Zusammefassung WBD

Von Simon Flüeli

Internet

- Inter(connected) Net(work)
- Grosses Netzwerk
- Virtuelle Verbindung PC zu PC
- Weltweit einmalige numerische Adresse (IP Adresse)
- Symbolische Namen zur einfachen Handhabung
- DNS übersetzt Namen in IP
- Internet Dienste
    - www
    - Telnet
    - SSH
    - FTP / SFTP
    - Email

Domains

Level	Bedeutung
Top-Level Domain	ch
Second-Level Domain	Zhaw
Third-Level Domain	www

Telnet, SSH (Secure Shell)

- Bidirektionale, byteorientierte Kommunikation zw. 2 PC
- Textkonsole, um an entf. PC anzumelden + Befehle ausführen
- Telnet: Übertragung im Klartext (Port 23)
- SSH: Übertragung verschlüsselt (Port 22)

FTP

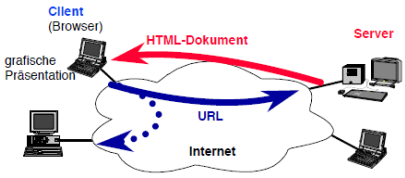
- File Transfer Protocol
- Dateiübertragungsprotokoll (bidirektional)
- Zugriffsrechte (Authentifizierung)
- Übertragung beliebiger Dateien
- Konvertieren von Datenformaten (Textdaten) in heterogener Umgebung
- Plattformübergreifendes Kopieren von Textdateien evtl. falsche Darstellung (Zeilenende: CRLF/CR/LF)

WWW

- Adressierung: Wie adressiere ich ein Dokument?
URI, URL, URN
- Protokoll: Wie komme ich an ein Dokument?
HTTP, TCP, IP
- Sprache: Wie ist ein Dokument aufgebaut?
HTML/XHTML, SVG

WWW-Funktionsweise

- Client-Server Prinzip
- Client stellt Verb. zum Server her und verlangt ein Dok.
- Auswahl Server: Via URL (Uniform Ressource Locator)
- URL: Protokoll://IP:Port/Pfad/#Textmarke
- Defaultportnummer: 80
- Server schickt Client Dokument und unterbricht Verbindung
- Client stellt Dokument auf Bildschirm dar



HTTP Protokoll

- Textbasiert
- Es werden Textnachrichten und Dateien ausgetauscht
- Antwort: Datei (normalerweise)
- Für Server ist jede Anfrage neu
- HTTP-Protokoll ist zustandslos (stateless)

Protokolle

Protokoll	Beschreibung
HTTP	Webseiten sollen unkompliziert und von vielen Leuten gesehen werden
HTTPS	Online Banking, Auto-Reservation, Webmail, Webshop
Telnet	Zugriff ohne Authent. direkt auf CMD eines Rechners darf es nicht geben
SSH	Verschlüsselte Version von Telnet. Standard für CMD-Zugriff auf entf. Rechner
FTP	Software, Treiber usw. können unverschlüsselt und ohne Authent. vom Server kopiert werden
Secure FTP & SFTP (SSH File Transfer Protocol)	Verschlüsseln der Anmeldedaten und Nutzdaten - SFTP mehr verbreitet
SCP (Secure Copy)	- SCP wie CP aber via SSH - Verschlüsselt & passwortgeschützt zw. zwei entfernten Rechnern

Hands on FTP

- Für Aufbau d. Verbindung zwei Angaben erforderlich:
Rechnername (dublin.zhaw.ch)
Portnummer (Webserver: 80; Telnet: 23)
- Kombination IP und Port: Socket

HTML

Hypertext Markup Language

```
<html>
<head>
<title>Beschreibung der Seite</title>
</head>
<body>

</body>
</html>
```

Drei Doctype Varianten: **strict**, **transitional** (Kompatibilität mit früheren Versionen (enthält veraltete Elemente), **frameset**

CSS

Definition:

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

```
h1 {font-family: helvetica;
    color: blue;
}
```

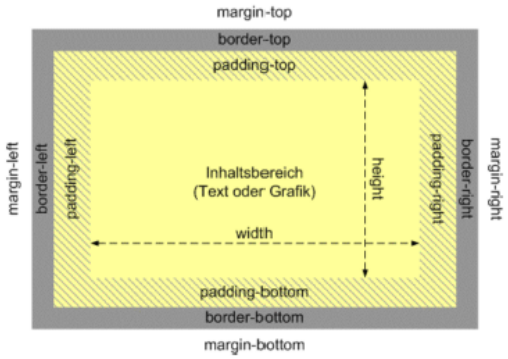
Selektor Eigenschaften Werte Deklaration

Code	Beschreibung
div i	Alle i, die unterhalb eines divs liegen
div > p	Alle p, die direkt unterhalb eines divs liegen
div * b	Alle b, die mindestens zwei Ebenen unterhalb eines divs liegen
div + p	Alle p, die unmittelbar auf ein div folgen
p[title]	P-Elemente, die ein Attribut title haben
p[title=Text]	P-Elemente, die ein Attribut title mit dem Wert Text haben
p[title~=Text]	P-Elemente, die ein Attribut title haben, in dessen Wert das Wort Text vorkommt (Werte durch Leerzeichen getrennt)
p[lang|=en]	P-Elemente, die ein Attribut lang haben, das mit en beginnt, gefolgt von einem Bindestrich und dem Rest des Attributwerts
td.Preis:before {content: „Preis:„}	Vor dem Inhalt des td der Klasse Preis
td.Preis:after {content: „, -€“]	Nach dem Inhalt des td der Klasse Preis
Grössenangaben	
Bildschirm	
px	Pixel
em	Aktuelle Schriftgrösse
%	Prozentual (vergrössern oder verkleinern)
Druck	
in	Inch
pt	Punkt(72pt = 1in)
cm	Zentimeter

Geordnete Listen
Nummerierung Standardmässig mit arabischen Zahlen

Ungeordnete Listen
Standardmässig mit dicken Punkten
Anpassungen mittels:
list-style-tyle, list-style-position, lyst-style-image

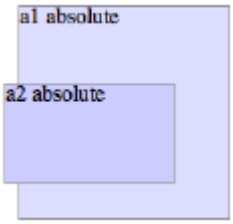
CSS Box Model



Margin
Bei aufeinanderfolgenden Elementen, die jeweil einen Abstand zueinander definieren, wird nur der **grössere** der beiden Abstände berücksichtigt.

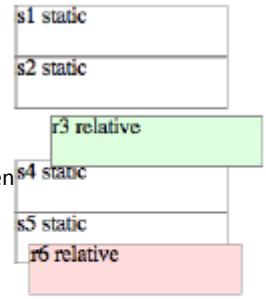
Margin: top right bottom left
Margin: top rightleft bottom
Margin: topbottom rightleft
Margin: topbottomrightleft

Positionierung
position: absolute
Absolute Positionierung
Gemessen am Rand des nächst höheren Elternelements, das nicht statisch positioniert ist
Scrollt mit dem Inhalt
Benötigt keinen Plaz im normalen Elementfluss
top, left, right, bottom möglich

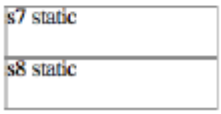


position: fixed
Absolute Positionierung
Gemessen am Browserfenster
Bleibt beim scrollen stehen
Z.B. Für Menus, die immer an der gleichen Stelle sein sollen

position: relative
Relative Positionierung
Gemessen an Normalposition des Elements
Element benötigt Platz entsprechend normalem Elementfluss
Es kann aber relativ zur Normalposition verschoben werden



position: static
Standardeinstellung
Position im normalen Textfluss



Float
Element soll von nachfolgendem Element umflossen werden
left: Element steht links und wird rechts umflossen
right: Element steht rechts und wird links umflossen
none: Kein Umfluss (Standard)

Clear
Ermöglicht es, einen Textumfluss abzubrechen
left: Erzwingt bei float:left die Fortsetzung unterhalb
right: Erzwingt bei float:right die Fortsetzung unterhalb
both: Erzwingt in jedem Fall die Fortsetzung unterhalb
none: Erzwingt keine Fortsetzung unterhalb (Standard)

Stylesheet Rangfolge

1. Benutzer Stylesheet mit !important
2. Autoren Stylesheet mit !important
3. Autoren Stylesheet
4. Benutzer Stylesheet
5. Browser Stylesheet

Spezielle CSS

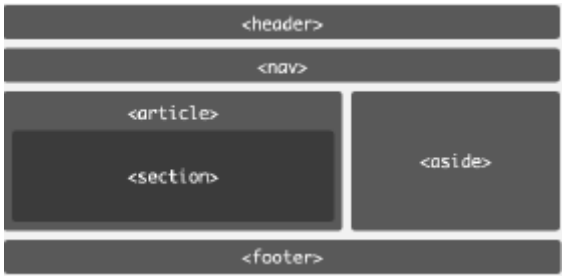
```
<link rel="stylesheet" media="print"
href="druck.css"/>
```

Transparenz

opacity
0 = durchsichtig, 1 = deckend
.halbtransparent {opacity: 0.3;}
color: rgba(0,0,255,0.3);

