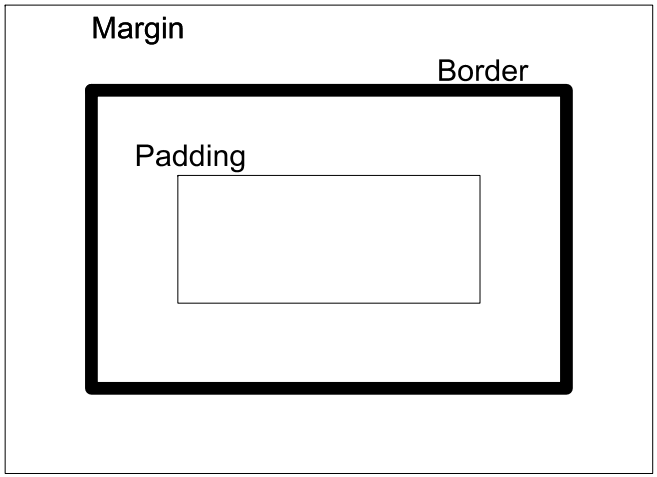
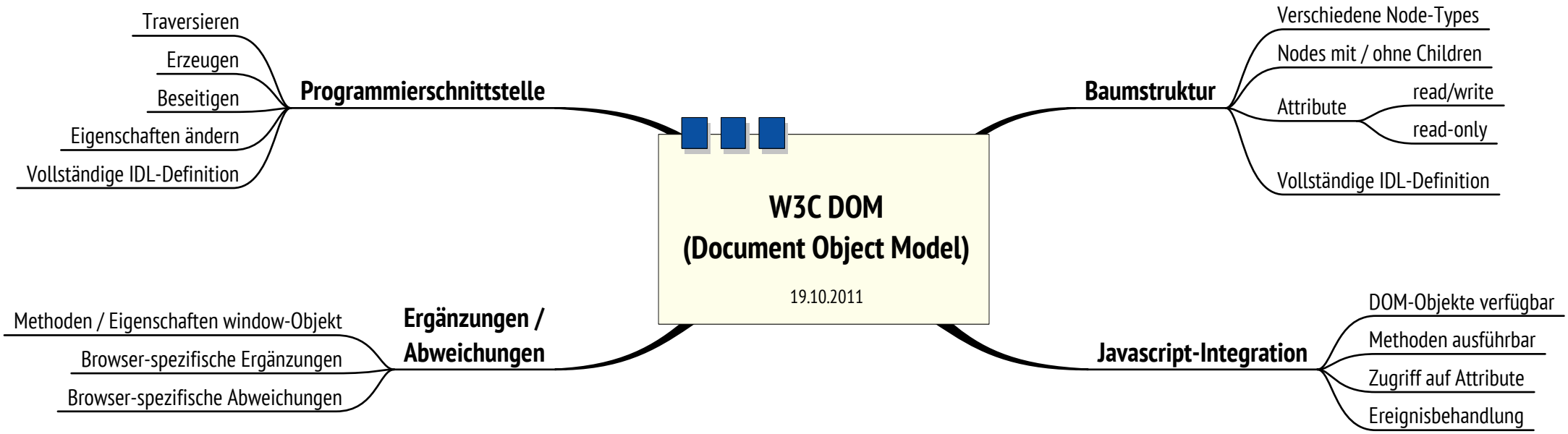
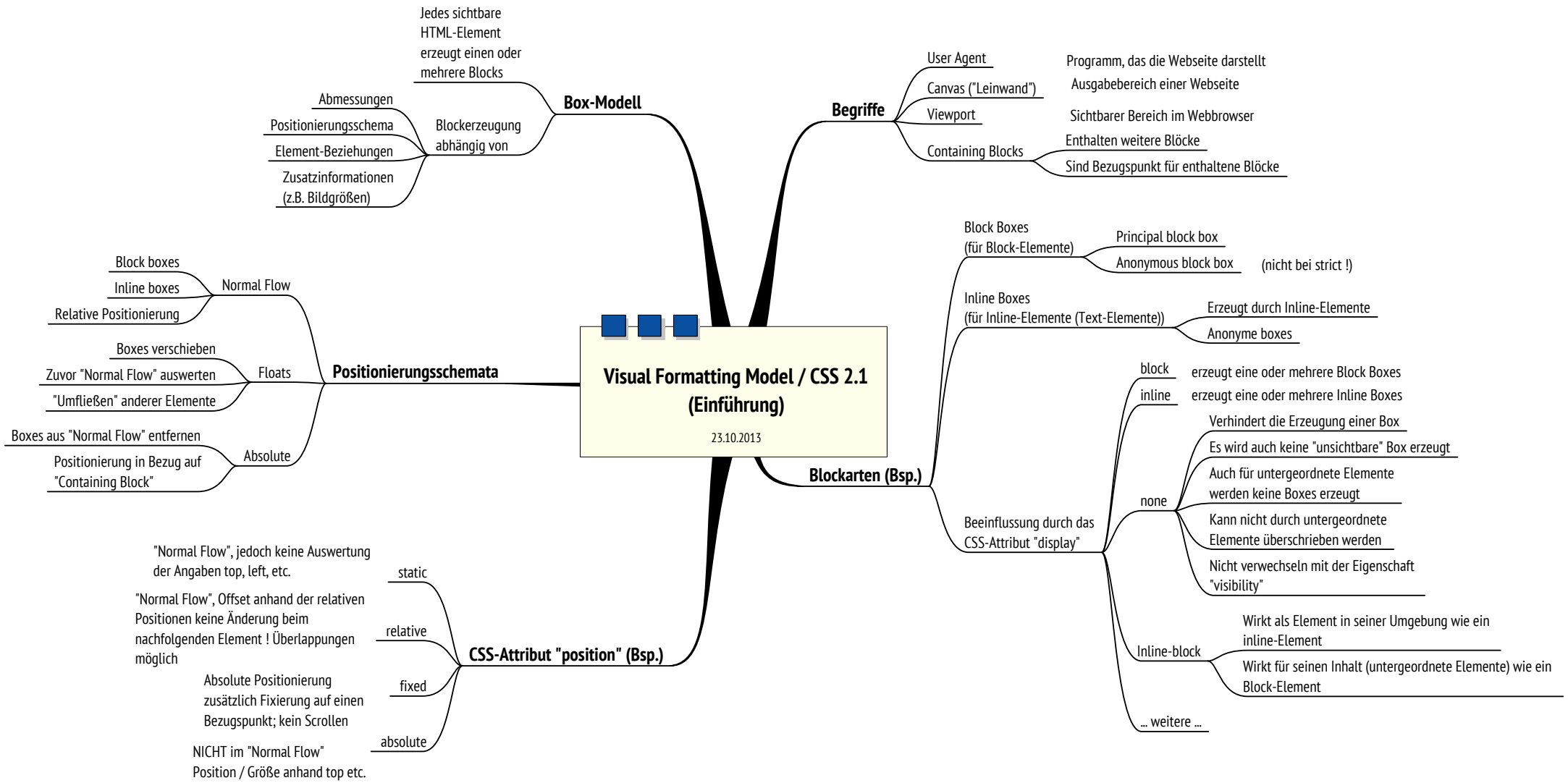


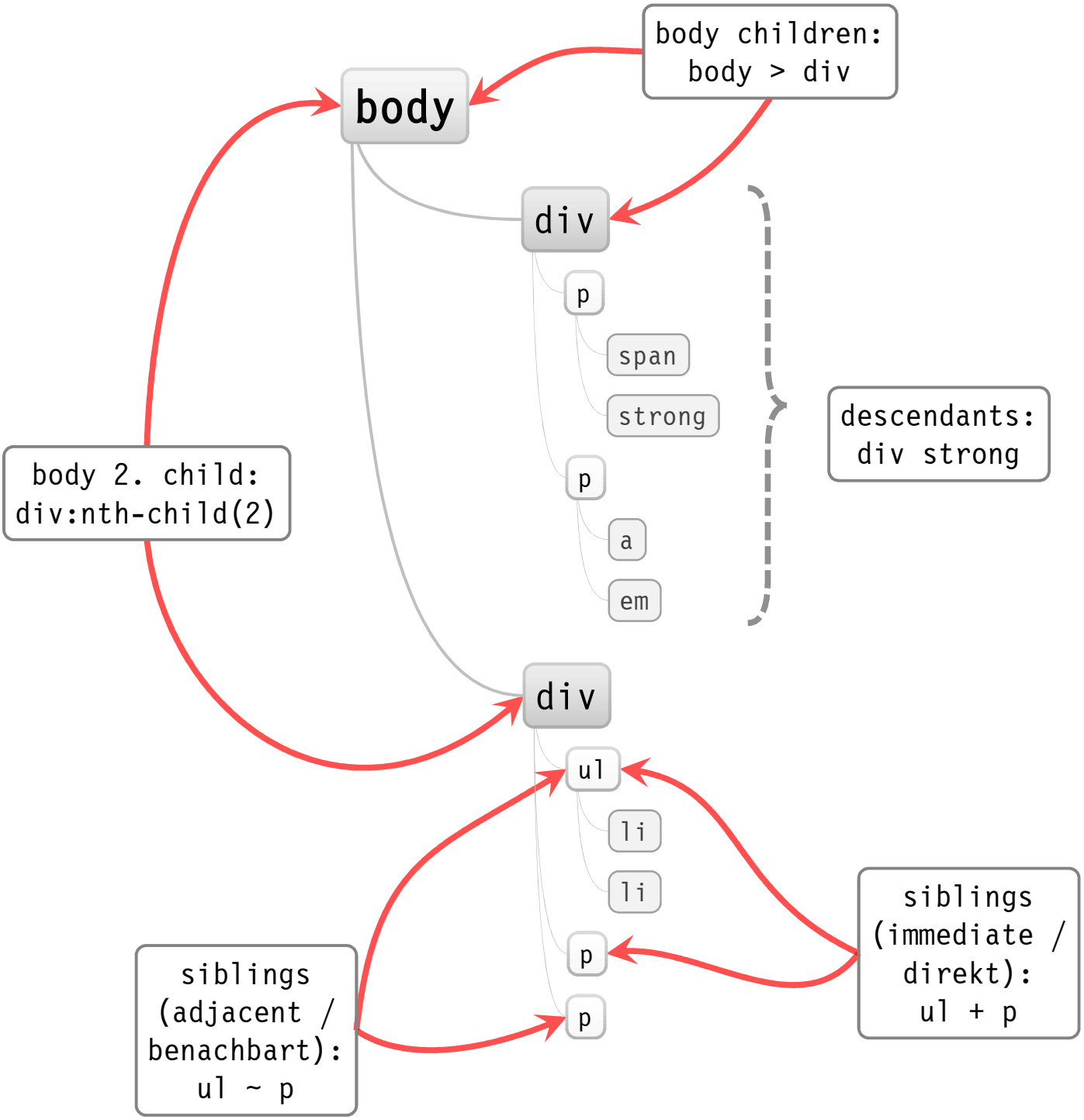
CSS / Basiseigenschaften (CSS 1 / CSS 2) (Auswahl)

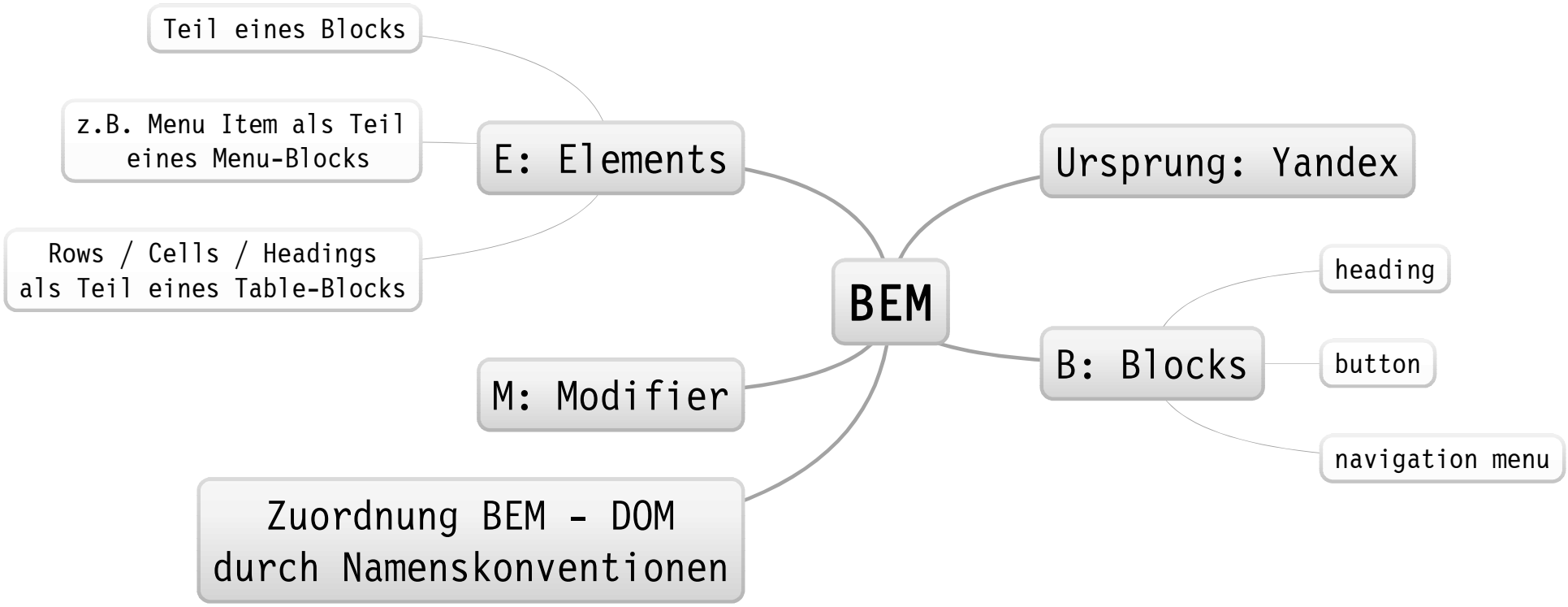
19.10.2011

- **Tabellen**
  - table-layout
  - border-collapse
  - border-spacing
  - empty-cells
- **Listen**
  - list-style-type
  - list-style-image
  - list-style-position
- **Größe**
  - width
  - height
- **Füllung**
  - padding
- **Abstand**
  - margin
- **Umrandung**
  - border-width
  - border-color
  - border-style
- **Schrift**
  - font-family
  - font-style
  - font-weight
  - font-size
- **Farben / Hintergrund**
  - color
  - background-color
  - background-image
  - background-repeat
  - background-position
- **Text**
  - text-decoration
  - text-transform
  - letter-spacing
  - word-spacing
- **Absatz**
  - line-height
  - text-ident
  - white-space
  - text-align
  - vertical-align


