen bedeutet, dass benutzerspezifisches Wissen – die »Situation« (der Kontext) des Benutzers – verwendet wird, um sowohl die Interaktion als auch die Funktion der Anwendung möglichst optimal anzupassen. Weiters macht Situationsbezogenheit neuartige Anwendungen möglich. Als Beispiel seien die sog. *Location-Based Services* (LBS) genannt. Abhängig vom Ort können hier beispielsweise angepasste Informationen angezeigt werden, z.B. die nächstgelegenen Restaurants. Anspruchsvollere Varianten sind fast unbegrenzt denkbar, z.B. die Beachtung kulinarischer Vorlieben des Benutzers, die Einbe-

ziehung einer elektronischen Nahverkehrsauskunft bei Angeboten im weiteren Umkreis, etc. Für die breite Einführung derart anspruchsvoller Anwendungen ist noch eine Reihe von nicht nur technischen Problemen zu lösen, z.B. bei der Erschließung der benötigten Kontextinformationen. Telekom-Betreiber beginnen nur zögerlich, technische Maßnahmen und Schnittstellen von LBS für Dritte zur Verfügung zu stellen und verlangen meist hohe Preise. Aufgrund des großen wirtschaftlichen Potentials wollen Telekom-Betreiber situationsbezogene Dienste selbst oder mit ausgewählten Partnern anbieten. Fehlende Konkurrenz und in der Folge geringe Qualität der bisher angebotenen Dienste lassen aber an diesem Geschäftsmodell Zweifel aufkommen.

Ein weiteres großes Hemmnis bei der Einführung situationsbezogener Web-Anwendungen ist der Benutzerwunsch nach Vertraulichkeit; gerade LBS werden immer wieder dahingehend diskutiert, ob Dritte missbräuchlich die Aufenthaltsorte von Benutzern nachvollziehen können. Die Vorteile situationsbezogener Web-Anwendungen, wachsendes Vertrauen und bessere technische Sicherheit sollten mittelfristig auch dieses Hemmnis überwinden helfen.

Seitens der technischen Unterstützung Seite situationsbezogener Web-Anwendungen sind deutliche Forschritte zu verzeichnen. Wie in 5.5 beschrieben, existieren plattformunabhängige Protokolle zur Beschreibung und Kopplung von Web-Services. Auch gibt es bereits Betreiber von öffentlichen Datenbanken zur Registrierung von Web-Services und zur Suche nach geeigneten Diensten. Allerdings sind damit erst die Möglichkeiten geschaffen, Informationen über die Benutzersituation zu erfassen und Dienste zu verknüpfen. Zu lösen bleibt das Problem einer einheitlichen Beschreibung von Kontext. So erfasst z.B. ein Telekom-Betreiber den Ort des Benutzers über die Zell-ID und kann somit die GPS Koordinaten der Mobilfunk-Basisstation liefern, bei der der Benutzer eingebucht ist. Teilweise kann mittels Triangulation der Aufenthaltsort weiter eingegrenzt werden. Die Nahverkehrsauskunft benötigt für die Planung aber Straßennamen. Entsprechende Standardisierungsbemühungen sind im Gange, viele Details müssen derzeit aber noch im Rahmen von Forschungsprojekten geklärt werden. Betrachtet man das Beispiel eines Restaurantführers, so würden ubiquitäre Web-Anwendungen gastronomieweit akzeptierte Klassifizierungen von Restaurants bzw. Speisen voraussetzen, solche Fragestellungen erfordern Anstrengungen aus dem Bereich des »Semantic Web«. Die beiden Beispiele zeigen, dass umfangreiche Standardisierung eine der wichtigen Voraussetzungen für ubiquitäre situationsbezogene Software ist.

5.6.2. Geräteunabhängige Anwendungen

Die Hersteller von Web-Engineering-Werkzeugen haben das Problem geräteunabhängiger Anwendungen längst erkannt, suggerieren aber zu optimistisch, das Problem sei mit der Transformation einer allgemeinen (XML-basierten) Darstellung auf verschiedene Markup-Sprachen der Endsysteme zu lösen (HTML, WML etc.). Unterschiedliche Implementierungen der Benutzeragenten, Vorgaben durch Betrieb-

systeme sowie physikalische und technische Unterschiede der verwendeten Geräte stellen in der Praxis große Hindernisse dar. Als Beispiele seinen genant: teilweise unvollständige Implementierung und unterschiedlichen Versionen von HTML, unterschiedlicher Umfang verfügbarer Benutzerinterfaceelemente (z.B. Listboxen, Radio-Buttons, usw.), Größe und Ausrichtung des Displays, zur Verfügung stehende Rechenleistung und Netzwerkbandbreite; diese Beschränkungen führten beispielsweise zu Entwicklung von WML als spezifischer Markup-Sprache für mobile Endgeräte.

Grundsätzlich kann man bei der Entwicklung von geräteunabhängigen Anwendungen zwei alternative Ansätze unterscheiden. Der erste »minimalistische« Ansatz reduziert die zu verwendenden Elemente auf ein Minimum, von dem angenommen wird, dass es von allen potentiellen Endgeräten unterstützt wird. Entscheidender Nachteil dieser Vorgehensweise ist, dass sowohl die Benutzbarkeit als auch das Aussehen der Anwendungen bewusst reduziert werden. Beim alternativen Ansatz, der Entwicklung adaptiver Anwendungen, wird der Kontext des Benutzers auf mehreren Ebenen einbezogen. Geräteunabhängigkeit berücksichtigt hier nicht nur die Aspekte des eigentlichen physikalischen Endgeräts, sondern geht nahtlos in Situationsabhängigkeit gemäß 5.6.1 über. Für die Realisierung stehen unterschiedliche Technologien und Standards zur Verfügung, weitere werden von Standardisierungsgremien entwickelt. Für die Beschreibung und Übertragung des Benutzer- und Gerätekontextes definierte die »Device Independence Working Group« (DI WG) des W3C, aufbauend auf den Arbeiten der »CC/PP Working Group«, den CC/PP Standard und ein dazugehöriges Übertragungsprotokoll. Mit Hilfe von CC/PP können Kontextprofile einheitlich beschrieben und zum Anwendungsserver übertragen werden.