HTML 5

Seitenstruktur von HTML 5



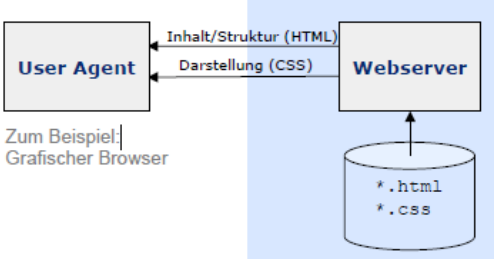

```
<body>
  <header>...</header>
  <nav>...</nav>
  <article>
    <section>
      ...
    </section>
  </article>
  <aside>...</aside>
  <footer>...</footer>
</body>
```

Doctype, Zeichensatz, XML Variante

```
<!doctype html>
<meta charset="UTF-8">
<?xml version="1.0" encoding="UTF-8" ?>
```

Statische Seiten

Bestehen aus HTML und CSS
Werden als fertige Dateien aus der Filesystemhierarchie geladen und vom Browser einmalig dargestellt



Dynamische Webseiten

Zusammenspiel diverser Technologien
Clientseitige Technologien (JavaScript, DOM, Ajax)
Serverseitige Technologien (SSI, CGI, JSP, PHP, ASP)

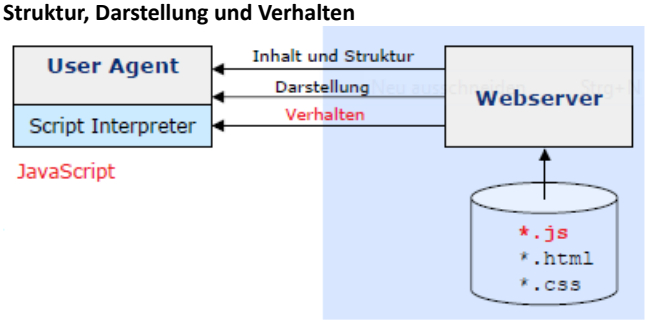
Client-Side Technologien

Erweiterung der Möglichkeiten von HTML und CSS:

- Reaktion auf Benutzereingaben
- Hinzufügen interaktiver Elemente
- Änderung von Inhalt und Darstellung
- Animationen
- Zugriff auf definierte Ressourcen des Client-Computers (Kamera, Mikrofon, GPS, Filesystem)