W3C DOM
(Document Object Model)
19.10.2011
Programmierschnittstelle
Traversieren
Erzeugen
Beseitigen
Eigenschaften ändern
Vollständige IDL-Definition
Baumstruktur
Verschiedene Node-Types
Nodes mit / ohne Children
Attribute
read/write
read-only
Vollständige IDL-Definition
Ergänzungen / Abweichungen
Methoden / Eigenschaften window-Objekt
Browser-spezifische Ergänzungen
Browser-spezifische Abweichungen
Javascript-Integration
DOM-Objekte verfügbar
Methoden ausführbar
Zugriff auf Attribute
Ereignisbehandlung


Visual Formatting Model / CSS 2.1 (Einführung)
23.10.2013
Box-Modell
Jedes sichtbare HTML-Element erzeugt einen oder mehrere Blocks
Blockerzeugung abhängig von
Abmessungen
Positionierungsschema
Element-Beziehungen
Zusatzinformationen (z.B. Bildgrößen)
Begriffe
User Agent
Programm, das die Webseite darstellt
Canvas ("Leinwand")
Ausgabebereich einer Webseite
Viewport
Sichtbarer Bereich im Webbrowser
Containing Blocks
Enthalten weitere Blöcke
Sind Bezugspunkt für enthaltene Blöcke
Positionierungsschemata
Normal Flow
Block boxes
Inline boxes
Relative Positionierung
Floats
Boxes verschieben
Zuvor "Normal Flow" auswerten
"Umfließen" anderer Elemente
Absolute
Boxes aus "Normal Flow" entfernen
Positionierung in Bezug auf "Containing Block"
Blockarten (Bsp.)
Block Boxes (für Block-Elemente)
Principal block box
Anonymous block box
(nicht bei strict !)
Inline Boxes (für Inline-Elemente (Text-Elemente))
Erzeugt durch Inline-Elemente
Anonyme boxes
Beeinflussung durch das CSS-Attribut "display"
block
erzeugt eine oder mehrere Block Boxes
inline
erzeugt eine oder mehrere Inline Boxes
none
Verhindert die Erzeugung einer Box
Es wird auch keine "unsichtbare" Box erzeugt
Auch für untergeordnete Elemente werden keine Boxes erzeugt
Kann nicht durch untergeordnete Elemente überschrieben werden
Nicht verwechseln mit der Eigenschaft "visibility"
Inline-block
Wirkt als Element in seiner Umgebung wie ein inline-Element
Wirkt für seinen Inhalt (untergeordnete Elemente) wie ein Block-Element
... weitere ...
CSS-Attribut "position" (Bsp.)
static
"Normal Flow", jedoch keine Auswertung der Angaben top, left, etc.
relative
"Normal Flow", Offset anhand der relativen Positionen keine Änderung beim nachfolgenden Element ! Überlappungen möglich
fixed
Absolute Positionierung zusätzlich Fixierung auf einen Bezugspunkt; kein Scrollen
absolute
NICHT im "Normal Flow"
Position / Größe anhand top etc.


body children:
body > div
body
div
p
span
strong
p
a
em
descendants:
div strong
body 2. child:
div:nth-child(2)
div
ul
li
li
p
p
siblings
(immediate /
direkt):
ul + p
siblings
(adjacent /
benachbart):
ul ~ p


BEM
Ursprung: Yandex
B: Blocks
heading
button
navigation menu
E: Elements
Teil eines Blocks
z.B. Menu Item als Teil eines Menu-Blocks
Rows / Cells / Headings als Teil eines Table-Blocks
M: Modifier
Zuordnung BEM - DOM durch Namenskonventionen

CSS: Media Queries

```
@media [not|only] type and (expression) [AND (expression)]
```

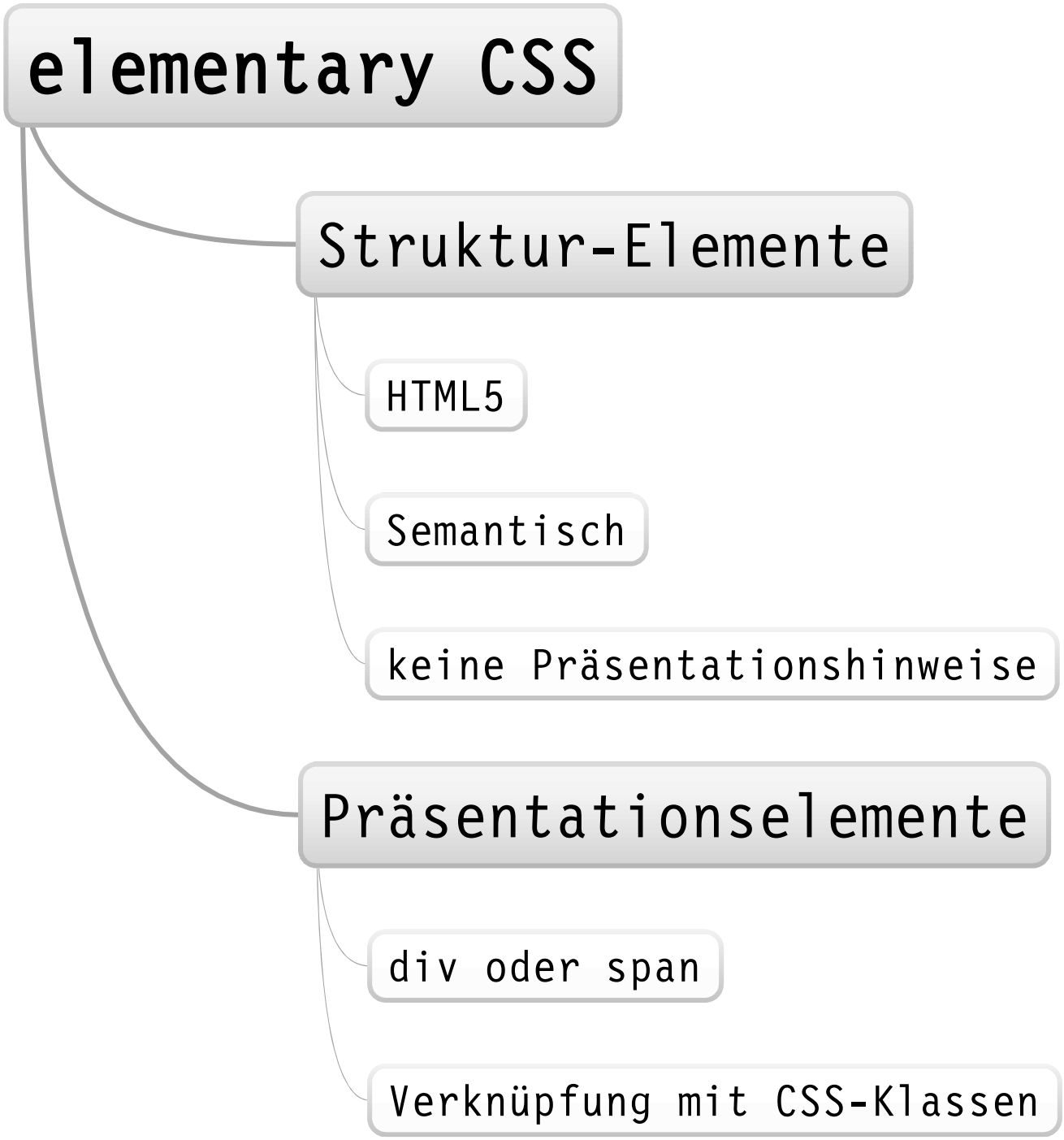
- media features
  - Größe viewport / Ausgabegeräte
  - Farbtiefen
  - Ausrichtung
- media type

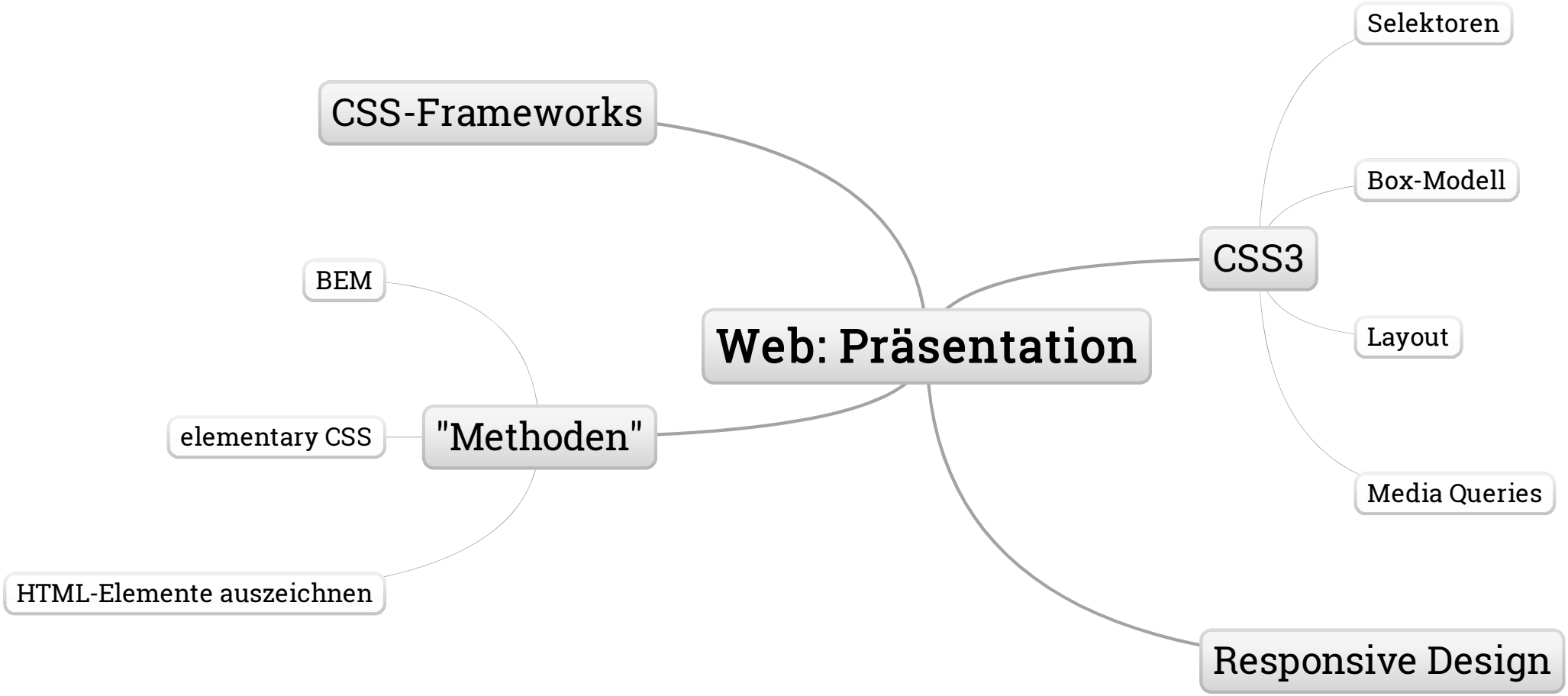
```
Angaben zu media type:
<link ... media="print" />
oder in style-Element
@media print {
   ...
}
```

- all
- braille
- embossed (Drucker Blindenschrift)
- handheld
- print
- projection
- screen
- speech (Screenreader)
- tty
- tv
- Viewport-Meta-Tag

```
<meta name="viewport" content="width=800px, initial-scale=1.0" />
```

(damit Viewport auf eine bestimmte Größe und intiale Skalierung festlegen)


elementary CSS
Struktur-Elemente
HTML5
Semantisch
keine Präsentationshinweise
Präsentationselemente
div oder span
Verknüpfung mit CSS-Klassen


Web: Präsentation
CSS-Frameworks
CSS3
Selektoren
Box-Modell
Layout
Media Queries
"Methoden"
BEM
elementary CSS
HTML-Elemente auszeichnen
Responsive Design

Webkrauts

- für mehr Qualität im Web

Ablösung für Float-Layouts

CSS3 Flexbox

Ablösung für Float-Layouts

Mehrspaltige Designs setzen Webworker heutzutage mit Floats um. Das mag sich in Zukunft ändern. Denn mit dem CSS3-Modul Flexbox steht eine nützliche Alternative in den Startlöchern. HTML-Elemente können damit spalten- oder zeilenweise angezeigt werden, die Reihenfolge der Elemente lässt sich steuern und Breiten oder Höhen passen sich automatisch an.

Anhand des CSS3-Flexbox-Moduls wird deutlich, wie dynamisch der W3C-Standard ist. Wo einerseits CSS-Eigenschaften wie `box-shadow` oder `border-radius` mittlerweile ohne Vendor-Präfix (z.B. `-moz-`, `-webkit-`) verwendet werden können, hat die Browser-Implementierung von `flex`-Eigenschaften gerade erst begonnen.

Der erste Entwurf des Flex-Moduls vom W3C stammt aus 2009, was im Vergleich zu anderen Modulen sehr jung ist. Trotzdem hat das Flex-Modul seit September 2012 den Status »Candidate Recommendation« – was so viel bedeutet, dass es keine tiefergehenden Änderungen mehr geben wird. Vom Entwurf bis zum jetzigen Stand hat sich das Modul mehrfach stark verändert, was den Umfang, vor allem aber die Bezeichnungen der CSS-Eigenschaften betrifft. Im Web sind Beispiele für Firefox zu finden, die mit `-moz-box-flex` arbeiten, oder für den Internet Explorer, die `-ms-flexbox` als Bezeichner verwenden.