Als bereits verfügbare Alternative zur Gewinnung von – allerdings proprietären – Kontextinformationen verwenden heutige Adaptionsmodule die Interpretation von Bestandteilen des HTTP Protokolls (z.B. die Kennung des Benutzeragenten). Für die eigentliche Adaption der Anwendung kommen ebenfalls unterschiedliche Technologien zum Einsatz. Für die serverseitige Adaption verwendet man dazu häufig Skriptsprachen (z.B. JSP/ASP) in Kombination mit Transformationsmodellen (XML, XSL, XSLT), während für die clientseitige Transformation Standards wie XHTML und CSS in Verbindung mit Skripting (Javascript) zur Verfügung stehen.

Einen alternativen Ansatz für die Adaption von Anwendungen verwenden sog. Transcoding Server. Die Adaption überführt dabei eine meist generische Beschreibung der Anwendung in ein neues Zielformat. Anfänglich wurde versucht, existierende HTML-basierte Beschreibungen ohne semantische Zusatzinformationen in ein neues Zielformat (z.B. WML) zu überführen, was kaum zufrieden stellende Ergebnisse lieferte. Heute erfolgt Adaption auf unterschiedlichen Ebenen der Web-Anwendung. So wird zum Beispiel (1) die Komplexität der Gesamtanwendung an die Situation des Benutzers und das zur Verfügung stehende Gerät angepasst (Adaption der Businesslogik), (2) der Inhalt und das Layout der einzelnen Seiten optimiert und (3) die Präsentation in Bezug auf die adressierte/vorhandene Implementierung der Markup-Sprache optimiert.

5.6.3. Wiederverwendbarkeit

Der Bereich Wiederverwendbarkeit ist bislang in Technologien für Web-Anwendungen deutlich zu kurz gekommen, teilweise fehlen selbst konzeptuelle Voraussetzungen im Vorfeld der Wiederverwendbarkeit. Dass entsprechende Konzepte deutlich in den Entwurf hineinspielen, kann am Beispiel objektorientierter Programmierung bzw. objektorientierten Entwurfs einfach nachvollzogen werden. Für den Bereich von Web-Anwendungen sollen nachfolgend drei Bereiche der Forschung und Entwicklung genannt werden, die Wiederverwendbarkeit substantiell zu befördern versprechen – der exemplarische Charakter versteht sich von selbst.

Konzepte für Geflechte: Weder für das Software Design noch für das Informationsdesign von Web-Anwendungen werden Konzepte hinreichend unterstützt, womit *Geflechte aus Elementen und Referenzen* elegant entworfen werden könnten. Erforderlich wären insbesondere Konzepte Aggregate (vgl. 5.2.2) und Hierarchien davon.

Ausgereifte Typkonzepte: Typen von *Elementen* und *Referenzen* wie in 5.2.4 eingeführt waren in der Ära vor HTML im Informationsdesign nicht unüblich und werden mit XML wieder möglich (wenn auch noch nicht gängig). Im Software Design (vgl. UML) werden Typen (Klassen) von *Elementen* recht zufrieden stellend unterstützt, in der objektorientierten Softwaretechnologie durch Klassenhierarchien und moderne Komponenten-Konzepte ebenfalls. Einheitliche Konzepte zur Typisierung von Elementen und Referenzen im integrierten Web-Design sind bislang aber in keiner Design-Notation zu finden. Während Typkonzepte für Elemente und Referenzen (auch für Anker, die hier aus Platzgründen nicht betrachtet werden) nicht unschwer zu entwickeln sind und von Web-Designern schon eingesetzt werden können, sind Typkonzepte für Geflechte noch als Herausforderung für die Forschung zu sehen (vgl. [MüHK98a, MüHK98b]). Insbesondere das Design von wieder verwendbaren Geflechten müsste in dieser Forschung ein Anliegen sein, so dass z.B. *best practice* im Entwurf von umfangreichen Websites, von Firmenauftritten, von Web-Based-Training-Konzepten u.a. modelliert werden könnte. Solche Konzepte für Geflecht-Design müssten insbesondere die *Dynamik,* also die Veränderung der Zahl und Anordnung von Elementen und Referenzen, sowohl über die Lebenszeit eines Geflechts als auch im Unterschied mehrerer Inkarnationen eines Geflecht-Typs, geeignet zu beschreiben gestatten. RDF ist bestenfalls ein rudimentärer Ansatz in die hier beschriebene Richtung.

Alternative Kompositionskonzepte: Geflechte wie bislang behandelt repräsentieren gerichtete Graphen. Angesichts fortgeschrittener Programmierkonzepte wie der ereignisbasierten Kommunikation aber auch fortgeschrittener Hypertext-Konzepte wie n-ärer Links (die in XML-basierten Standards teilweise unterstützt werden) ist zu vermuten, dass das Konzept des »Geflechts« als solches, so wie bisher betrachtet, erweitert werden müsste. Arbeiten in diese Richtung, die hier erwähnt werden sollten, sind den Autoren allerdings bisher nicht bekannt.