--> Zusätzliche Eigenschaft der Website: Verhalten

Struktur, Darstellung und Verhalten



JavaScript

```
<script type="text/javascript">
</script>

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

Objekt-basiert und event-gesteuert

- Keine Klassen (keine Vererbung/Polymorphismus) aber Prototypen
- Nicht statisch typisiert
- Objekte können dynamisch geändert werden
- Funktionen sind Objekte

Ist eine vollständige Programmiersprache

- wird jedoch selten für „ganze“ Programme eingesetzt
- eher kleine Funktionserweiterungen von HTML-Seiten
- kann serverseitige Programmierung nicht komplett ersetzen

Ist ziemlich plattformunabhängig

- es gibt jedoch Implementierungsunterschiede in Browsern
- eventuell durch Benutzer ausgeschaltet

Wozu?

- Überprüfen von Formulareingaben (Validierung)
- Animationen/Effekte (dynamische Menüs, Navigation in Bildergalerien, aktualisieren von Inhalten (Ticker, Uhr, ...)
- GUI ähnlich Desktop-Applikationen
- Steuern von Multimedia-Inhalten

Sicherheit

- Kein Zugriff auf lokales System (Dateien)
- Trotzdem gelegentlich Sicherheitsprobleme

DOM Scripting
- Zugriff auf Elemente einer Seite über Document Object Model
- Z.B. Zum dynamischen Verändern des Seiteninhalts (auslesen / setzen / hinzufügen)
- Normalerweise auf die Manipulation von X/HTML und CSS mit Hilfe von JavaScript bezogen

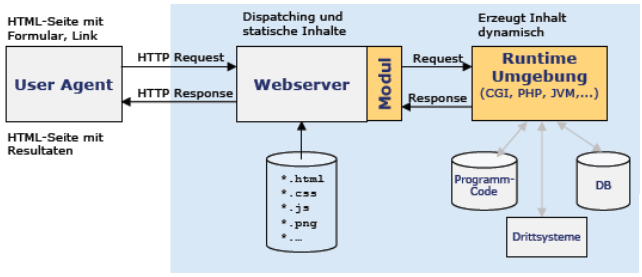
Zunächst notwendig dafür:
- Zugriff auf Elemente der HTML-Struktur
- Möglichkeit zum Erzeugen neuer Knoten

Dies geht mit Hilfe einiger Methoden des document-Objekts: getElementById(), getElementsByName(), getElementsByTagName(), createAttribute(), createElement(), createTextNode()

Server-Side Technologien
Laufen auf Web-/Application Server
Ziele
- Dynamisches Erzeugen der Webseiten auf Anfrage
- Ausgabe variabler Inhalte
- Auswerten von Formulardaten und Reaktion auf Benutzereingaben

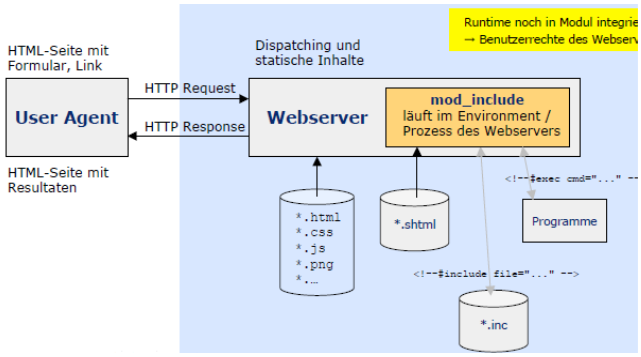
Technologien
- Server Side Includes (SSI)
- Common Gateway Interface – CGI (Shell, C/C++, Perl)
- PHP, Active Server Pages (ASP), Java Server Pages (JSP)

Schnittstellen zu weiteren Systemen
- Zugriff auf Datenbanken
- Einbindung von Drittsystemen (ERP, E-Mail, ...)
- Konfiguration und Steuerung (Drucker, Router, ...)
- Einbinden von Internet-Services (Facebook Twitter, ...)
- Webservices (SOAP, XML-RPC, REST, ...)



- Inhalt wird nicht als fertige HTML-Datei geladen, sondern von Programm/Script generiert anhand Angaben aus Request
- Programme laufen in Runtime-Environment, welche Umgebung für Programm bereitstellt und dieses ausführt (z.B. Tomcat, JVM)
- Für Kommunikation mit Laufzeitumgebung wird Modul im Webserver benötigt
- Kommunikation zw. Modul und Runtime-Environment ist nicht fest vorgeschrieben, sondern jeweils für Anforderungen der spezifischen Umgebungen definiert und umgesetzt (z.B. CGI, AJP)
- Welches Modul angesprochen wird entscheidet der Dispatcher des Webservers anhand der Konfiguration (z.B. Dateiendung)

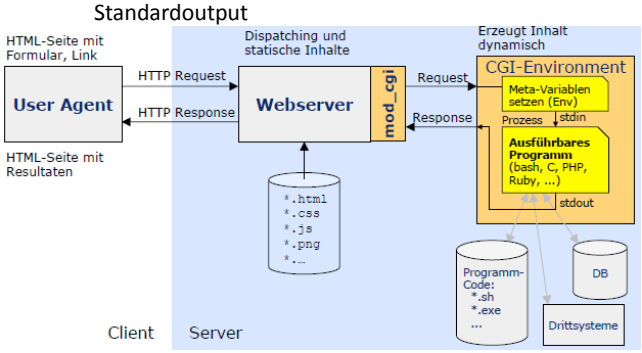
Server Side Includes (SSI)
- Kommandos in HTML-Comments verborgen und mit # markiert
- Webserver prüft Dateien mit Endung .shtml auf SSI Kommandos und führt diese aus. Muss auf Server aktiviert sein



Vorteile
- Einfaches Mischen von Design und Applikation
- Auf mehreren Seiten vorkommende Teile werden zentralisiert

Nachteile
- Sehr limitiert in Möglichkeiten
- Seiten haben kein Createdate und können nicht gecached werden
- Bei manchen Servern laufen SSI-exec mit dem Web-Server-Benutzerkonto
- Bei vielen Servern aus Sicherheitsgründen gesperrt

CGI Common Gateway Interface
- Allgemeine Schnittstelle zw. Webserver und externem Programm
- Für div. Programmiersprachen verwendbar
- Häufig Shell-Scripts, Perl, Python, C, PHP, Lisp
- Übergaben von Daten an Script über Umgebungsvariablen, Kommandozeilenparameter oder Standard Input
- Ausgabe der dynamisch erstellten Datei über Standardoutput



Server-Side Sprachen
Verbreitete Server-Side Sprachen
- Perl, Ruby on Rails, PHP, JSP, ASP.NET

Entscheidung welche
- Zielplattform Datenbasis
- Verfügbarkeit personeller Ressourcen und Know How
- Persönlicher Geschmack
- Budget

Perl

- Plattformunabhängig, Open Source
- Direkt in Apache eingebunden
- Mächtige Sprache, viele Anwendungsgebiete
- Syntax wirkt auf Einsteiger kryptisch
- Umfangreiche Erweiterungs-Module verfügbar
- Unterstützung für viele Datenbanken erhältlich

--> Mächtig, eingeschränkte Überschaubarkeit für Einsteiger, gute DB Unterstützung, viele Erweiterungen

Ruby on Rails

- Plattformunabhängig, Open Source
- Rein objektorientierte, dynamisch typisierte Sprache, keine primitive Typen: Alles ist ein Objekt
- Grundsatz „convention over configuration“
- Kann wie Perl oder Python CGI Skripte ausführen
- Klassen und Module lassen sich nachträglich erweitern oder verändern
- Rails: Komplettes erprobtes MVC-Framework

--> Mächtig, praxiserprobt, aus einem Guss, kurze Entwicklungszeiten

PHP

- Plattformunabhängig, Open Source
- Direkt in Apache eingebunden
- Sprachstil & Syntax orientieren sich an C, Java, Perl (somit leicht les- und lernbar)
- Spezialisiert auf Scripting im Web -> kompakter Code
- Direktes API für viele Datenbanken
- Unterstützung vieler Drittsysteme (XML, ...)
- Sammlung von Erweiterungen

--> Übersichtlich, leichter Einstieg, gute DB Unterstützung

Active Server Pages

- Für Microsoft Server (IIS) auf Windows Server
- Implementationen für andere Server erhältlich (aber selten)
- ASP-Seiten werden in .NET-Sprachen codiert (VB, C#, ...)
- Gute & einfache Erweiterbarkeit dank Nutzung von .NET- oder COM-Objekten (Component Object Model)
- Flexibler Zugriff auf beliebige Datenquellen/-ablagen mittels ADO, vor allem auch MS-proprietäre wie Exchange, Excel, Access

-> Mächtig, gut für MS Umgebungen, leichter Einstieg, einfacher Datenzugriff

Java Server Pages

- Läuft in JVM auf diversen Plattformen
- Konzept ähnlich wie ASP/PHP (Java-Code in HTML Seiten eingebettet)
- Hohe Leistungsfähigkeit, da JSP Seiten beim ersten Aufruf in reinen Java-Code übersetzt und kompiliert werden und nach dem Start im Speicher vorgehalten werden
- Einfache Erweiterbarkeit mit Tag Libraries (Funktionalität von Java-Libraries in HTML Tags gekapselt: DB-Zugriff, XML, ...)

--> Mächtig, durchgängige Verwendung von Java als Programmiersprache, leichter Einstieg dank Scripting

Webseiten mit Java

Java Web Applikation: Serverseitige Anwendung bestehen aus versch. Java Komponenten (Servlets, JSP, Java Beans)
Servlets: Serverseitige Analogie zu Applets

Servlet Container

Tomcat, GlassFish

Aufgabe

- Kommunikation mit Webserver
- Lifecycle Management (Verwalten des Lebenszyklus des Servlets (Laden d. Klassen, Initialisierung, Aufruf der Methoden, ...)
- Multithreading Support (erstellt für jeden Java-Thread ein Servlet Request)
- Deklarative Konfiguration (XML Deployment Descriptor) konfiguriert Struktur der Anwendung, ohne Code ändern zu müssen (Sicherheit, Filter, Initialisierung)
- JSP-Support (übersetzen von JSP in Java-Servlets)

Servlets

API-Klassen

- java.servlet.Servlet (Java-Interface, welches die 5 Basisfunktionen des Servlet API definiert | Lifecycle-Funktionen: init(ServletConfig), service(ServletRequest, ServletResponse), destroy() | Konfiguration und Statusinformation (getServletConfig(), getServletInfo())
- java.servlet.GenericServlet (Abstrakte Implementierung des Servlet-API | protokollunabhängig)
- java.servlet.http.HttpServlet (Basisimplementierung für HTTP/S)

Servlet Livecycle

init(ServletConfig)

- Wird vor der Behandlung des ersten Requests durch Servlet-Container aufgerufen
- ServletConfig enthält Initialisierungsparameter (Servlet-Context)
- Servlet-Context wird in <web-app> im deployment-descriptor web.xml spezifiziert

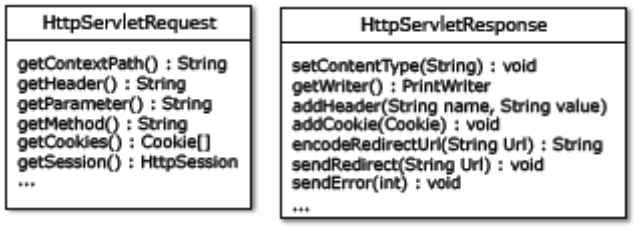
service(ServletRequest, ServletResponse)

- Wird bei jedem Client-Request aufgerufen
- Eine Instanz behandelt alle Requests (Multithreading)
- HttpServlet verteilt die service-calls auf protokollspezifische Methoden:
  - doGet(), doPost(), doPut(), doHead(), ...
  - Jeweils mit den erweiterten Parametern HttpServletRequest und HttpServletResponse

destroy()

- Wird aufgerufen bevor das Servlet aus dem Speicher entfernt wird (z.B. Nach Timeout von 30' definiert im web.xml)
- Bereinigen von Ressourcen

HttpServletRequest/-Response



doGet() und doPost() enthalten jeweils zwei Parameter

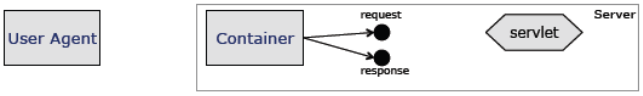
- HttpServletRequest request
  - Dient dem Zugang zu Informationen im Client-Request
    - request.getRemoteAdr(); (=CGI REMOTE_ADR)
    - request.getParameter(„name“); meldet den Parameter mit dem Namen „name“ zurück
- HttpServletResponse response
  - Dient zur Rückgabe der Servlet-Antwort
    - response.setContentType(„text/plain“);
    - PrintWriter out = response.getWriter();
    - PrintWriter ermöglicht Schreiben in Servlet-Text-Output-Stream: out.println(„Hallo“ + i);

Request Handling

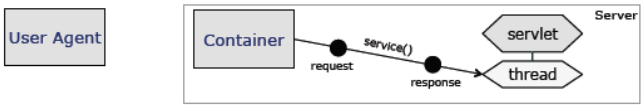
Benutzer ruft Servlet via URL auf

Container erkennt Anfrage für Servlet und kreiert zwei Objekte

- HttpServletRequest -> request
- HttpServletResponse -> response

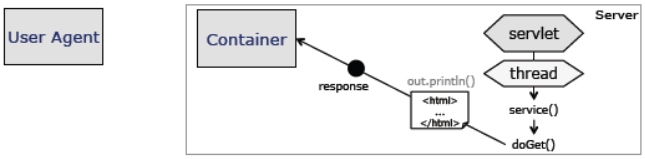


Container identifiziert angefordertes Servlet, kreiert und alloziert einen Thread für den Request und übergibt die request- und response- Objekte an den Thread



Container ruft die Methode service() auf, welche wiederum doGet() aufruft (oder doPost() je nach Methode)

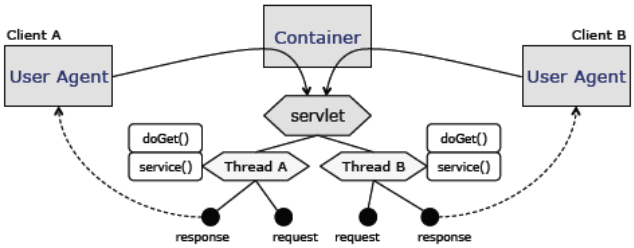
doGet() generiert die dynamischen Seiten im response-Objekt



Container konvertiert das response-Objekt in einen HTTP-Response, beendet den Thread und löscht request- und response-Objekt

Servlet Threading

Container startet Threads, um mehrere Requests eines einzelnen Servlet zu verarbeiten -> Nur eine Instanz jeder Servlet Klasse



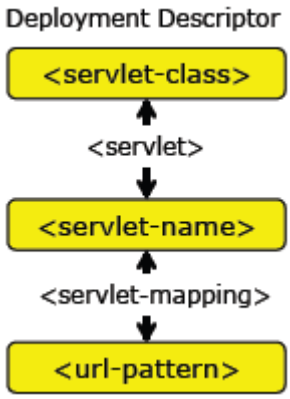
-> Servlets müssen Thread-Safe implementiert werden

- Gemeinsame Instanzvariablen (Vorsicht)
- Synchronisierung des Zugriffs auf Ressourcen (Session, Files, ...)
- HttpServletRequest- und HttpServletResponse-Objekte werden jedoch pro Anfrage neu erstellt

Servlet Adresse

Wie weiss der Container, unter welcher Adresse welches Servlet angesprochen werden soll?

- Klassen-Bezeichnung
  - Vollständiger eindeutiger Java-Klassenname inkl. Packages
- Deployment-Name
  - Künstlicher interner Name, ausschliesslich als Referenz in der Konfiguration gedacht
- URL
  - Webadresse, unter welcher das Servlet für die Clients erreichbar ist



Im Deployment Descriptor (web.xml) werden die in diesem Anwendungs-Context verfügbaren Servlets definiert (Name, Klasse, URL, Initialisierungsattribute, ...):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2.2.dtd">
<web-app>
  <servlet>
    <servlet-name>FirstServlet</servlet-name>
    <servlet-class>wbd.FirstServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>FirstServlet</servlet-name>
    <url-pattern>/first</url-pattern>
  </servlet-mapping>
</web-app>
```