Dieser Beitrag beschäftigt sich jedoch nicht mit dem, was in der Vergangenheit ausprobiert wurde, sondern soll einen allgemeinen Überblick verschaffen, welche Möglichkeiten Flexbox in Zukunft bieten wird. `flex`-Eigenschaften werden aktuell von Opera (ab der Version 12.10) sowie von Google Chrome (ab der Version 21.0) mit dem Vendor-Präfix `-webkit-` unterstützt (mobile Browser sind in diesem Beitrag nicht berücksichtigt). Von daher wird in den CSS-Beispielen die Schreibweise für den Chrome sowie die Standardschreibweise gezeigt. Die Demo-Quelltexte findet ihr am Ende des Beitrags.

Was macht Flexbox?

Vereinfacht gesagt lassen sich in einer Flexbox Elemente in Reihen oder Spalten anzeigen. HTML-Elemente passen sich je nach Deklaration dem verfügbaren Platz an und können unterschiedlich ausgerichtet werden. Auch die Reihenfolge innerhalb der Reihen und Spalten ist individuell veränderbar.

Die Grundlage für die folgenden Beispiele stellt ein einfaches HTML-Konstrukt dar, welches das Grundgerüst jeder Webseite sein könnte. Die Beispiele sind in den Quelltexten fortlaufend zu finden, das x in der `id="beispiel-x"` ist dabei durch die jeweilige Beispielnummer ersetzt.

```
1. <div id="beispiel-x">
2.   <nav>Navigation</nav>
3.   <section>Inhalt</section>
4.   <aside>Weiteres</aside>
5. </div>
```

Über CSS wird für jedes Element eine eigene Hintergrundfarbe, zusätzlich Rahmen und runde Ecken für alle Elemente vergeben. Dies hat noch nicht mit dem Flexbox-Modul zu tun, sondern dient lediglich der Anschaulichkeit:

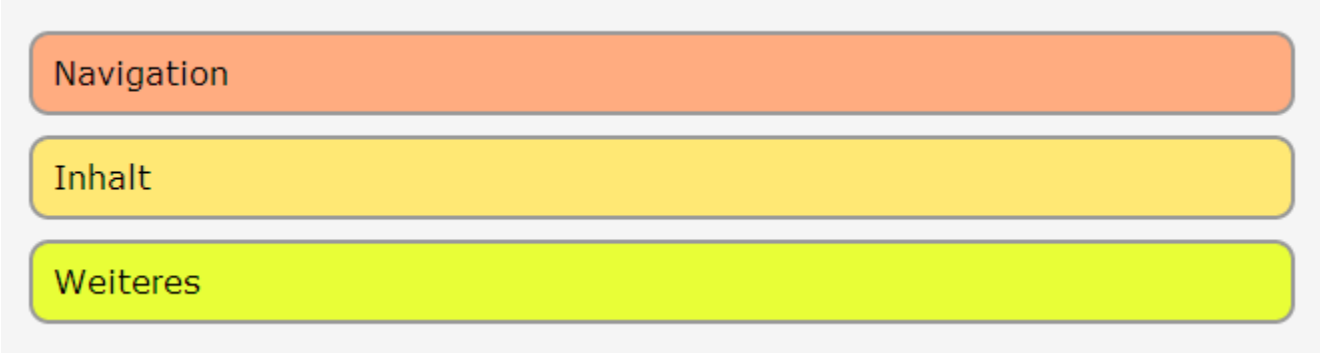


Abbildung 1: Darstellung des einfachen HTML-Codes

Grundlage für die `flex`-Darstellung

Die CSS-Eigenschaft `display` hat mit CSS3 zwei neue Parameter bekommen: `flex` und `inline-flex`. Eine dieser Eigenschaften muss dem umgebenden `div` zugewiesen werden, was die Flexbox-Darstellung »aktiviert«. Hier, wie in allen folgenden Beispielen, wird neben der Standardschreibweise die Eigenschaft mit dem Vendor-Präfix `-webkit-` eingesetzt, damit Google Chrome diese erkennt. Safari unterstützt die Eigenschaften jedoch nicht, auch wenn dies ein Webkit-Browser ist.

```
#beispiel-01 {
  display: -webkit-flex;
  display: flex;
}
```

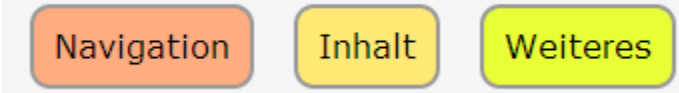


Abbildung 2: Darstellung mit `display: flex`

Dies erinnert zuerst an die Darstellung per `float: left`. Allerdings, und dies ist der große Vorteil gegenüber `float`: nachfolgende Elemente sind von `display: flex` nicht betroffen und umfließen die Flexbox nicht. Vergleichbares wie ein `clear: both` ist nicht notwendig.

Dass sich die einzelnen HTML-Elemente nun nebeneinander anordnen, liegt daran, dass mit dem Setzen von `display: flex` automatisch die CSS-Eigenschaft `flex-direction` greift. Diese hat den Standard-Parameter `row`. Würdest du die Eigenschaft `flex-direction` explizit hinterlegen, sähe die vollständige CSS-Regel wie folgt aus:

```
#beispiel-01 {
  display: -webkit-flex;
  display: flex;
  -webkit-flex-direction: row; /* Standard */
  flex-direction: row; /* Standard */
}
```

Alternativ kannst du `flex-direction: column` vergeben, allerdings werden dann die drei HTML-Element in einer Spalte angezeigt; die Browserdarstellung entspricht somit Abbildung 1.

Angabe einer relativen Basis-Breite

Die `flex`-Eigenschaften werden eigentlich erst dann spannend, wenn Breiten eingesetzt werden:

```
#beispiel-02 nav,
#beispiel-02 aside {
  -webkit-flex-basis: 20%;
  flex-basis: 20%;
```

```
5. }
6.
7. #beispiel-02 section {
8.     -webkit-flex-basis: 60%;
9.     flex-basis: 60%;
10. }
```

Mit diesen CSS-Regeln werden die Navigation und das `aside`-Element auf 20% gesetzt. Der Inhaltsbereich wird mit einer Breite von 60% versehen. Solch ein Laoyut würden wir heutzutage mit floatenden divs realisieren (bzw., so was wurde früher gerne über ein Tabellen-Layout umgesetzt). Die Darstellung nimmt nun die ganze Breite ein:

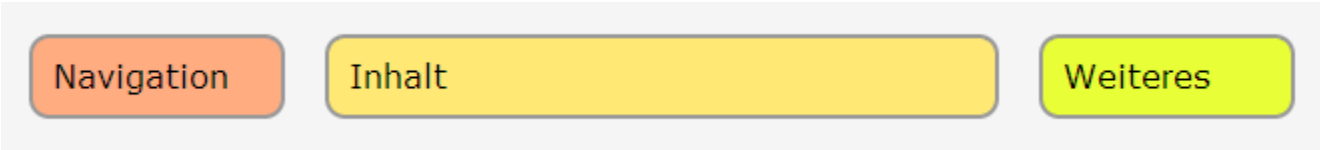


Abbildung 3: Automatische Verteilung der einzelnen Blöcke nach Einsatz von `flex-basis`.

Allerdings stellt `flex-basis` keine definitive Breite oder Höhe dar, sondern dient als relative Grundeinheit, nach dem sich die einzelnen Elemente proportional innerhalb der Flexbox anpassen. Ist die Summe der Ausmaße geringer als die Breite bzw. Höhe der umgebenen Box, wird auch nur dieser Platz eingenommen. Wobei in der Breite ohne weitere Angaben 100% der verfügbaren Breite relevant ist, in der Höhe muss die Flexbox mit einer bestimmten Höhe versehen sein, damit sich die Elemente verhältnismäßig anpassen. Wenn hingegen die Summe aller Breiten bzw. Höhen mehr als 100% ergeben, oder feste Pixel-Werte vergeben werden, die zusammen addiert mehr als die Dimension der Flexbox einnehmen, passen sich die in der Flexbox liegenden Elemente automatisch an den verfügbaren Platz an:

```
1. #beispiel-03 section {
2.   -webkit-flex-basis: 120%;
3.   flex-basis: 120%;
4. }
```

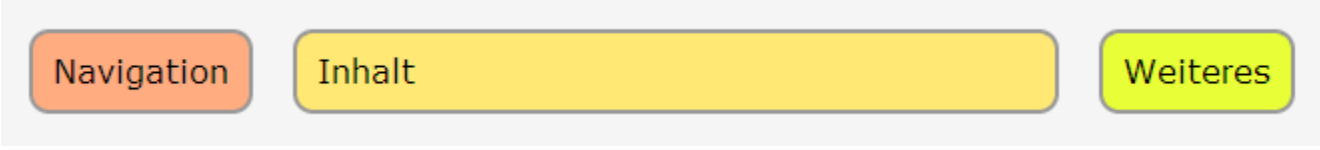


Abbildung 4: Flex-Elemente passen sich automatisch an den verfügbaren Platz an, auch wenn sie mehr als 100% in der Eigenschaft `flex-basis` aufweisen.

Der große Vorteil eines Flex-Layouts wird in Abbildung 4 deutlich. Selbst wenn ein `flex-basis`-Wert zu hoch vergeben wird, sortieren sich die Elemente trotzdem automatisch an, lediglich mit andern Breitenverhältnissen. Dies unterscheidet sich wesentlich von einem floatenden `div`-Layout, bei dem eine zu hohe Breite, oder aber auch `margin`- oder `padding`-Werte dazu führen, dass Elemente in die nächste Reihe rutschen, wenn die verfügbare Gesamtbreite überschritten ist.

Automatische Höhenanpassung der Blöcke

Der nächste Schritt zeigt das, was eigentlich jeder Frontend-Entwickler schon immer wollte: drei Elemente nebeneinander, unabhängig vom Inhalt auf die gleiche Höhe zu bringen. Genaugenommen ist dies ja der Grund, warum früher jahrelang Tabellenlayouts genutzt wurden, da diese als einziger Weg erschienen, ein mehrspaltiges Layout zu entwickeln, bei dem alle Spalten gleich hoch sind. Diesen Effekt bei einem floatendem `div`-Layout zuverlässig zu erzielen, war bislang nur mit verschiedenen Workarounds möglich, wie durch den geschickten Einsatz von Hintergrundgrafiken oder auch der Anpassung der Höhen über JavaScript. Flexbox »erledigt« dies mit nur einer Eigenschaft:

```
1. #beispiel-04 {
2.   display: -webkit-flex;
3.   display: flex;
4.   -webkit-align-items: stretch;
5.   align-items: stretch;
6. }
```

`align-items` hat den Standardwert `stretch` und müsste hier nicht gesetzt werden. Auch ohne diese explizit gesetzte Eigenschaft werden alle Flexbox-Elemente auf die gleiche Höhe gestreckt. Alleine die Höhenangabe reicht aus, auch wenn sie nur für ein Element in einer Reihe vergeben ist, damit alle Elemente die gleiche Höhe annehmen.

```
1. #beispiel-04 section {
2.   min-height: 150px;
3. }
```

Die CSS-Regel setzt lediglich den Inhaltsblock auf mindestens 150px, Navigation und `aside`-Element haben keine zusätzliche Deklaration:

Abbildung 5: Boxen strecken sich standardmäßig auf die gleiche Höhe.

Für `align-items` können andere Werte vergeben werden, welche die Anordnung der Boxen auf der Y-Achse festlegt. In den folgenden Beispielen sind die Parameter `flex-start`, `flex-end` und `center` gezeigt:

Abbildung 6: `align-items: flex-start`

Abbildung 7: `align-items: center`

Abbildung 8: `align-items: flex-end`

Spätestens jetzt wird das Potential der Flexbox deutlich. Sie bietet einerseits das `float`-Verhalten und sortiert Boxen nebeneinander, ohne aber die unerwünschten Nebeneffekte von `float` zu haben. Zusätzlich stehen Möglichkeiten der horizontalen Ausrichtung zur Verfügung, wie bislang nur über Tabellen, ggf. geschickter Positionierung oder über Workarounds mit JavaScript zu realisieren war.

Verändern der Reihenfolge

Das Flexbox-Modul bietet noch mehr. Über einen anderen Parameter für `flex-direction` kann die Reihenfolge der Blöcke umgedreht werden:

```
1. #beispiel-05 {
2.   -webkit-flex-direction: row-reverse;
3.   flex-direction: row-reverse;
4. }
```

Abbildung 9: Umgekehrte Reihenfolge der Blöcke per `row-reverse`

Nicht nur die Umkehrung der Blöcke von links nach rechts ist möglich, auch von oben nach unten. Dazu dient der Wert column-reverse für die CSS-Eigenschaft `flex-direction`.

```
1. #beispiel-06 {
2.   -webkit-flex-direction: column-reverse;
3.   flex-direction: column-reverse;
4. }
```

Abbildung 10: Umgekehrte Reihenfolge innerhalb einer Spalte per `flex-direction: column-reverse`

In Abbildung 10 wird nicht nur deutlich, dass die Blöcke von unten nach oben dargestellt werden. Ebenfalls ist hier zu erkennen, wie sich die oben vergebene Höhe für das `section`-Element auswirkt: Boxen werden auf eine Höhe gestreckt, wenn sie innerhalb einer Reihe angezeigt werden. Werden mehrere Boxen in mehreren Reihen angezeigt, haben die Höhen in verschiedenen Reihen keinen Einfluss aufeinander.

Aber nicht nur das Umdrehen der Boxen ist möglich. Auch die konkrete Reihenfolge kann über die neue CSS-Eigenschaft `order` gesteuert werden:

```
1. #beispiel-07 nav {
2.   -webkit-order: 3;
3.   order: 3;
4. }
```

```
5.
6.  #beispiel-07 section {
7.    -webkit-order: 1;
8.    order: 1;
9.  }
10.
11. #beispiel-07 aside {
12.   -webkit-order: 2;
13.   order: 2;
14. }
```

Dabei wird über diese CSS-Regeln und über die Eigenschaft `order` jedem einzelnem Element innerhalb der Flexbox eine bestimmte Position zugewiesen. Sollte die Reihenfolge durch `column-reverse` oder `row-reverse` grundsätzlich verdreht sein, greift auch die eigens gesetzte Reihenfolge umgekehrt. Oben definierte CSS-Regeln wirken sich wie folgt aus:

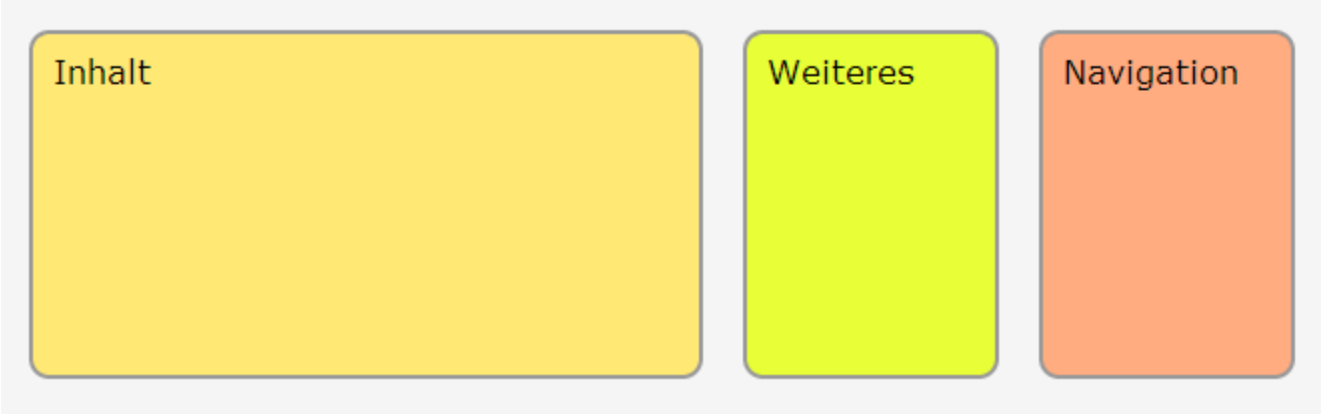


Abbildung 11: Per `order` kann die Reihenfolge der Elemente in der Flexbox konkret angegeben werden.

Umbruch von Elementen in einer Flexbox

Oben wurde über `flex-basis` die Basis-Breite für die einzelnen Elemente angegeben, was dazu führte, dass sich die Boxen automatisch auf die verfügbare Breite verteilten. Anders verhalten sich Elemente in einer Flexbox, wenn sie mit einer festen Breite per `width` oder `min-width` versehen sind. Zur Verdeutlichung wird der HTML-Code um zwei div-Elemente erweitert:

```
1. <div id="beispiel-08">
2.   <nav>Navigation</nav>
```

```
    <section>Inhalt</section>
    <aside>Weiteres</aside>
    <div class="box-4">Box 4</div>
    <div class="box-5">Box 5</div>
</div>
```

Alle Elemente in der Flexbox werden im nächsten Schritt mit einer festen Breite versehen:

```
#beispiel-08 nav,
#beispiel-08 aside,
#beispiel-08 section,
#beispiel-08 div {
  width: 150px;
  min-height: 100px;
}
```

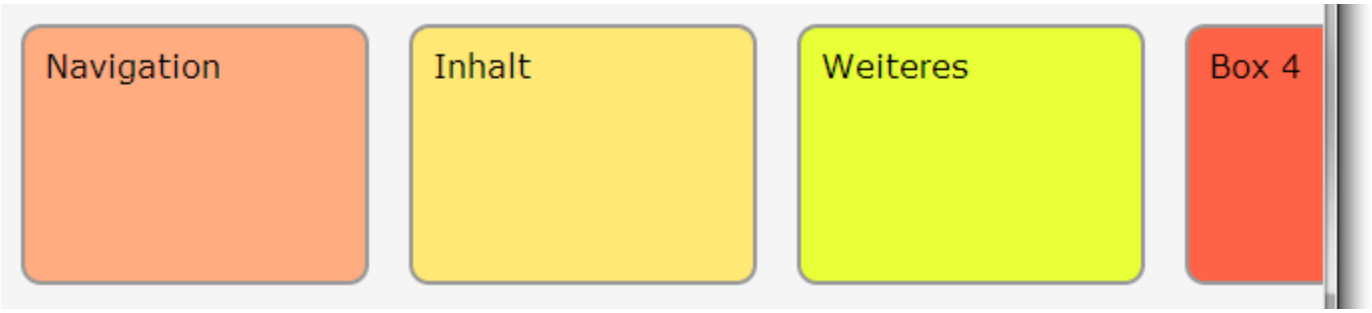


Abbildung 12: Elemente mit fester Breite fließen über die umgebene Flexbox (hier über die Größe des Browserfensters) hinaus.

Dieses Verhalten in Abbildung 12 erinnert daran, wenn ein `overflow: hidden` gesetzt sei. Schlechter als beim `float`, brechen die Elemente nicht in die nächste Reihe um, sondern fließen aus dem Layout heraus.

Für diesen Fall bietet das Flexbox-Modell eine weitere Eigenschaft: `flex-wrap` mit dem Parameter `wrap`. Standardwert für `flex-wrap` ist `nowrap` und dafür verantwortlich, dass die einzelnen Elemente innerhalb der Flexbox nicht umgebrochen werden und, wie in der Abbildung 12, über die Dimensionen der Flexbox hinaus laufen.

```
#beispiel-09 {
  -webkit-flex-wrap: wrap;
  flex-wrap: wrap;
}
```

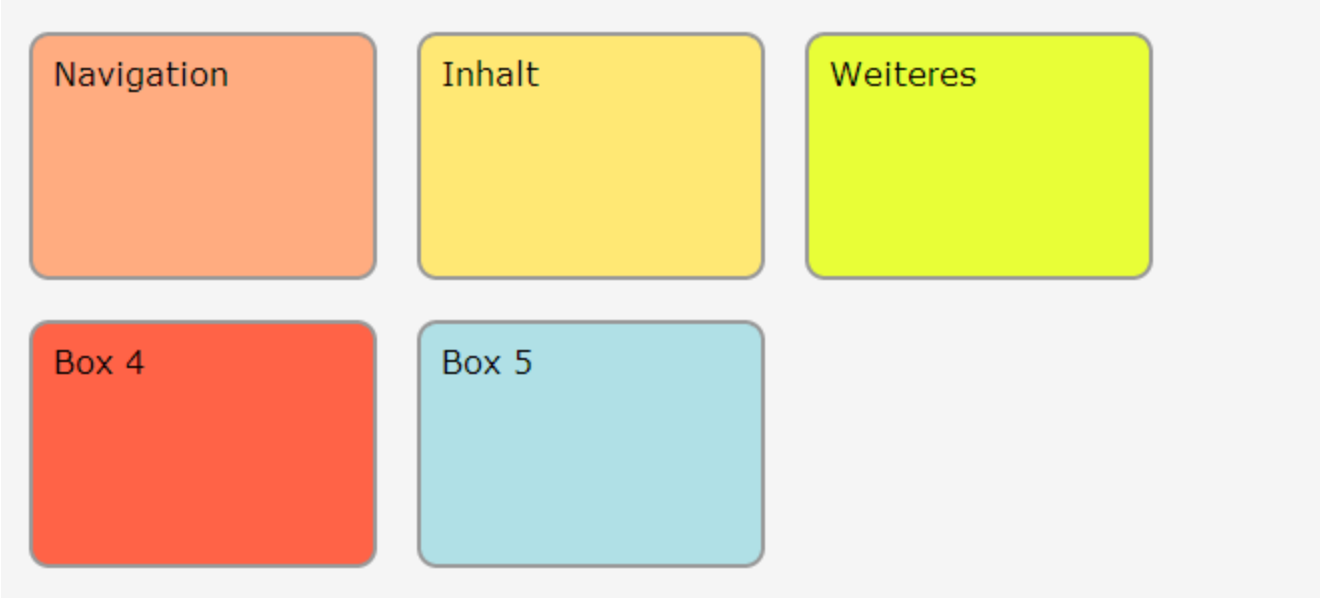


Abbildung 13: Per `flex-wrap: wrap` brechen Elemente in eine neue Reihe um

Ausdehnen von Elementen mit fester Breite

Zusätzlich kann das Ausdehnungsverhalten von Elementen in Flexboxen verändert werden. Die Eigenschaft `flex-grow` beeinflusst, dass auch Boxen mit festen Breiten gedehnt werden. Hier sich nummerische Werte möglich. Sind zwei Boxen in einer Flexbox, eine davon mit dem Wert `flex-grow: 1`, die andere mit dem Wert `flex-grow: 5`, dann nimmt letztere Box mehr Raum als die erste ein.

```
1. #beispiel-10 nav,
2. #beispiel-10 aside,
3. #beispiel-10 section,
4. #beispiel-10 div {
5.   width: 150px;
6.   -webkit-flex-grow: 1;
7.   flex-grow: 1;
8.   -webkit-flex-basis: auto;
9.   flex-basis: auto;
10. }
```

In diesem Beispiel werden alle fünf Boxen mit `flex-grow: 1` versehen, was dazu führt, dass sich alle Boxen im Verhältnis gleich vergrößern. Zusätzlich wird noch eine weitere Eigenschaft benötigt: `flex-basis: auto`. Das `auto` für `flex-basis` sorgt dafür, dass sich die Boxen auf die volle Breite ausdehnen.

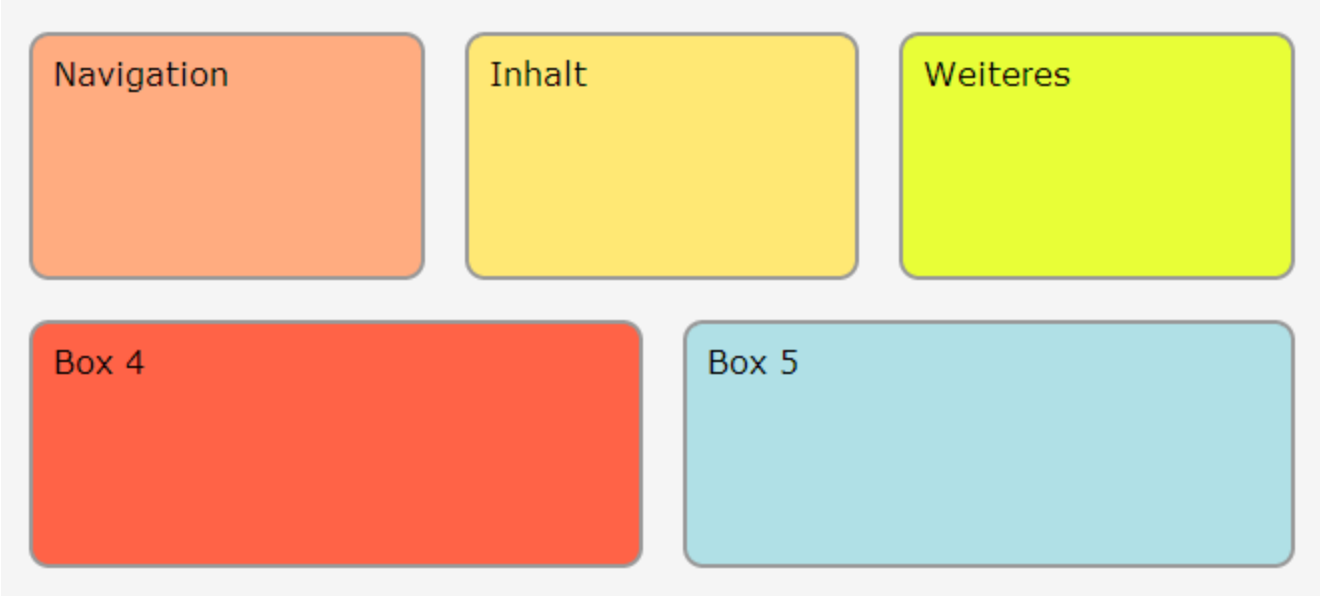


Abbildung 14: Per `flex-grow: 1` und `flex-basis: auto` werden die einzelnen Boxen auf die ganze Breite gestreckt ndash; und dies jeweils pro Reihe.

Responsive Design und Flexbox

Genau genommen ist das letzte Beispiel vor allem für Responsive Design relevant, soll aber hervorheben, wie einfach Flexbox mit der `@media`-Regel kombiniert werden kann. Grundlage hierfür ist das Beispiel 4, das in Abbildung 5 zu sehen ist. Die Darstellung einschließlich Browserfenster sieht wie folgt aus:

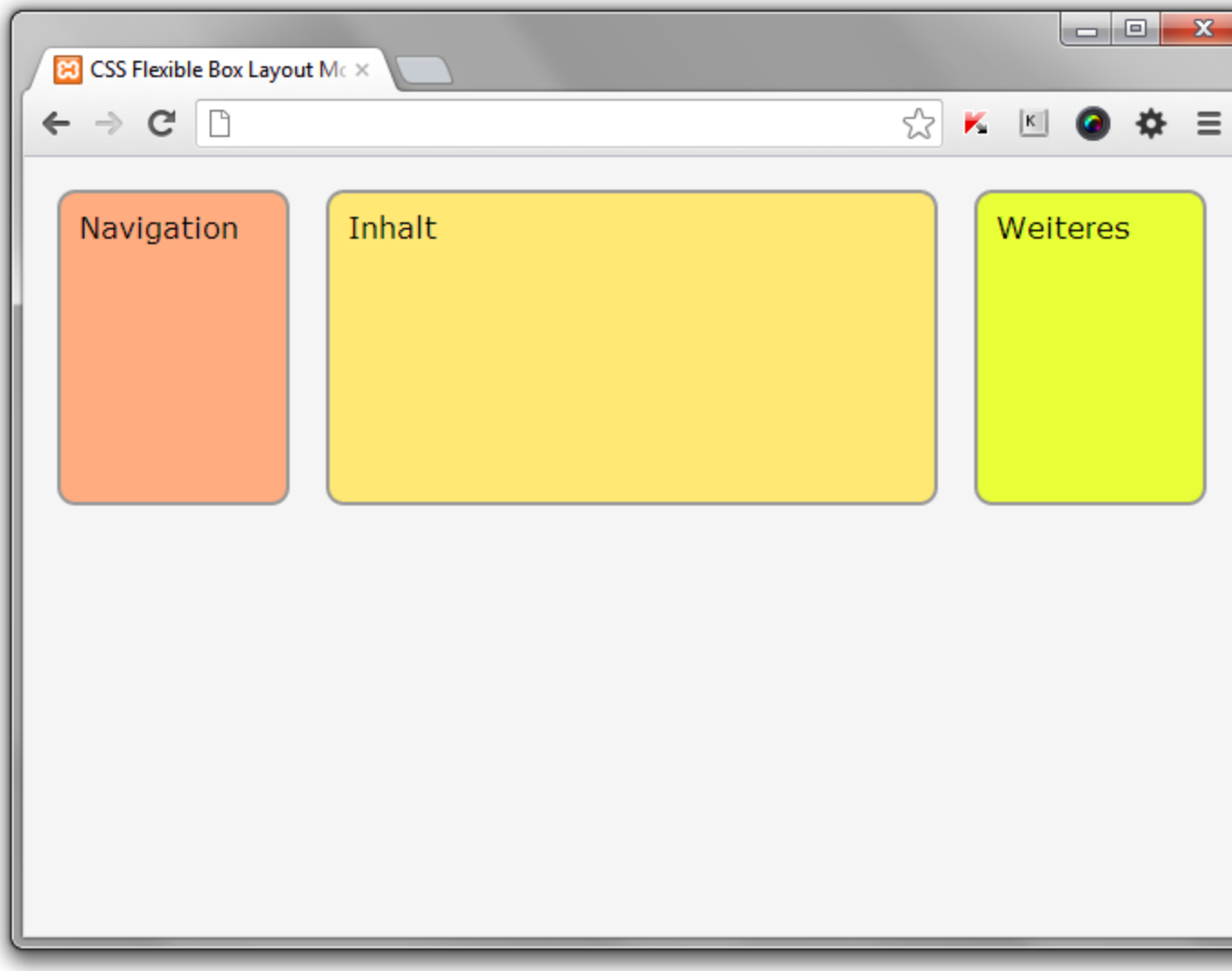


Abbildung 15: Bei normaler Breite greift in der Flexbox die reihenweise Darstellung der einzelnen Elemente.