Interner Deployment Name des Servlets

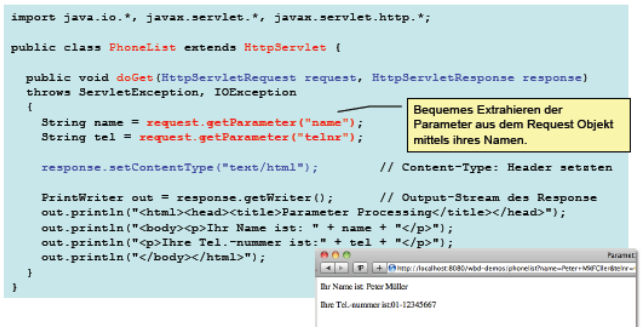
Klasse eines Servlets (inkl. Package-Pfad)

Pfad/Name unter welchem das Servlet angesprochen wird (relativ zum Context) Weitere Beispiele: /address/save, /bestellung/*, *.do, ...

Beispiel für eine Formularauswertung

```
<form method="get" action="phonelist">
  Ihr Name? <input type="text"
name="name" /><br />
  Tel. Nr.? <input type="text"
name="telnr" /><br />
  <input type="submit" value="Senden">
</form>
```

Auswertung mit Servlet



JSP

Konzept ähnlich wie PHP oder ASP

- Dokumentenzentriert: Java-Code in HTML Seiten
- Besser geeignet für Designer (Views)

Benötigt JSP / Servlet-fähigen Container

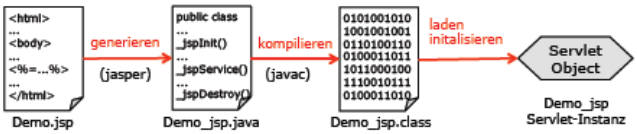
- Aus den Code-Bruchstücken werden automatisch Servlets generiert, mit Java Compiler übersetzt und dann als normales Servlet ausgeführt

Vorteile

- Gute Leistungsfähigkeit und Effizienz, da JSPs wie Servlets nur beim ersten Aufruf übersetzt werden und dann im Speicher bleiben
- Gute Erweiterbarkeit dank Nutzung von JavaBeans und eigene JSP-Tag-Libraries

Beim ersten Aufruf

- generiert automatisch aus der JSP-Datei ein Java-Servlet (z.B. Jasper)
- compiliert das Servlet (javac)
- lädt die Klasse und führt sie aus



JSP Markup

Skriptlet

```
<% Java-Code %>
```

- Java-Code wird an der entspr. Stelle in der Seite eingefügt
- XML-Variante: <jsp:scriptlet>JavaCode</jsp:scriptlet>

Ausdruck

```
<%= Java-Ausdruck %>
```

- Java-Ausdruck wird ausgewertet und das Resultat an dieser Stelle in die Seite eingefügt
- Identisch zu: <% out.print(Java-Ausdruck); %>
- XML-Variante: <jsp:expression>Java-Ausdruck</jsp:expression

Deklaration

```
<%! Java-Code %>
```

- Java-Code wird auf Klassenebene (ausserhalb der jspService-Methode) eingefügt. z.B. Instanzvariable, Hilfsmethode, ..
- XML-Variante: <jsp:declaration>Java-Code</jsp:declaration>

Kommentar

```
<%-- Kommentar --%>
```

- Kommentar wird von JSP-Compiler ignoriert. d.h. Ist in HTML-Seite nicht vorhanden
- XML-Variante: <!-- Kommentar -->

JSP Direktiven

```
<%@ xxx ... %>
```

- XML-Variante: <jsp:directive.xxx .../>
  - werden beim Konvertieren ausgewertet (compile time)
  - <%@ include file=".../Header.jsp" %>
    - fügt Header einbindung

page

```
<%@ page xxx="..." %>
```

kontrolliert die Struktur und Funktionen des generierten Servlets

Wichtigste Attribute

Code	Beschreibung
import="java.util.*, java.text.*"	Fügt import-Statements ein
Content-type="text/html;charset="UTF-8"	Definiert den Content-Type-Header
pageEncoding="UTF-8"	Zeichen-Codierung der JSP-Datei
session="true"	Session erzeugen (default)
errorPage="MyError.jsp"	Wird bei Exception aufgerufen"
isErrorPage="true"	Ist eine ErrorPage Zugriff auf Exception-Info möglich

JSP Actions

Aktionen, die vom JSP-Converter ins Servlet eingebunden und zur Laufzeit ausgewertet werden

```
<jsp:forward page="../Login.jsp" />
<jsp:include page="../DynamicFooter.jsp"
/>
```

Vordefinierte Variablen

Variable	Beschreibung
request	HttpServletRequest-Objekt
response	HttpServletResponse-Objekt
out	PrintWriter-Objekt response.getWriter()
session	Session-Objekt request.getSession()
application	Application-Context getServletContext() Kontext zum Ablegen von gemeinsamen Daten (set/getAttribute())
pageContext	Page-Context, der solange lebt, wie die Generierung der Seite dauert. Gemeinsamer Speicher, wenn die Seite aus mehreren JSP-Dateien aufgebaut wird set/getAttribute()
config	Konfigurationsinfo des Servlets getServletConfig()

Model View Controller

Model

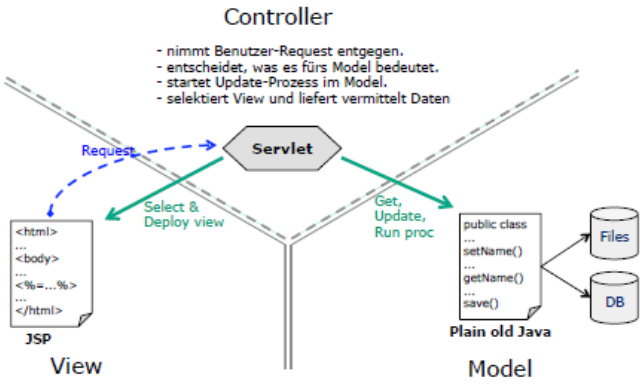
- Kapselt den aktuellen Status der Anwendung
- Beschreibt das Datenmodell der Anwendung (Struktur, Persistenz, ...)
- Enthält auch die Geschäftsprozesse (Business Logic)
- Ist unabhängig von View und Controller

View

- Zuständig für die Darstellung der Daten
- Entgegennahme der Interaktionen mit Benutzer

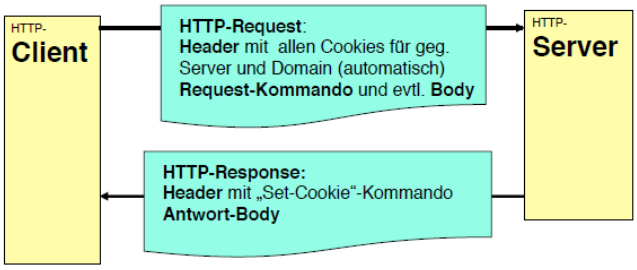
Controller

- Vermittler zwischen Model und View
- Nimmt Benutzeraktionen entgegen und veranlasst Änderungen im Model
- selektiert darzustellende View und bereitet sie vor
- fungiert bei gewissen Ausprägungen auch als Beobachter des Models, um Änderungen an die View weiterzuleiten



Cookies

Ermöglicht, Informationen vom Server auf Client permanent zu speichern.