Zusätzlich wird ein Media-Query definiert:

```
1. @media all and (max-width: 500px) {
2.   #beispiel-11 {
3.     -webkit-flex-direction: column;
4.     flex-direction: column;
5.   }
6. }
```

Diese Regel sorgt dafür, dass unter einer Fensterbreite von 500px die Darstellung in der Flexbox von der reihenweisen Darstellung (`row`) auf die spaltenweise Darstellung (`column`) umschaltet. Die Darstellung desselben HTML-Codes sieht dann wie folgt aus:

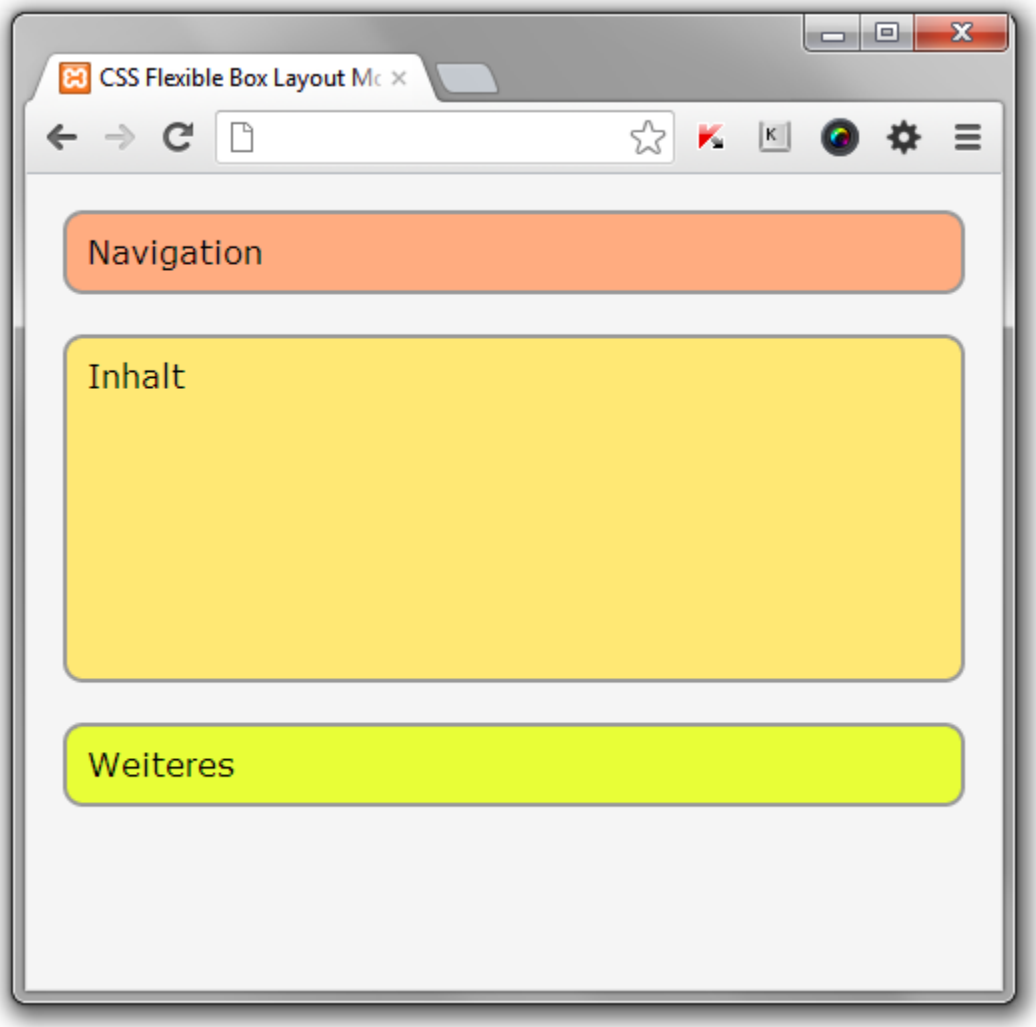


Abbildung 16: Bei geringer Fensterbreite springt das Layout zu einer spaltenweisen Darstellung um.

Fazit

Das Flexbox-Modul des CSS3-Standards bietet vielfältige Möglichkeiten, welche die Eigenschaften von fließenden Boxen per `float` mit den Verhalten von Tabellen kombiniert, zusätzlich aber noch weitere Steuerungsmöglichkeiten der Elemente innerhalb einer Flexbox ermöglicht.

Langfristig können so einfach stabile Layouts aufgebaut werden. Ab wann das genau der Fall sein wird, ist heute nicht eindeutig zu sagen. Firefox hat die Implementierung des

Flex-Moduls für die Version 18 angekündigt (erscheint im Januar 2013); vor allem wird es aber vom Internet Explorer abhängen, wann `flex`-Eigenschaften browserübergreifend eingesetzt werden können. Behält Microsoft die Veröffentlichungzyklen seines Browsers bei, ist der Internet Explorer 11 in den nächsten ein bis drei Jahren zu erwarten. Inwiefern dieser das Flex-Modul implementiert sein wird, ist heute nicht abzusehen.

Weiterführende Links

- http://css-tricks.com/old-flexbox-and-new-flexbox/
- http://msdn.microsoft.com/en-us/library/ie/hh673531%28v=vs.85%29.aspx
- https://developer.mozilla.org/en-US/docs/CSS/flex
- http://caniuse.com/#feat=flexbox
- http://www.w3.org/TR/css3-flexbox/
- Demoquelltexte zum Download

Der Autor

Stephan Heller [daik.de] arbeitet seit 2004 als Freelancer.
Er ist ZEND zertifizierter-PHP-Entwickler, hat derweil seinen Schwerpunkt auf die

Frontendentwicklung mit CSS und HTML verlagert. Vor allem mit dem Ziel, komplizierte Dinge einfach zu machen und am Ende gut nutzbare und durchdachte Webprojekte zu haben.
Im September 2012 ist sein Buch »Workshop HTML5 & CSS3 - Weblayouts professionell umsetzen« erschienen.

@daik_de | Autorenprofil

Der Artikel

veröffentlicht am:
11. Dezember 2012

- Serie: Adventskalender 2012
- Kommentare: 9

Care & Share

- Als E-Mail senden
- Auf Google+ teilen
- Auf Facebook teilen
- Tweet this
- delicious-Bookmark

Kommentare

Sören Hentzschel
am 11.12.2012 - 10:55

Danke für den Artikel, der war wirklich sehr gut zu lesen! :)

Kleine Korrektur: Mozilla plant den flexbox-Support offziell für Firefox 20:
http://blog.dholbert.org/2012/12/css3-flexbox-enabled-in-nightlies-ready...

Permanenter Link

Christian
am 11.12.2012 - 13:17

Wieder ein sehr gelungener Artikel. Vielen Dank.

Permanenter Link

Olaf Gleba (Webkraut)
am 11.12.2012 - 14:15

Sehr schöner Artikel, danke. Und kommt zur richtigen (Weihnachts)Zeit, wo man ja den ein oder anderen Wunsch im Kopf spazieren führt. Auch wenn sich das wohl eher auf Weihnachten 2013 bezieht. Aber Vorfreude ist ja feines.

Auch wenn *floats* als Layout-Krücke etabliert sind, bin ich gerade recht begeistert, was mit *display: inline-block* alles möglich ist. Auch eine Krücke, ja, aber mit weniger Nebenwirkung und Möglichkeiten, die floats nicht bieten.

Permanenter Link

Alex
am 11.12.2012 - 14:37

DANKE! Auch für Hobby-Webbastler sehr verständlich und anschaulich geschrieben.

Permanenter Link

Aaron
am 11.12.2012 - 14:39

Danke für den Überblick! Flexbox wird das Bauen von Layouts einiges praktischer machen.

Permanenter Link

Paul
am 11.12.2012 - 21:17

Hmm, schon spannend, aber gab es nicht auch einmal einen Ansatz, der versucht klassische Layoutmanager im Standard zu integrieren. Fänd ich persönlich sinnvoller! Aber ich bin ja kein Webexperte..

Permanenter Link

Stephan Heller (Autor)
am 12.12.2012 - 11:19

Hallo Zusammen,
vielen Dank für das Feedback - ja, Flexbox ist noch Zukunftsmusik - und auch nur eine von vielen neuen Möglichkeiten.
Spannend ist es allemale, man darf wirklich gespannt sein, was sich durchsetzten wird.
Viele Grüße, Stephan

Permanenter Link

Mo
am 17.12.2012 - 09:49

Es gibt einen Polyfill für Flexbox: Flexie (https://github.com/doctyper/flexie)

Permanenter Link

Harry
am 21.12.2012 - 20:41

Danke für die interessante Darlegung.
Ich werde es im Hinterkopf behalten und bei Gelegenheit auch mal selbst ein wenig in diese Richtung testen. Allerdings solange die Sache noch nicht browserübergreifend eingesetzt werden kann, wird es (bei mir) wohl über die Testphase nicht hinausgehen.

Permanenter Link

Die Kommentare sind geschlossen.

Web-Engineering

1 / Einführung

Hochschule Niederrhein
University of Applied Sciences
Elektrotechnik und Informatik
Faculty of Engineering and Computer Science

World Wide Web

- Ressourcen, z.B. Dokumente, identifizierbar bereitstellen
- Hypertexte : enthalten Verweise auf Ressourcen
- Multimediale Komponenten (Ton/Bild/Video) einbeziehen
- Ergänzung / Integration anderer Internet-Dienste
  - Wie telnet, ftp, ssh, Email-Dienste (pop,imap,smtp) etc.
- Verwendung vorhandener Standardprotokolle
  - TCP, IP

Basiskonzepte (1)

- Bereitstellung Ressourcen
  - Client- / Server-Architektur
    - (viele) Webclients zur Anforderung und Anzeige von Ressourcen
    - (einzelne) Webserver zur zentralen Bereitstellung und Auslieferung der Ressourcen
    - Zustandsloses Protokoll zur Client-Server-Kommunikation :
      - Anforderung Ressource
      - Auslieferung Ressource

Basiskonzepte (2)

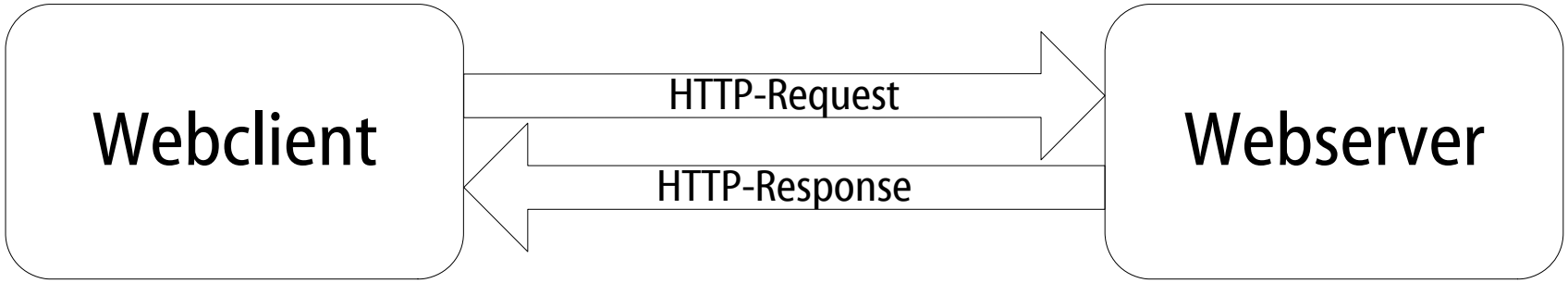
- Ressourcen / Dokumente
  - Hypertext
    - Verweise auf Ressourcen oder Marken in Dokumenten
  - Strukturierte Dokumente
    - Gliederung
    - Auszeichnung von Texten zur Kennzeichnung der Bedeutung
  - Präsentation
    - Standardisiert
    - benutzerdefiniert

Basiskonzepte (3)

Anforderung
Darstellung

Vermittlung

Bereitstellung
Auslieferung



Erweiterungen

- Ursprünglich weitgehend statische Sicht auf Dokumente
- Benutzerinteraktionen vorgesehen für
  - Die Verwendung von Verweisen (Hyperlinks)
  - Einfache Formulare
- Weiterentwicklung:
  - „Dokument“ verallgemeinert als Anforderung einer Ressource, die auch dynamisch erstellt, bearbeitet oder ausgeführt werden kann $\Rightarrow$ Web-Applikationen, Web-Services
  - Größere Dynamik Benutzerschnittstelle
    - Veränderung der clientseitigen Datenstrukturen
    - Erweiterte Funktionalität Interaktionselemente
    - Integration von Medien