Setzen von Cookies

```
Cookie myCookie = new Cookie("summe", "245");
myCookie.setDomain("zhwin.ch");
// Lebensdauer in Sekunden setzen (z.B. 3600s = 1h)
//   wenn negativ : Cookie beim Beenden des Browsers löschen
//   wenn 0 (Null): Cookie sofort löschen
myCookie.setMaxAge(3600);
response.addCookie(myCookie);
```

Auslesen von Cookies

```
String summe = null;
Cookie[] cookies = request.getCookies();
if (cookies != null) {
  for(Cookie cookie : cookies) {
    if (cookie.getName().equals("summe")) {
      summe = cookie.getValue();
    }
  }
}
```

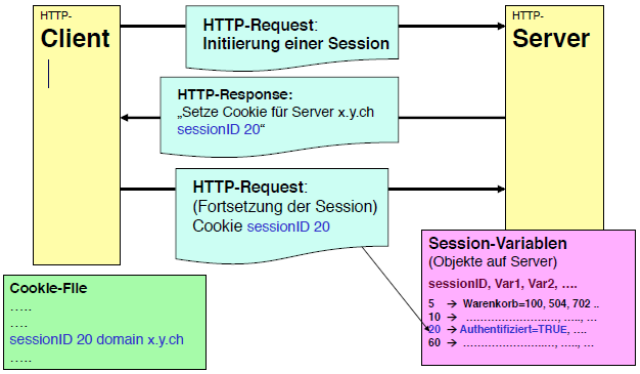
Session

Serverseitiger Datenspeicher, in dem Daten requestübergreifend gespeichert werden können

- beliebige Datenmenge (Java-Objekte)
- Sicher (nicht extern manipuierbar)

Verbindung zw. Request und Session via SessionId in

- Cookie (sessionId = 0x9087A987EB889834)
- Rewriting der sessionId in jeder Antwortseite



Session erstellen: HttpSession session = request.getSession()
-> gibt assoziierte Session zurück
-> erstellt neue Session, wenn nötig

Werte speichern / lesen: session.setAttribute(„name", wert);
Objekt wert = session.getAttribute(„name");

Session beenden: session.invalidate();

JSP und Java Beans

- JavaBeans API: Standardformat für wiederverwendbare Klassen
- Andere Programme können automatisch Informationen über diese Klasse entdecken und können dann die Bean-Klasse kreieren und manipulieren, ohne dass dafür User-Code geschrieben werden muss
- Für den JSP-Einsatz sind folgende Bean-Eigenschaften relevant
  - Bean-Klasse muss einen leeren Konstruktor haben
  - Bean-Klasse sollte keine public instance Variablen haben
  - Für die Manipulation aller wichtigen Instanzvariablen sollten die Methoden getXxx and setXxx bereitgestellt werden

Anwendung von Beans in JSP

- Verwendung von resusable Classes
- Spezielle, komplexe Funktionalität
- Speicherung persistenter Daten, die z.B. Von mehreren Seiten geteilt werden müssen

Syntax

```
<jsp:useBean id="name"
class="package.Class" />
```

Zugriff auf Bean-Eigenschaft. Beispiel Property „title“

Lesen

```
<jsp:getProperty name="book1"
property="title" />
```

oder

```
<%= book1.getTitle() %>
```

Schreiben

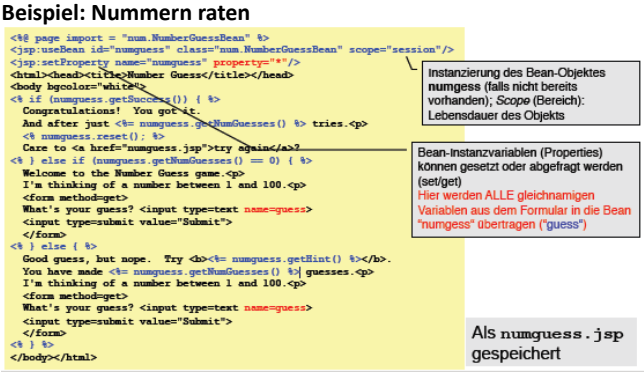
```
<jsp:setProperty name="book1"
property="title" value="title" />
```

oder

```
<%= book1.setTitle("title") %>
```

Zugriff auf alle Parameter einer Anfrage: property=“*“

Beispiel: Nummern raten



XML – eXtensible Markup Language

HTML bietet

- gute Bildschirm-Darstellung
- einfach strukturierte Dokumente
- einfaches, normiertes, öffentliches Format
- Darstellung von CSS vom Inhalt separierbar

HTML bietet nicht

- Möglichkeit, einen beliebigen Element-Inhalt zu beschreiben (z.B. Datum, Preis, Anzahl, ISBN, ...), weil nur fixe Anzahl Tags definiert ist

Idee: die Daten beschreiben sich selbst. Wie?

- z.B. Durch die Reihenfolge im Dokument
- Durch zusätzliche, im Dokument enthaltene „markup“-Informationen --> XML

```
<HighscoreList>
  <Player>
    <Rank>104</Rank>
    <Name>sIF</Name>
    <Level>86</Level>
    <Exp>95911271</Exp>
    <RankMutation>-12</RankMutation>
    <LevelMutation>0</LevelMutation>
    <ExpMutation>-1767876</ExpMutation>
  </Player>
</HighscoreList>

<book>
  <title>XML in Action</title>
  <anzahl>5</anzahl>
  <author>
    <vorname>Michael</vorname>
    <nachname>Young</nachname>
  </author>
  <preis>45</preis>
  <verlag>Microsoft</verlag>
</book>
```

markup language: Tags und Attribute
extensible: Tags nicht vordefiniert -> neue Tags möglich

Problem: Browser kann nicht

- alle denkbaren Tags kennen
- wissen, wie unbekannte Tags dargestellt oder interpretiert werden

Lösung

- Informationen über Aufbau und Grammatik-Regeln des Dokuments an Programme liefern, die XML Dokumente verarbeiten
  - DTD (Document Type Definition) oder XML-Schema
- Darstellungs-Regeln definieren -> CSS oder XSL (Extensible Stylesheet Language)

Unterschiede HTML – XML

```
<?xml version="1.0" encoding="ISO-8859-
1" standalone="no" ?>
```

- <!DOCTYPE ... ... „...“ „...“> mit DTD Spezifikation folgt direkt anschliessend XML-Dokument hält die DTD ein -> valid
- Tags richtig geschachtelt -> Baumstruktur

- Zugriff auf Elemente mit DOM (Document Object Model) möglich
- wellformed <tag1>..<tag2>..</tag2>...</tag1> XML-Parser brechen bei nicht wohlgeformten Dokumenten mit einer Fehlermeldung ab
- Alle Tags schliessen
- XML unterscheidet Gross- /Kleinschreibung. Tags werden kleingeschrieben
- Attribute immer mit Wert <tag attribut="wert">...</tag>
- Attributwerte in „..." eingeschlossen

Wichtige XML-Eigenschaften

- Wellformed
  - Start- und End-Tag-Paare
  - spezielle Form für „empty elements": <... />
    - Diese Elemente
      - enthalten Info in den Attributen und nicht zwischen dem Beginn- und Endtag
      - enthalten keine Info, aber haben Einfluss auf Formatierung
  - Keine überlappenden Elemente (richtig verschachteln)
  - Attribute korrekt in „..." eingeschlossen
- Valid
  - Wellformed... plus
  - Korrekte Struktur und Grammatik des Dokumentes (definiert in DTD)

Definition XML-Struktur

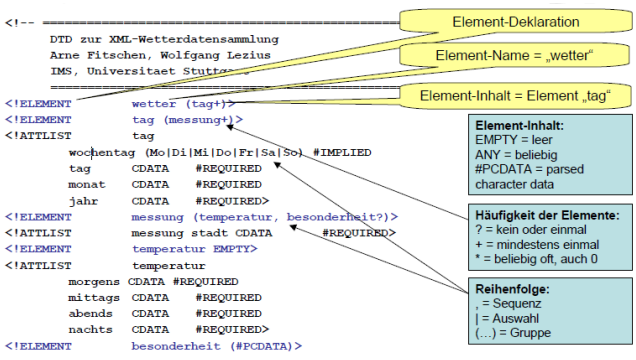
DTD (Document Type Definition)

- Die DTD wird mit Hilfe der Deklarations-Regeln von XML erstellten

XSD (XML Schema Definition)

- wesentlich flexibler als DTD
- Geschrieben in XML
- Genauere Datentypen
- Vererbung möglich
- Unterstützt Namespace

DTD Beispiel




```
<!-- ============================================
     DTD zur XML-Wetterdatensammlung
     Arne Fitschen, Wolfgang Lezius
     IMS, Universitaet Stuttg
     ============================================ -->
<!ELEMENT       wetter (tag+)>
<!ELEMENT       tag (messung+)>
<!ATTLIST       tag
        wochentag (Mo|Di|Mi|Do|Fr|Sa|So) #IMPLIED
        tag     CDATA   #REQUIRED
        monat   CDATA   #REQUIRED
        jahr    CDATA   #REQUIRED>
<!ELEMENT       messung (temperatur, besonderheit?)>
<!ATTLIST       messung stadt CDATA     #REQUIRED>
<!ELEMENT       temperatur EMPTY>
<!ATTLIST       temperatur
        morgens CDATA #REQUIRED
        mittags CDATA   #REQUIRED
        abends  CDATA   #REQUIRED
        nachts  CDATA   #REQUIRED>
<!ELEMENT       besonderheit (#PCDATA)>
```

XSD Beispiel

XML-Schema anstelle der DTD (Auszug)

```
<!-- Typ zum Messung-Element: temperatur- und evtl. besonderheit-Element -->
<xsd:complexType name="messungType">
  <xsd:sequence>
    <xsd:element name="temperatur" type="temperaturElementType"/>
    <xsd:element name="besonderheit" type="xsd:string" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
  <xsd:attribute name="stadt" type="xsd:string"/>
</xsd:complexType>
<!-- Typ zum Temperatur-Element: alle Attribute gehorchen einem eigenen Typ -->
<xsd:complexType name="temperaturElementType">
  <xsd:attribute name="morgens" type="temperaturType"/>
  <xsd:attribute name="mittags" type="temperaturType"/>
  <xsd:attribute name="abends"  type="temperaturType"/>
  <xsd:attribute name="nachts"  type="temperaturType"/>
</xsd:complexType>
<!-- Typ spezifiziert eine Temperaturangabe: {-50,..,50} -
<xsd:simpleType name="temperaturType">
   <xsd:restriction base="xsd:integer">
     <xsd:minInclusive value="-50"/>
     <xsd:maxInclusive value="50"/>
   </xsd:restriction>
</xsd:simpleType>
```

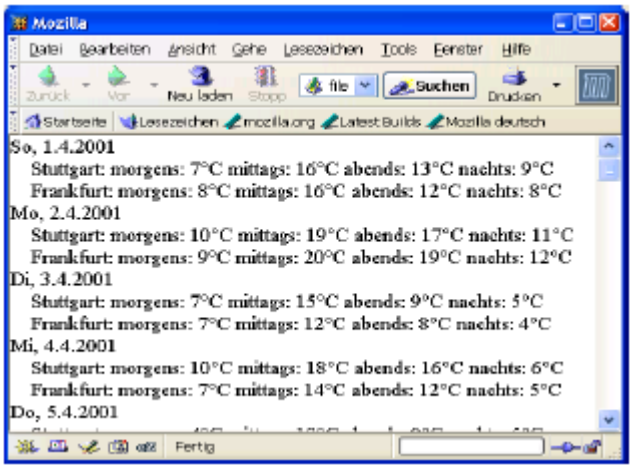
XML- statt SGML-Syntax
→ Kein spezielles Tool nötig
→ Genauere Datentypen
→ Vererbung möglich
→ unterstützt Namespaces
→ selbstbeschreibend
→ Schema verdrängt DTD

Formatierung (CSS, XSL)

```
tag { display: block; }
tag:before { content:
  attr(wochentag)", "
  attr(tag)"."
  attr(monat)"."
  attr(jahr); }

 messung { display: block;
   text-indent: 1em; }
 messung:before { content:
   attr(stadt)": "; }

 temperatur:before { content:
   "morgens: "attr(morgens)"°C "
   "mittags: "attr(mittags)"°C "
   "abends: "attr(abends)"°C "
    "nachts: "attr(nachts)"°C"; }
```



CSS -> XSL

<u>CSS ist einfach und beschränkt</u>

- Spezifiziert lediglich die Formatierung jedes XML-Elements
- Element-Reihenfolge aus der XML-Datei bleibt erhalten
- Herausfiltern von Daten ist schlecht möglich
- Automatische Inhaltsverzeichnis kaum möglich

XSL ist komplex und mächtig

- Wandelt mit Hilfe von XSL ein XML-Dok. In ein anderes XML-Dok. Um -> XSL-Transformation
- Wandelt XML mit Hilfe von XSL in HTML um
- Umwandlung in Nicht-HTML-Formate (z.B. PDF) möglich
- Programmiersprache mit Regeln, aber ohne Variablen (Filtern, Reihenfolge, Sortierung, Umwandlung, Addieren usw.)

XSL-Transformation

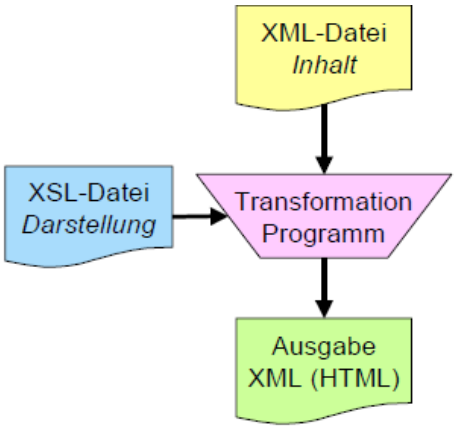
Transformation ist möglich:

Clientseitig

- Browser erzeugt aus XML-Datei mit Hilfe der XSL-Datei den HTML-Code für die Ausgabe
- Vorteil: funktioniert auch lokal
- Nachteil: braucht neuen Browser

Serverseitig

- CGI-Programm oder Web-Server wandelt XML-Daten in HTML um und schickt dem Browser die fertige Website
- Vorteile: Browser sieht reines HTML-Datei, Vollständige Trennung Inhalt/Design
- Nachteil: aufwendiger



XLL: XLink und XPointer

XLL: eXtensible Linking Language

- XML-Links beschreiben Beziehungen zw. Dokument-Teilen
- Darstellung eines Links, bleibt der Applikation überlassen

Beziehungen zw. Teilen eines Dokuments

- Eindeutige Bezeichnung eines Teils mit id-Attribut
- Def. einer Beziehung mit IDRED und IDREFS

XPointer für Beziehungen zw. Teilen in versch. Dokumenten

- kann mit Ids arbeiten
- oder mit dem DOM und XPath

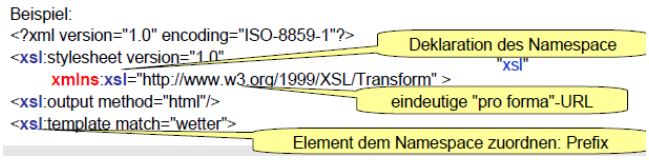
XLink für Hyptertext-Verweise

- simple entspricht dem HTML-Link <a> mit mehr Optionen
- extendes für Links mit mehreren Zielen und/oder zw. verschiedenen Dokumenten

Namespaces

Mehrere DTDs für ein Dokument -> Namenskonflikt

- z.B. Kommt das Element <title> in fast allen DTDs vor, aber mit unterschiedlichen Definitionen
- Namespases schaffen eindeutige Verhältnisse



XML-Parser

XML-Anwendung

- liest die XML-Daten ein
- verarbeitet sie
- und generiert eine Ausgabe

DOM-Parser

-> XML-Daten als Baumstruktur im Speicher abgebildet

- Hoher Speicherbedarf, Ausgabe erst am Schluss
- Elemente einfügen, löschen, verschieben, modifizieren möglichkeit

SAX-Parser (Simple API for XML Access)

-> Daten „zeilenweise“ lesen, verarbeiten und ausgeben

- Methoden-Aufruf, wenn ein Tag eingelesen wird (event-based)
- Kleiner Speicherbedarf, Ausgabe kontinuierlich
- Element-Reihenfolge fix, weil Parser nur kleinen Ausschnitt sieht

Java, PHP und JS unterstützen XML-Parsing

Gestaltung

Nähe

- Inhaltliche Nähe durch räumliche Nähe repräsentieren
- Zusammengehörende Inhalte: Visuelle Einheiten
- Struktur des Inhalts visualisieren

Leerraum

- Nicht Platzvergeudung, sondern Gestaltungsmittel
- Einsetzen, um Informationen sinnvoll zu ordnen und verständlich zu machen

Geschlossenheit

- Dinge mit geschlossenem Umriss werden von Wahrnehmung gruppiert

--> Durch Nähe zeigen, was inhaltlich zusammengehört und den Inhalt damit strukturieren

--> Keine Angst vor leeren Flächen

Ausrichtung

- Jedes Element soll eine visuelle Verbindung mit etwas Anderem auf der Seite haben
- Jedes Element auf einer Seite sollte an anderen Elementen ausgerichtet sein
- Keine zentrierten Überschriften über linksbündigen Flattersatz
- Zentrieren generell sparsam einsetzen

Ähnlichkeit
Ähnliche Dinge werden von Wahrnehmung gruppiert.

- Farbe, Textur, Grösse, Helligkeit, ...

Wiederholung
Durch Wiederholung ähnlicher Elemente inhaltliche Beziehungen ausdrücken

- Überschriften mit gleicher Schriftart, Schriftgrösse und einheitlichem Abstand von anderen Elementen
- Einheitliche Aufzählungszeichen

--> Konsistenz: Ergebnis von Ähnlichkeiten
--> Ähnlichkeit bietet Möglichkeit, Elemente über Distanzen (auch Seiten) zu verbinden
--> Ergänzt Prinzipien Nähe und Ausrichtung

Kontrast
Unterschiede zw. Elementen betonen

Was **unterschiedlich** ist, sollte nicht nur leicht unterschiedlich sondern richtig unterschiedlich sein

Auf verschiedene Arten erreichbar

- Schriftgrösse
- Schriftarten (aber keine ähnlichen)
- dünne und dicke Linien
- horizontale / vertikale Richtung
- kleine / grosse Grafik
- Farben (Vorder- und Hintergrund)

<u>Ziele</u>

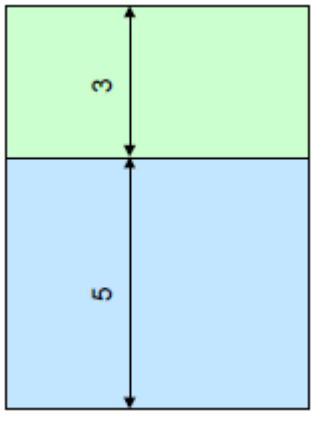
- Interesse, Aufmerksamkeit erzeugen
- Inhalt organisieren, strukturieren

Elemente sollten gleich oder unterschiedlich gestaltet sein, jedoch nicht ein wenig unterschiedlich.