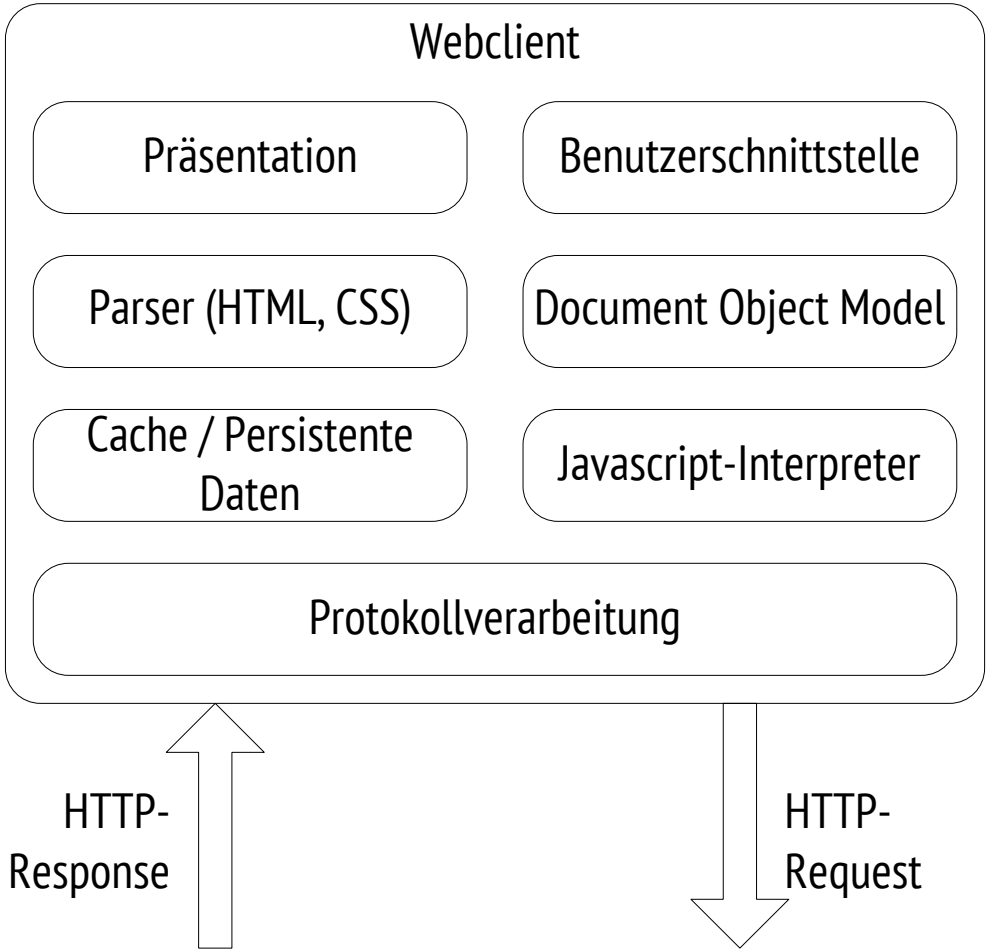
Webclient (1)

- Auch als „Webbrowser“ bezeichnet
  - to browse : blättern, durchsuchen
  - Bezeichnung bezieht sich auf die ursprüngliche Hauptaufgabe (Dokumente anfordern, darstellen)
- Aufgaben
  - Anforderungen (Requests) per HTTP erzeugen
  - Rückmeldungen (Responses) verarbeiten
    - Dokumentbeschreibung interpretieren
    - Dokument als Datenstruktur intern aufbauen
    - Dokument präsentieren („render“)
    - Benutzerinteraktionen verwalten
  - Programme ausführen (z.B. javascript mit Hilfe eines Interpreters)

Webclient (2)

- Weitere Eigenschaften
  - Zwischenspeicherung (Cache) von Inhalten und Ressourcen (Medien etc.)
  - Speicherung persistenter Daten auf dem Client-System
    - „Cookies“ / Weitere Mechanismen
    - Einstellungen etc.
  - Kein transparenter Zugriff auf die Fähigkeiten des Client-System !
    - Insbesondere keine transparente Nutzung der Ressourcen des Client (Dateisystem, Prozesse, Hauptspeicher, Dienste)

Webclient (3)


Webclient
Präsentation
Benutzerschnittstelle
Parser (HTML, CSS)
Document Object Model
Cache / Persistente Daten
Javascript-Interpreter
Protokollverarbeitung
HTTP-Response
HTTP-Request

Webserver

- Aufgaben
  - Protokollbearbeitung
    - Anforderungen (Requests) der Webclients bearbeiten
    - Gelieferte „Namen“ = Adresse des Dokuments auf die physikalische Ablage abbilden
    - Rückmeldungen (Responses) erzeugen
      - Angeforderte Dokumente und Medien übertragen
      - Oder Fehlermeldungen liefern
  - Anforderungen und Fehler protokollieren
  - Erweiterte Request-Bearbeitung:
    - Erzeugung Response durch Programmausführung

HTTP (1)

- HTTP = Hypertext Transfer Protocol
- Protokoll : Vereinbarung zwischen zwei Partnern über den Austausch von Leistungen und Daten
- Zustandslos :
  - Request / Response sind in sich abgeschlossene Transaktion
  - Unabhängig von vorangegangenen Transaktionen
  - Ggf. Zeitüberwachung (Timeout) des Response

HTTP (2)

- Klartext, d.h. keinerlei Binärform oder Verschlüsselung
  - Variante HTTPS : ‚sichere' Übertragung durch Verschlüsselung des gesamten Datenaustauschs
- Varianten
  - Aktuell : Version 1.1
    - Dauerhafte Verbindungen möglich (→ Netzwerkmanagement)
    - Eindeutige Kennungen für Dokumente (→ Caching)
  - Version 1.0
    - Einführung des Befehls POST
    - Medientypen verwenden
  - Version 0.9
    - Nicht mehr aktuell, nur noch historisch interessant

HTTP (3)

- Requests:
  - Message-Header
    - Request-Line
      - Methode, z.B. GET oder POST
      - Angefordertes Dokument / angeforderte Ressource
      - HTTP-Version
    - Verschiedene Informationen / Einstellungen
  - Message-Body
    - Z.B. Daten, die zum Webserver übertragen werden sollen
- Wichtigste Methoden:
  - GET zur Anforderung von Dokumenten
  - POST zur Anforderung von Leistungen mit Übertragung von (umfangreicheren) Daten vom Webclient zum Webserver

HTTP (4)

- Responses:
  - Message-Header
    - Status-Line
      - HTTP-Version
      - Statuscode
      - Erläuterung
  - Message-Body
    - Der eigentliche Inhalt der Antwort

HTTP (5)

- Datenübertragung mit GET und POST
  - Daten (z.B. aus Formularen) des Webclient werden stets als Key/Value-Paare übertragen
  - Bei GET
    - Daten werden an die Dokument-Adresse angehangen
    - Sind daher bei Webbrowsern i.d.R. in der Adresseingabe sichtbar
    - Sind auch in der Liste der besuchten Dokumente sichtbar
    - Datenmenge ist begrenzt; Sonderzeichen kodieren !
  - Bei POST
    - Getrennte Übertragung der Daten
    - Umfangreicher möglich

HTTP (6)

- Weitere Methoden
  - HEAD : nur Header senden
  - PUT : Dateien hochladen
  - DELETE : Ressource auf dem Server löschen
  - TRACE : Anfrage spiegeln
  - OPTIONS : Fähigkeiten des Server mitteilen
  - CONNECT : spezielle Verbindungen herstellen

Identifikation von Ressourcen (1)

- Uniform Resource Identifier
  - Ziel : nach einheitlicher Syntax benannte Adressen, die im Anwendungskontext eindeutig sind
  - Anwendungskontext Web : URI muss „weltweit“ eindeutig sein
    - D.h. aber auch : ein im Anwendungskontext Web gültigen URI kann man als eindeutigen Bezeichner nutzen !
  - i.d.R. logische Adresse, d.h. Umsetzung in eine physikalische Adresse (z.B. welcher Webserver, welche Datei auf dem Webserver) durch verschiedene Mechanismen
- Unterbegriff URL (uniform resource locator) : Auffinden von Ressourcen beschreiben
- URN (uniform resource name) : inzwischen veralteter Begriff

Identifikation von Ressourcen (2)

- URI-Syntax allgemein :
  - <Schema>:<Schema-spezifischer Teil>
  - Typische Schemata : http ftp mailto file
- Schema http :

```
http://[<Benutzer>[:Passwort]@]
       Server[:<Port>]/
       [<Pfad>][?<Anfrage>][#<Fragment>]
```

- Beispiele :

```
http://www.hsnr.de/
http://lionel.kr.hs-niederrhein.de/~beims/testseite.html#web
```

Identifikation von Ressourcen (3)

- Schema http:
  - Authority : Benutzer … Port
  - Server : eindeutige Identifikation einer Domain oder IP-Adresse
  - Port : Kennzeichnung eines Dienstes, der auf dem Server ausgeführt wird
  - Pfad : der logische Zugriffspfad zur Ressource, muss vom Webserver interpretiert und umgesetzt werden
  - Anfrage : bei GET übertragene Werte
  - Fragment : Verweis auf einen *Anker* im Dokument (nur sinnvoll, wenn die Ressource ein HTML- oder XHTML-Dokument ist)

SGML, HTML, XML, XHTML (1)

- SGML : Standard Generalized Markup Language
  - Texte mit Auszeichnungen versehen, die Hinweise auf die logische Struktur und Bedeutung geben
  - i.d.R. ergeben sich Baumstrukturen
  - Strikte Trennung von Struktur (SGML-Dokument) und Präsentation
    - Bei einheitlicher Struktur kann Präsentation auf unterschiedlichen Medien in verschiedener Weise erfolgen
  - Definition der Dokument-Struktur
    - Sog. DTD = Document Type Definition
    - Damit Überprüfung der Gültigkeit eines Dokuments möglich (Validierung)

SGML, HTML, XML, XHTML (2)

- Markup : Elemente, i.d.R. mit öffnender und schließender Marke (*Tag*)

```
<title>Beispiel eines Titel</title>
```

- Typische Anwendungen von SGML:
  - HTML
  - Docbook
- Nachteil von SGML:
  - Kompliziert in der Anwendung

SGML, HTML, XML, XHTML (3)

- HTML
  - Ziel : Beschreibung von Hypertext-Dokumenten
  - Im Laufe der Entwicklung (Versionen 2, 3.2, 4, 5) erhebliche Erweiterungen und Änderungen
    - Dabei auch : Einführung von Elementen, die die Präsentation beeinflussen (sollen)
    - Beispiele :
      - Font-Element zur Spezifkation der Schriftart
      - Bold-Element zur Hervorhebung in einer bestimmten Weise
      - Italic-Element zur Hervorhebung in einer bestimmten Weise

SGML, HTML, XML, XHTML (4)

- HTML
  - In neueren Versionen korrekt definiert, in früheren Versionen dagegen nicht eindeutig oder korrekt
  - Webbrowser lassen daher viele Beschreibungsfehler zu !
- Prinzipieller Aufbau eines HTML-Dokuments

```
<html>
   <head>
     ...
   </head>
   <body>
     ...
   </body>
</html>
```

SGML, HTML, XML, XHTML (5)

- Head-Abschnitt
  - Nimmt Angaben zum Dokument auf
    - Z.B. Stichworte zur Charakterisierung des Inhalts, kann durch Suchmaschinen ausgewertet werden
  - Führt benötigte Ressourcen auf
- Body-Abschnitt
  - Enthält den eigentlichen Dokumentinhalt
  - Nur dieser Teil wird zur Präsentation ausgewertet

SGML, HTML, XML, XHTML (6)

- XML
  - Vereinfachung von SGML
  - Ursprünglich ebenfalls nur für das Electronic Publishing gedacht
  - Inzwischen Beschreibungsstandard für viele Arten von Daten
  - Prüfmöglichkeiten :
    - „wohl geformt“ : wurde die XML-Syntax eingehalten ?
    - „valide“ : wurden die XML-Elemente richtig eingesetzt ?

SGML, HTML, XML, XHTML (7)

- XML, Beispiel

```
<Vorlesung>
   <Titel>Web-Engineering</Titel>
   <Dozent>Beims</Dozent>
</Vorlesung>
```

  - „wohl geformt“ : prüfbar, weil nur die grundsätzliche XML-Syntax geprüft wird
  - „valide“ : kann nicht geprüft werden, weil eine Beschreibung der zugelassenen Elemente und der zulässigen Baumstruktur fehlt
    - Solche Beschreibungen können als DTD (wie bei SGML) oder mit XML-Schema angegeben werden

SGML, HTML, XML, XHTML (8)

- XHTML
  - Re-Definition von HTML mit den Mitteln von XML
  - Syntax eindeutiger und strenger
    - Schließende Marken zwingend erforderlich
    - Kleinschreibung
    - Attribute von Elementen in Anführungszeichen
  - Präsentationsspezifische Elemente entfernt
  - erweiterbar

HTML5 (1)

- Weiterentwicklung von HTML4
- Getrieben durch Herstellerkonsortium, Übernahme auch durch das W3C
  - Konkurrierende Spezifikationen W3C – Hersteller
  - Konkurrierende Vorgehensweisen bei der Weiterentwicklung
- Löst HTML4 und XHTML ab
- Wichtige Erweiterungen wie *canvas*, neue Elemente zur Seitengestaltung, endgültiger Verzicht auf einige problematische Elemente
- Verschiedene Erweiterungen wie WebStorage, WebWorkers, WebSockets

HTML5 (2)

Wichtige neue Elemente:

- Zur Strukturierung der Inhalte: `article`, `footer`, `header`, `main`, `section`, `figure` / `figcaption`
- Zur Verbesserung der Benutzbarkeit: `aside`, `menu`, `nav`, weitere Sub-Typen bei `input`
- Zur Aufnahme spezieller Inhalte: `canvas`, `audio`, `video`, `svg`, `output`

CSS

- Cascading Style Sheets
  - Ziel : auf einfachem Weg angeben, wie die Elemente in einem (X)HTML-Dokument präsentiert werden
  - Besteht aus Regeln
    - Die einen *Selektor* aufweisen, mit dem festgelegt wird, auf welche Elemente die Regel angewendet wird
    - Die keine, eine oder viele Key-Value-Paare enthalten, die die einzelnen Darstellungseigenschaften festlegen
  - Beispiel : alle Absätze (Paragraphen : p) mit rotem Hintergrund versehen

```
p { background-color: red; }
```