Goldener Schnitt

- Aufteilung 1:1.618 (bzw. ~3:5) besonders harmonisch
- Kommt häufig in Natur vordefiniert

Kleinere Strecke zur grösseren gleiches Verhältnis wie grössere zur Gesamtstrecke



Farben

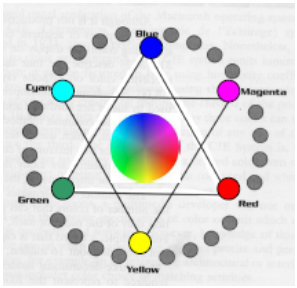
- Können Kontrast erhöhen
- Können Layouts lebendiger machen
- Wirkung auf Leser beachten

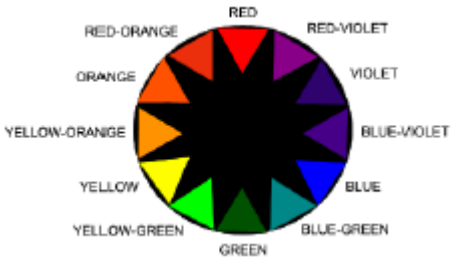
<u>Wichtig</u>

- Zum Design & Inhalt passende Farben
- Passende Hintergrundfarbe
- Passende Farbkombinationen

<u>Vermeiden</u>

- Keine auffälligen Hintergrundbilder
- Weisser Text auf schwarzem Hintergrund





<u>Kombinationen von Farben</u>
Nah-komplementäre Farben

- ergeben gute, angenehm wirkende Farbkombinationen
- Im Farbkreis gegenüberliegende Farbe finden, dann zwei Farben in gleicher Distanz links und rechts finden

Nachbarn und verwandte Farben

- erzeugen auch Farbharmonie
- liegen nebeneinander auf dem Farbkreis

Monochromatische Farbkombinationen

- Variationen von nur einer Farbe, die durch die Beigabe von schwarz und weiss verändert wird

Texte und Typographie
Ziel: Gute Lesbarkeit der Texte

Faktoren der Lesbarkeit
Schriftfamilie, Laufweite (Zeichenabstand), Wortabstand, Schriftgrad, Schriftbild (kursiv, fett), Zeilenabstand, Satzbreite, Satzart (Blocksatz, Flattersatz), Farbe

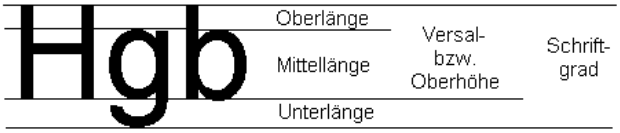
Druck ausserdem:
- Druckqualität
- Papieroberfläche
- Papierfärbung

Bildschirm ausserdem:
- Monitoreinstellung, Auflösung, Schäfe
- Monitoroberfläche (Spiegelung)
- Fonttyp
- Zeichendarstellung

Versalien und Gemeine
Versalien: Grossbuchstaben des Alphabets
Gemeine: Kleinbuchstaben des Alphabets

Kontur von Wörtern
Wichtig für schnelles Erkennen von Wörtern

Schriftgrad (-Grösse)



Beim Druck
- 1 Punkt (pt) = 0.353mm (= 1/72 inch)
- Schaugrössen: 18pt, 24pt, 36pt, 48pt, 60pt
- Auszeichnungen: 14pt, 16pt
- Lesegrössen: 8pt, 9pt, 10pt, 12pt
- Konsultationsgrössen: 6pt, 7pt

Serifen / Sanserif
Serifen
- verbessern Lesbarkeit
- nur für Druck geeignet, weil Bildschirmauflösung meist zu gering für Details der Serifen

Beschreibung	**CSS**
Schriftschnitt	font-style - italic, oblique, normal font-variant - small-caps, normal font-weight - bold, bolder, lighter, 100, 200, ... , 900, normal
Laufweite	letter-spacing: 1px;
Satzbreite	Width, min-width, max-width
Zeilenabstand	Line-height: 15px;

Kombination von Schriften
Drei Effekte
- Konkordanz: Weitgehende Übereinstimmung, harmonisches Gesamtbild
- Konflikt: Ähnlichkeit, weder Übereinstimmung noch richtiger Kontrast
- Kontrast: Klare, erkennbare Unterschiede

Konkordanz lässt sich mit versch. Schnitten derselben Schriftfamilie erreichen
Wirkt ruhig, formal, etwas langweilig

Konflikt entsteht bei Kombination ähnlicher Schriftfamilien -> vermeiden

Kontrast: Was nicht gleich ist, sollte richtig verschieden werden
-> lenkt Aufmerksamkeit auf Kontrast

Satzarten
Blocksatz
- Gut lesbar, wirkt ruhig
- Benötigt ausreichende Satzbreite, sonst können Lücken entstehen
- Gut in Spalten verwendbar

Linksbündiger Flattersatz
- Wirkt lebendig
- Muss eingesetzt werden, wenn Satzbreite nicht ausreichend für Blocksatz
- Abschnittswechsel schlechter erkennbar (Einzug oder Blindzeile nötig)

Rechtsbündiger Flattersatz
- Wie linksbündiger Flattersatz
- Geeignet zum Beispiel links von einer geraden Kante

Zentrierter Satz
- Wirkung: repräsentativ, vornehm, würdevoll aber auch: altmodisch, langweilig
- Nur einsetzen, wenn solche Wirkung beabsichtigt ist

```
text-align: left | center | right | justify
```

Absätze – Einzug

```
text-indent: 3em;
```

Usability
Häufige Probleme
- Fehlende Botschaft auf Startseite
  - Worum gehts?
- Fehlende Orientierungsmöglichkeiten
  - Wo bin ich? Wie komme ich nach ...?

- Unternehmens- statt Kundenorientierung
- Optik vor Inhalt und Funktion
  - Lesbarkeit, Benutzbarkeit, Verständlichkeit werden zugunsten der Gestaltung geopfert
- Lange Ladezeiten
- Offline-Inhalte im Web

Grundlegende Fragen
- Ziel der Website?
- Zielpublikum?
- Organisatorische Rahmenbedingungen?
  - Geschäftsprozesse, Mitarbeiter
- Technische Rahmenbedingungen
  - Technologien, Browser

Inhalte
- Relevanz
- Bildschirmgerechte Menge
- Bildschirm- und Druckversion

Site-Struktur
Lineare Struktur
- Navigation einfach: vor/zurück

Hierarchische Struktur
- Allgemein üblich, verständlich, erlernbar
- Standard für Strukturiereung einer Site

Navigation / Bedienung
Ziel: Intuitive, erwartungskonforme Bedienung
- Passend gewählte Bezeichnungen
- Verweise eindeutig erkennbar
- Verweis-Ergänzungen (PDF, ..)

<u>Navigationsleiste</u>
- Nicht zu viele Verweise
- Wichtige Einträge zuerst
- Mehr Elemente: Logisch gruppieren

<u>Sitemap</u>
- Übersicht über Struktur der Seite
- Bei grossen Sites sinnvoll
- Normal: Textbasiert, soll schnell laden

Seitenlayout
- Ziele
  - Konsistenz innerhalb der Site, übersichtlich, leicht erfassbar, aufgeräumt

- Stark abhängig vom Zielpublikum
  - Einige Konventionen: Position des Logos, Logo mit Verweis auf Startseite, Navigationsbereiche

Layout Varianten
Fixed width layout
- Feste Breitenangabe, meist in px

Liquid, fluid layout
- Pass sich an Grösse des Browserfensters an
- Gilt auch für Inhalte (Breite der Texte)
- Ohne Breitenangabe
- Alternativ mit Breitenangabe in Prozentual

Elastic layout
- Passt sich an gewählte Schriftgrösse an
- Breitenangabe in em
- Layout reagiert auf Funktion Text skalieren des Browsers

Fenstergrösse
- Kontrolle über Fenstergrösse Benutzer überlassen
- Tabu: ändern der Fenstergrösse mittels Script
- Öffnen neuer Fenster bei Klick auf Link nur in Ausnahmefällen sinnvoll

Frames
Nachteile
- Probleme bei Suchmaschinen
  - Inhalte werden nicht gefunden
  - Suchergebnis verweist auf Inhalts- oder Navigationsseite
- Probleme mit Bookmarks
  - Gerade angezeigte Seitenkombination oder Ausgangs-Frameset als Bookmark?
  - Unterschiedliches Verhalten der Browser
- Probleme beim Drucken
  - Benutzer wissen nicht, wie nur Inhaltsbereich drucken
- Platzverbrauch
  - Permanent angezeigte Titelleiste verbraucht unnötig Platz
- Verweise von aussen
  - Wie wird auf bestimmten Inhalt verlinkt?
- Wenig aussagekräftige URL

Formulare
- Übersichtlicher Aufbau
- Eingabefelder ausgerichtet
- Sinnvolle Wahl der Eingabeelemente (Textfeld, Radio-Buttons, Checkbox)
- Fehlerhafte Felder direkt markieren und Hilfshinweise geben
- Bereits ausgefüllte Felder müssen für Korrektur mit dem alten Inhalt gefüllt sein

Seitenelemente
Text
- Nicht durch Bilder realisieren
  - Fixe Grösse
  - Höhere Ladezeiten
  - Bei Verweisen kein Hinweis, ob bereits besucht
  - Kein Copy&Paste
- Ausnahmen
  - Navigationselemente oder ggf. Überschriften (aber dann mit ALT-Attribut)
  - E-Mail-Adress vor Spammern verbergen

Bilder
Leichter zu merken und oft schneller zu erfassen als Textform
Bild: räumliche / hierarchische Strukturen
Text: logische Zusammenhänge, abstrakte Konzepte
- Fürs Web optimieren (Anz. Farben, Format)
- Grössere Bilder mit Vorschaubildern
- Attribute für Grössenangabe benutzen
- Nicht durch Browser skalieren
- Attribut ALT ist obligatorisch

- Attribut title für Tooltip-Text

Accessability

Weshalb Barrierefreiheit?