CSS3

- Weiterentwicklung des CSS-Standards
- In Zusammenhang mit HTML5
- Besteht aus vielen Einzelspezifikationen, mit unterschiedlichem Status (Gültigkeit / Verabschiedung als Standard)
- Relevant:
  - wesentliche Erweiterungen bei den Selektoren
  - Erweiterte Möglichkeiten bei Hintergründen
  - Web-Fonts

Web-Engineering

2 / Präsentation mit CSS (Überblick)

Hochschule Niederrhein
University of Applied Sciences
Elektrotechnik und Informatik
Faculty of Engineering and Computer Science

HTML-Dokumente präsentieren

- Normalerweise sollen HTML-Dokumente auf einem Bildschirm dargestellt werden
  - Verfahren notwendig, den Baum der HTML-Elemente in eine Anordnung auf dem Bildschirm zu transformieren
  - Strukturierung des Dokuments wirkt sich auch grundsätzlich auf Präsentation aus
- Konzept "Textfluss"
  - Wichtige Dokumentteile blockweise untereinander anordnen, entspricht den Gewohnheiten der Gestaltung von gedruckten Dokumenten

Textfluss (1)

- Block level Elemente :
  - Erzeugen Blöcke entsprechend dem Inhalt des Dokuments
  - Blöcke werden (normalerweise) untereinander angeordnet
  - Ohne weitere Angaben verschiedene Annahmen über Abstände der Blöcke, Platz um die Blöcke herum
  - Anpassung an Platz für body-Element, falls Webbrowser-Fenster in der Größe verändert wird (falls keine anderen Angaben vorliegen)

Textfluss (2)

- Inline Elemente :
  - Erzeugen keine Blöcke
  - Nehmen innerhalb eines Blocks soviel Platz ein wie nötig
  - Werden (normalerweise) innerhalb eines Blocks von links nach rechts nebeneinander angeordnet
  - Werden ggf. 'zeilenweise' umgebrochen

Textfluss (3)

- Abweichende Vorgehensweisen bei
  - Listen
    - Listenelemente beginnen neue 'Zeile'
    - Einrückung
  - Tabellen
    - Als Ganzes ein Block Level Element
    - Tabellenreihen untereinander
    - Nebeneinanderliegende Zellen

Probleme bei der Präsentation (1)

- Unterschiedliche Medien
- Bildschirm
  - unterschiedliche physikalische Eigenschaften einzelner Geräte, z.B.
    - Wide Screen
    - Bildschirm des durchschnittlichen PC-Arbeitsplatzes
    - Notebook / andere mobile Endgeräte
  - Generell geringe Auflösung
  - Verfügbarkeit von Schriftarten, Korrektheit von Farbwiedergaben nicht gesichert
  - Variable Seitengröße, vertikales Scrollen sinnvoll einsetzbar

Probleme bei der Präsentation (2)

- Print
  - Hohe Auflösung, gesicherte Farbdarstellungen
  - Seitenkonzept
    - Feststehende Seitengrößen
    - Seitenumbruch
  - Statisch

- Ziele daher :
  - *Struktur und Präsentation trennen !*
  - Anpassung / Änderung Präsentation je nach Kontext / Anforderungen

Präsentationsregeln (1)

- Für HTML-Elemente Darstellungseigenschaften beschreiben
- CSS : Cascading Style Sheets
  - Für alle SGML/XML-basierte Sprachen geeignet
  - Menge von Regeln
  - Jeder Regel bestehend aus :
    - Selektor : auf welche Elemente anwenden ?
    - Darstellungseigenschaften in {}
      - Key / Value-Paare

Präsentationsregeln (2)

- CSS : Normierung durch W3C
- 3 Level definiert, Webbrowser-Unterstützung unterschiedlich (Level 1 i.d.R. ganz, Level 2 viele Regeln, Level 3: keine geschlossene Spezifikation, z.Zt. CSS3-Selektoren, WebFonts, Hintergründe)
- CSS1 (Level 1) : Font-, Text-, Box-, Color- und Klassifizierungseigenschaften
- CSS 2.1 (Level 2) : Box-Model, Visual Formatting Model, Tables, Erweiterungen Level 1

Wirksamkeit von Regeln : Kaskade

- Mehr als eine Regel kann die Darstellung von HTML-Elementen bestimmen
- Bei Regel-Konflikten ist entscheidend, wo die Regel definiert wird :
  - Standard-Stylesheets des Webbrowsers / des Benutzers
  - Per `@import` eingefügte Stylesheets
  - Stylesheet-Definition im HTML-head-Bereich
  - Regeln, die als Attribut beim HTML-Element angegeben werden
  - Priorisierung durch die Angabe `important` (beachte Trennzeichen "!")

CSS für bestimmte Medien

- Wirksamkeit der CSS-Stilregeln kann auf die Ausgabe über definierte Medien beschränkt werden
- Angabe bei eingebetteten Styles mit Attribut media
- Angabe bei als Zusammenfassung von Stilregeln mit @media <typ> { … Regeln … }
- Beispiele:
  - Keine Angabe oder Attribut media="screen" bzw. @media screen
  - Attribut media="print" bzw. @media print
- Weitere Typen z.B. projection, braille, handheld

Web-Engineering

Web-Applikationen / Architekturvarianten

Hochschule Niederrhein
University of Applied Sciences
Elektrotechnik und Informatik
Faculty of Engineering and Computer Science

Web-Applikationen: Architektur (1)

Aufteilung Verantwortlichkeiten

- Server:
  - Datenhaltung
  - Applikationslogik
  - Auslieferung User Interface (Applikationslogik)
- Client:
  - User Interface (Applikationslogik)

Web-Applikationen

- Verantwortlichkeiten:
  - User Interface
    - Web-Client, i.d.R. HTML5 / CSS / javascript
  - Applikationslogik
    - Kann teilweise oder ganz auf dem Web-Client vorhanden sein
    - Kann teilweise oder ganz auf dem Web-Server vorhanden sein
  - Datenhaltung
    - Auf dem Web-Server vorhanden
    - Oft mit Hilfe eines Datenbankmanagementsystems (DBMS)
    - Zur Zwischenspeicherung: eventuell auf dem Web-Client

Web-Applikationen: Aufteilungsvarianten (1)

- Variante 1 – 3: Schwerpunkt serverseitige Verarbeitung

	Client	Server
1	HTML5: Links, Formulare	Statische Ressourcen, direkte Aufbereitung
2	HTML5: Links, Formulare	Template Engine, Daten aufbereiten
3	HTML5: Links, Formulare Javascript (UI-Ereignisse)	Template Engine, Daten aufbereiten

Web-Applikationen: Aufteilungsvarianten (2)

- Variante 4: Verarbeitung aufgeteilt

	Client	Server
4	Single-Page, Hintergrund-Transfer HTML5, javascript (UI-Ereignisse, AJAX)	Template Engine, Daten aufbereiten

Web-Applikationen: Aufteilungsvarianten (3)

- Varianten 5 – 7: Schwerpunkt clientseitige Verarbeitung, Server ist (nur noch) Datenlieferant

	Client	Server
5	Single-Page, Hintergrund-Transfer (Daten!), Markup direkt erzeugen HTML5, javascript (UI-Ereignisse, AJAX)	Schnittstelle Daten verwalten (Erzeugen, Lesen, Ändern, Löschen)

Web-Applikationen: Aufteilungsvarianten (4)

- Varianten 5 – 7: Schwerpunkt clientseitige Verarbeitung, Server ist (nur noch) Datenlieferant

	Client	Server
6	Single-Page, Hintergrund-Transfer (Daten!), Template-Engine HTML5, javascript (UI-Ereignisse, AJAX)	Schnittstelle Daten verwalten (Erzeugen, Lesen, Ändern, Löschen)

Web-Applikationen: Aufteilungsvarianten (5)

- Varianten 5 – 7: Schwerpunkt clientseitige Verarbeitung, Server ist (nur noch) Datenlieferant

	Client	Server
7	Single-Page, Hintergrund-Transfer (Daten/Javascript), Template-Engine, HTML5, javascript (UI-Ereignisse, AJAX)	Schnittstelle Daten verwalten (Erzeugen, Lesen, Ändern, Löschen) Script(s) bereitstellen

Web-Applikationen: Strukturen bilden (1)

Serverseitig:

- Requests bearbeiten (→ Aufgaben verteilen)
- Daten verwalten (→ DB API)
  - Auch: „Geschäftsprozesse“ = Regeln beachten
- Responses erzeugen (→ Darstellung transformieren)

Clientseitig:

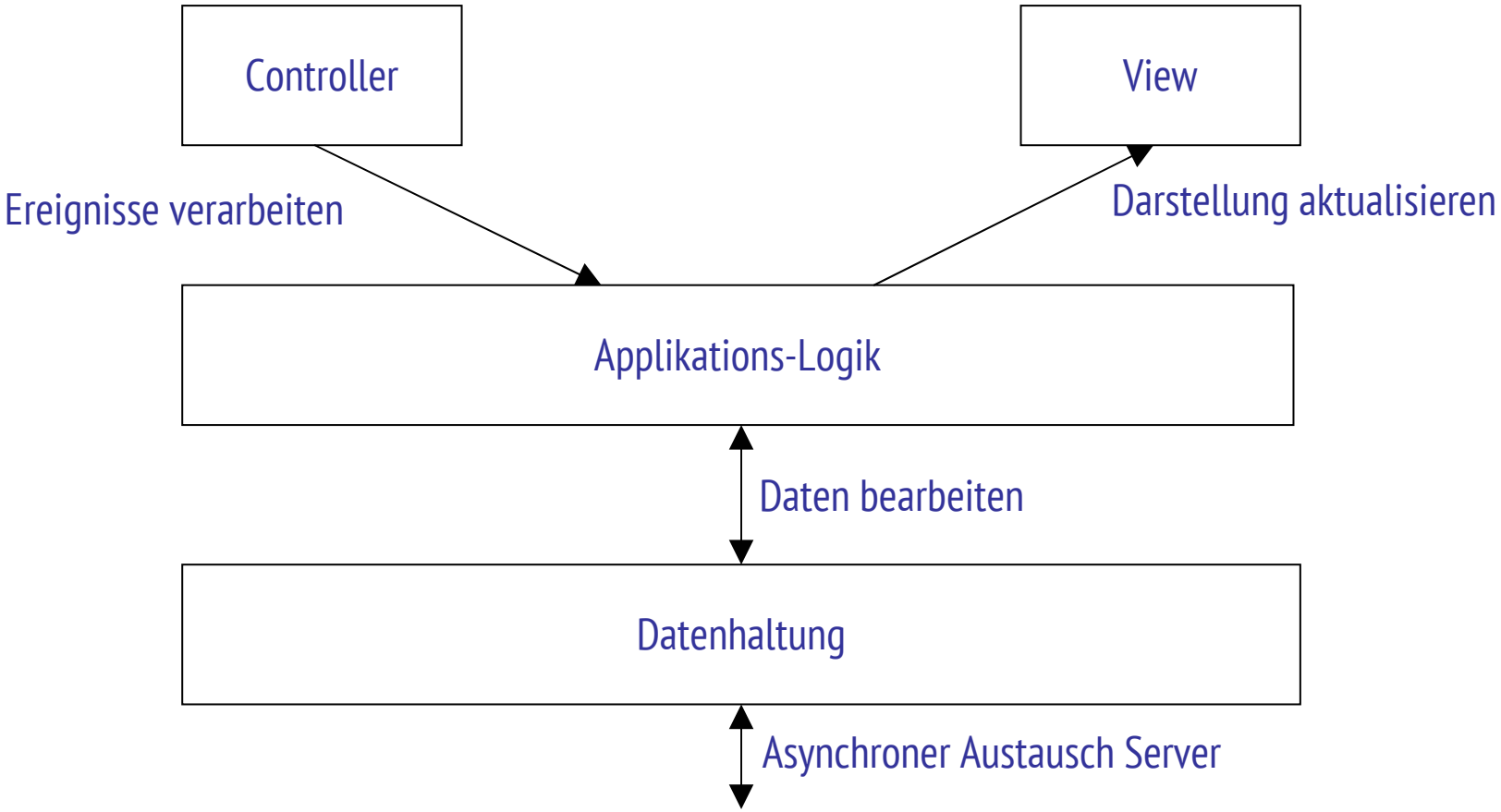
- Asynchrone Abläufe
  - UI-Ereignisse
  - Netzwerk-Ereignisse (HTTP-Response)

Web-Applikationen: Strukturen bilden (2)

Clientseitig:

- Eventservice: Ereignisse normieren
- Ideen MVC-Ansatz nutzen:
  - Controller:
    - (externe) Ereignisse entgegennehmen
    - verteilen
  - Model:
    - Daten und Regeln verwalten
  - View:
    - Sicht erzeugen

Web-Applikationen: Strukturen bilden (3)



Web-Applikationen: Strukturen bilden (4)

Realisierungsmöglichkeit:

- Eventservice verwenden
  - Basis: Entwurfsmuster „Publish-Subscribe“
  - In javascript: nur 1 Thread
  - „timeout“-Funktion: Ausführung nach Abschluss des aktuellen Ausführungskontextes

Web-Applikationen: Strukturen bilden (5)

Clientseitig:

- Einfache javascript-Funktionen reichen i.d.R. nicht mehr aus
- Erweiterbarkeit
  - Prototype-Konzept
  - Implementierung von „Klassen“ und Vererbung
- Verwendung von JS-Bibliotheken / Frameworks

Web-Engineering

Web-Applikationen / REST

Hochschule Niederrhein
University of Applied Sciences
Elektrotechnik und Informatik
Faculty of Engineering and Computer Science

Literatur

- Tilkov: "REST und HTTP", 2. Auflage 2011, d.punkt-Verlag

Web-Applikationen: Architektur (1)

Aufteilung Verantwortlichkeiten, z.B.