- Weil Gesetz es verlangt
- Webseiten werden besser gefunden
- Wirtschaftlichkeit (bis zu 30% mehr Kunden können erreicht werden)
- Soziales engagement
- Damit Menschen, unabhängig von Einschränkungen, Internetangebot nutzen können

Screen Reader
Applikation, die Text vorliest

Screen Magnifier
Vergrösserungssoftware

Web Content Accessibility Guidelines 2.0
4 Grundsätze

- Wahrnehmbarkeit
- Bedienbarkeit
- Verständlichkeit
- Robustheit

12 Richtlinien
3 Konformitätsebenen: Levels A, AA, AAA

- A: Alle Checkpunkte der Priorität 1 sind erfüllt
- AA: Alle Checkpunkte der Priorität 1 und 2 sind erfüllt
- AAA: Alle Checkpunkte der Priorität 1, 2 und 3 sind erfüllt

ATAG (Authoring Tool Accessibility Guidlines)

- Software und Services zur Erzeugung von Webseiten und Webinhalten

UAAG (User Agent Accessibility Guidelines)
User Agents: Browser, Media Player, Assistierende Technologien
UAAG erklären, wie Benutzeragenten für Menschen mit Behinderungen zugänglich gemacht werden

Zugängliche Inhalte: Grafiken

- Aussagekräftiges alt-Attribut muss definiert werden
- Infos über Funktionalität des grafischen Links weder
- aus URL noch aus Dateiname der Grafis herauszulesen
- Aussagekräftiges alt-Attribut über Funktion des Links

<u>Formatierungsgrafik</u>
Leeres alt-Attribut (für ScreenReader unsichtbar machen)


```
<li>
<img src="arrow.jpg"
alt=""/>Zahlen
</li>
```

Zugängliche Inhalte: Multimedia

- Untertitel zu Videos für Hörbehinderte
- Videos mit Audiobeschreibung für Sehbehinderte
- MediaPlayer per Tastatur bedienbar

Zugängliche Inhalte: Farben
Problem 1: Kontrast zw. Vorder- und Hintergrundfarbe zu klein
Problem 2: Farben werden als einziges Instrument verwendet, um bestimmte Informationen zu vermitteln (z.B. Fehlermeldung nur durch Farbe hervorgehoben

Zugängliche Inhalte: Tabellen
Layouttabelle

- Wird benutzt, um Elemente auf einer Website zu positionieren
- Mögliches Problem: Lesereihenfolge

Datentabelle

- Wird benutzt, um zusammenhängende Informationen darzustellen
- Besitzt explizite oder implizite Markierung von Spalten- und/oder Zeilenüberschriften
- Mögliche Probleme:
  - Vermittlung von Spalten- und Zeilenüberschriften
  - Rasche Vermittlung von Tabelleninhalten

Zugängliche Navigation: Lesereihenfolge
Der Aufbau des HTML-Codes bestimmt die Reihenfolge, in der ein ScreenReader eine Website vorliest
--> Struktur ist erkennbar, wenn CSS abgeschaltet wird

Zugängliche Navigation: Überschriften
ScreenReader können zu Überschriften springen

- Level der Überschriften haben keinen Einfluss auf Lesereihenfolge
- Level kann Hinweis auf Dokumentenstruktur vermitteln

Überschriften können mit CSS versteckt werden

- ScreenReader können trotzdem zu Überschriften springen

Zugängliche Navigation: Sprunglinks
Über Sprunglinks kann zu definierten Ankerpunkten im Dokument gesprungen werden
<a name="anker" />
Skip to Content

Sprunglinks sind oft unsichtbar

- Unter Bildern getarnt
- Ausserhalb des Bildschirms
- Sichtbar, sobald Fokus darauf liegt

Zugängliche Navigation: WAI-ARIA Landmark Roles
Landmark Roles

- Gruppe der WAI-ARIA-Rollen, die – als Navigationsbalken – Bereiche einer Webseite darstellen
- ...
- ...
- ...

Zugängliche Dateneingabe: Barrierefreies Formular

- Input-Elemente mit entsprechenden Labels versehen
  - <label for="vname">Vorname *</label>
  - <input type="text" id="vname" name="vorname">
- Informationen über korrekte/inkorrekte Eingaben als Kombination aus Form/Icon, Grösse und Farbe darstellen
  - Obligatorische Felder mit * Kennzeichnen
  - In Rückmeldung angeben, welche Felder noch ausgefüllt werden müssen

Zugängliche Dateneingabe: Gruppieren von Controls
Eingabefelder können mit <fieldset> zusammengefasst werden

- Fieldsets können geschachtelt werden

Inhalt von <legend> wird vor jedem Control vorgelesen

Zugängliche Dateneingabe: Schaltflächen

Normale Schaltfläche: type="button", „submit", „reset"

- Prompting ist im Control vorhanden
  - value="Speichern"
  - Falls value nicht angegeben, wird Defaultwert verwendet (submit, reset)
  - ScreenReader liest value-Attribut vor
- Bei type="button" gibt es keine Standardwerte
  - <input type="submit">
  - <input type="reset">
  - <input type="submit" value="Skip">
  - <input type="b1" type="button">

Grafische Schaltfläche

<input type="image" name="search" src="search.png" value="Start Search" alt="Start Search">

- ScreenReader bevorzugt das alt-Attribut, d.h. Ist eines vorhanden, wird es vorgelesen
- Fehlt das alt-Attribut, wird das value-Attribut vorgelesen

Beispiel XSL

```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
>
<xsl:output method="html"/>
<xsl:template match="/">
<html>
<head>
   <title>Books Ordered by Author and
   Title</title>
</head>
<body>
   <h2>Books ordered by Author and
   Title</h2>
   <table border="1">
        <tr>
          <th>Author</th>
          <th>Title</th>
          <th>Id</th>
        </tr>
        <xsl:for-each select="//book">
        <xsl:sort select="author"/>
        <xsl:sort select="title"/>
        <tr>
          <td><xsl:value-of select="author"/>
          </td>
          <td><xsl:value-of select="title"/>
          </td>
          <td><xsl:value-of select="@id"/>
          </td>
        </tr>
        </xsl:for-each>
   </table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

```
package wbd.control;

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

import wbd.model.Store;

public class ProcessOrder extends HttpServlet {
private static final long serialVersionUID = 1L;

protected void doPost(HttpServletRequest request, HttpServletResponse
response)
        throws ServletException, IOException
{
        response.setContentType("text/html");
  PrintWriter out = response.getWriter();

  String lastname = request.getParameter("lastname");
        String firstname = request.getParameter("firstname");
        String product = request.getParameter("product");

        Map<String,String> errors = new
        HashMap<String,String>();

        if (lastname == null ||
                lastname.trim().isEmpty()) {
        errors.put("lastname", "Bitte geben Sie einen
        Nachnahmen ein");
        }
        if (firstname == null ||
                firstname.trim().isEmpty()) {
        errors.put("firstname", "Bitte geben Sie einen
        Vornamen ein");
        }
        if (product == null || product.trim().isEmpty())        {
                errors.put("product", "Bitte geben Sie einen
                Artikel ein");
        }

  out.println("<html><head><title>Bestellung</title></head>");
  out.println("<body>");
        if (errors.isEmpty()) {
        // Formular ist OK -> Bestellung speichern
        Store store = Store.getInstance(getServletContext());
        String customer = firstname+" "+lastname;
        Long orderId = store.addOrder(customer, product);
        out.println("<h2>Bestellung erfolgreich</h2>");
        out.println("<p>Vielen Dank "+firstname+"
                "+lastname+".</p>");
        out.println("<p>Ihre Bestellung fÃ¼r Produkt '" + product + "'
wurde unter der Bestellnummer "+orderId+" gespeichert!</p>");
                } else {
                // Fehler im Formular -> Fehlermeldung
                out.println("<h2>Bestellung
                        fehlgeschlagen</h2>");
                out.println("<p>Korrigieren Sie bitte
                        folgende Fehler:</p>");
                out.println("<ul>");
                for (String error : errors.values()) {
                out.println("<li>"+error+"</li>");
                }
                        out.println("</ul>");
                }
                out.println("<p><a
        href=\"view/OrderForm.html\">Zur&uuml;ck zum
        Formular</a></p>");
  out.println("</body></html>");
        }

}
```