- Server:
  - Datenhaltung
  - Applikationslogik
  - Auslieferung User Interface (Applikationslogik)
- Client:
  - User Interface (Applikationslogik)

Web-Applikationen: Architektur (2)

Zustände:

- Server:
  - Nächste erwartete Operationen gemäß Applikationslogik
  - Benutzerspezifisch / Sitzungsspezifisch
- Client:
  - User-Interface-Zustände (lokal)
  - Problem: Synchronisation mit Serverzuständen
- Probleme:
  - passt nicht zu zustandslosem HTTP
  - Enge Kopplung

Web-Applikationen: Architektur (3)

- Ziele:
  - Kopplung minimieren
  - Zustände einfacher verwalten
- Ideen:
  - Konzept „Ressource“ einführen
    - Definiert den Zustand
    - Wird ausgetauscht in unterschiedlichen Repräsentationen
  - Einheitliche Methoden => HTTP-Methoden
  - Hyperlinks ermöglichen Zustandswechsel

Web-Applikationen: Architektur (4)

Konsequenzen:

- Server verwaltet keine Sitzungszustände
- Server verwaltet stattdessen Ressource-Zustände
- Zustand wird mit Ressource zum Client übertragen
- Architektur von Web-Applikation:
  - Erzeugen, Ändern und Präsentieren von Ressourcen
  - Zustand: Ressource im Client bearbeiten
  - Zustandswechsel: per Hyperlink zu einer anderen Ressource

Web als Beispiel: Kollektion von Ressourcen

REST: Architekturansatz nach R. Fielding

Architekturansatz

- Beschrieben in der Dissertation von Roy Fielding

REST = "Representational State Transfer"

- Repräsentation
  - Versetzt Client in einen Zustand
- Zustand
  - Durch angeforderte Ressource, die als Repräsentation geliefert wird
- Übergang
  - In einen neuen Zustand durch Anforderung einer neuen Ressource (z.B. per Hyperlink)

REST: Eigenschaften (1)

- resource identification
  - Eindeutig identifizierbare Ressourcen
- uniform interface
  - Standardmethoden
  - Standardrepräsentationen
  - Ressourcen mittels der Repräsentationen bearbeiten
- self-descriptive messages
  - Standard-Nachrichten
  - Gemeinsames (?) Verständnis der Kommunikationspartner

REST: Eigenschaften (2)

- HATEOAS:
  - „Hypermedia as the engine of application state“
  - Weder Client noch Server verwalten einen Session-Zustand
  - Alle zustandsrelevanten Daten in den HTTP-Nachrichten
    - URI
    - Angaben in Message-Header und Message-Body
    - Enthaltene Hypermedia-Angaben (Hyperlinks)
- Fazit:
  - Standard-Methoden, -Protokolle, -Präsentationen nutzen
  - Zustandslosigkeit, lose Kopplung
  - Identifizierbare Ressourcen

REST: Ressourcen

- Alles, was durch eine URI **eindeutig** identifizierbar ist
  - Allein durch Protokoll, Server, Pfad
  - Statische Webseiten
  - Kollektion von Ressourcen
  - Nicht nur (Daten-)Objekte im klassischen Sinne, auch z.B.:
    - Zwischenergebnisse (z.B. bei Bestellvorgang)
    - Endergebnisse (z.B. Rechnung) mit Verweisen auf andere Ressourcen
    - Transaktionen (Zusammenfassung von Operationen)
    - Applikationsübergreifende Methoden
- Repräsentationen
  - Standardformen verwenden
  - „Content Negotiation“: Abstimmung zwischen Client und Server über Form
  - Bearbeiten, d.h. keine „direkte“ Bearbeitung einer Ressource

REST: uniform interface (1)

Generische Methoden: "uniform interface"

- Auf Ressourcen anwenden
- Mit HTTP-Befehlen/verbs implementieren, z.B.
  - GET: auf Ressource zugreifen (select)
  - POST: neue Ressource erzeugen (insert) (spezielle Operationen)
  - PUT: Ressource ändern (update)
  - DELETE: Ressource löschen (delete)
  - HEADER: Metainformationen zur Ressource
  - OPTIONS: Metainformationen zum Server

  (es gibt auch andere Einteilungen bei PUT/POST!)

REST : uniform interface (2)

- Anfragen (Requests) verwenden dann keine Parameter mehr
  - Statt : GET /person/?id=4711&action=save&Name= …
  - Jetzt: PUT /person/4711
    - Daten im Request-Body
  - Statt: GET /person/delete/?id=4711
  - Jetzt: DELETE /person/4711

    (Eine numerische Objekt-Id ist nicht zwingend erforderlich)

REST : uniform interface (3)

- HATEOAS: *(siehe: REST Eigenschaften (2))*
  - „Hypermedia as the engine of application state“
- Hyperlinks:
  - Globales Namensschema => alle Ressourcen verknüpfbar
  - Steuerung Zustand:
    - Server teilt dem Client über Hyperlinks mit, welche Aktionen er als nächstes ausführen kann
    - Applikation erhält neuen Zustand, indem der Client einem Hyperlink folgt

Implementierungsmöglichkeiten

- Bei CherryPy:
  - "MethodDispatcher" verwenden
    - Klassen erhalten das Attribut "exposed"
    - Methoden GET, POST, PUT, DELETE (ggf. weitere)
      - Alle Methoden in Großbuchstaben werden veröffentlich und auf HTTP-Verbs abgebildet
- Bei anderen Ansätzen
  - Aus Umgebungsvariablen ermitteln, welches HTTP-Verb verwendet wird

Web-Engineering

Semantic Web

Hochschule Niederrhein
University of Applied Sciences
Elektrotechnik und Informatik
Faculty of Engineering and Computer Science

World Wide Web

- Netzwerk
  - Ressourcen unterschiedlicher Art
  - Hypertext: Verweise auf Ressourcen, insbesondere Textdokumente
- Implizite Bedeutung der Ressourcen und ihrer Beziehungen
  - Unterschiedliche Interpretationen möglich
  - Nur durch den Benutzer interpretierbar

Wie kommt man an Informationen ?

- Suchmaschinen:
  - Information-Retrieval: Merkmale extrahieren aus unstrukturierten Daten
  - Kontext der durchsuchten Daten kann nur schwer einbezogen werden
  - Interpretation des Suchergebnisses dadurch eingeschränkt
  - Vereinfacht gesehen: Übertragung des Prinzip der Textmustersuche in Dateisystemen auf das WWW

Metadaten: Anreicherung mit Informationen

- Verbesserung der Suche:
  - Unstrukturierte Daten mit beschreibenden Daten (= Metadaten) anreichern
  - Beispiel Datenbanken:
    - SELECT * FROM personen WHERE name LIKE 'B%'
    - Auswertung des DB-Schema = Metadaten möglich
- Übertragung auf das WWW:
  - WWW erweitern zu einem *semantischen* Netz

Semantisches Netz (1)

- Formales Modell für Begriffe und deren Beziehungen (Relationen)
- i.d.R. Graphen:
  - Knoten als Begriffe
  - Kanten als Beziehungen
- Einfaches Beispiel:
  - Mindmaps: zentrale Begriffe, hierarchische Beziehungen

Semantisches Netz (2)

- Regeln
  - Zur Beschreibung
    - Der Bildung von Begriffen und Relationen
    - Der Gültigkeit von Begriffen und Relationen
    - Von Einschränkungen
- Mit Begriffen, Relationen und Regeln als Metadaten kann die Bedeutung des semantischen Netzes "WWW" angegeben werden

Semantic Web (1)

- Initiative von Tim Berners-Lee
- Ziele:
  - Verbesserung der Recherche durch Einbeziehung inhaltlicher Aspekte = Bedeutung der Inhalte
    - auch für den Menschen
    - Vor allem für die automatisierte Verarbeitung durch DV-Systeme
  - Automatische Ableitung neuen Wissens
- WWW mit Metadaten anreichern in Form von *Ontologien*

Semantic Web (2)

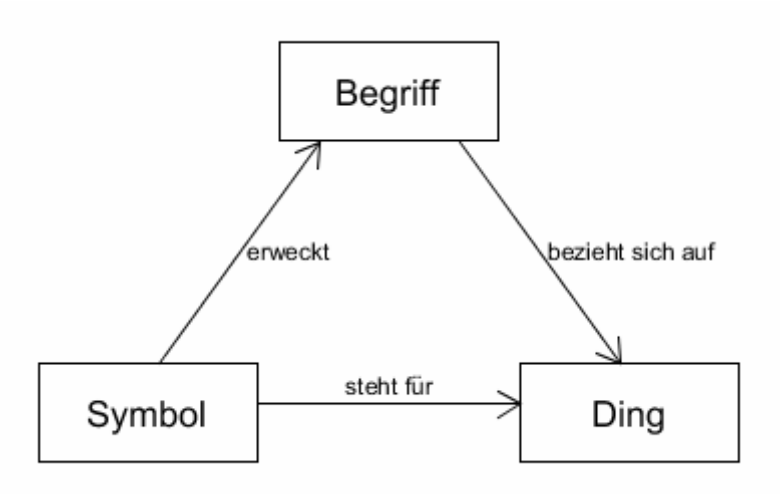
Zitat Tim Berners-Lee (2002):

"The WWW contains documents intended **for human consumption**, and those intended **for machine processing**. The Semantic Web will enhance the latter. The Semantic Web will not understand human language ... The Semantic Web ist about machine languages: well-defined, mathematical, boring, but processible. **Data, not poetry**."

Exkurs 1: Semiotik (1)

- Allgemeine Lehre von Zeichen, Zeichensystemen und Zeichenprozessen
- Bereiche (Interpretationen/Abbildungen):
  - Syntax: Zeichen ↔ Zeichen
  - Semantik: Zeichen ↔ Bedeutung
  - Pragmatik: Zeichen ↔ Benutzer und Situation

Exkurs 1: Semiotik (2)



- Semiotisches Dreieck:
  - Gegenstand (Ding) (aber auch: Sachverhalte, Ereignisse…)
  - Begriff
  - Benennung (Symbol)

Exkurs 2: Ontologie (1)

- Beschreibung von
  - Begriffen
  - Relationen
  - Regeln

  für einen bestimmten Gegenstandsbereich
- Begriffe treten auf als
  - Typen
  - Instanzen
- Regeln beziehen sich auf
  - Bildung
  - Gültigkeit
  - Einschränkungen

Exkurs 2: Ontologie (2)

Abgrenzung zu anderen Konzepten:

- Thesaurus:
  - Systematisch geordnete Sammlung von Begriffen, die in thematischer Ordnung zueinander stehen
  - Deskriptoren: beschreibende Attribute
  - Synonyme: gleichbedeutende Worte
  - Homonyme: Worte, die verschiedene Begriffe bezeichnen
  - Ober-/Unterbegriffe

Exkurs 2: Ontologie (3)

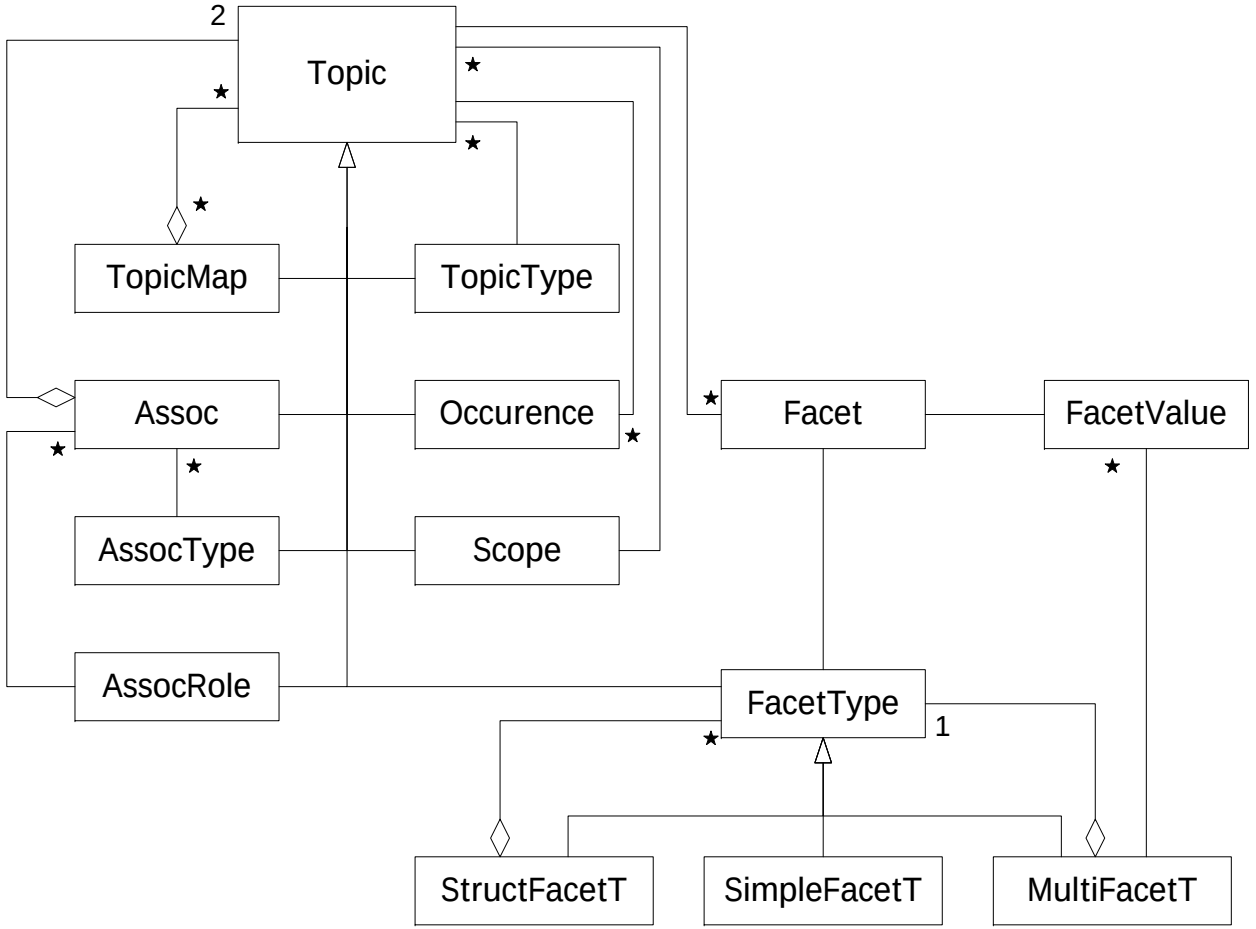
(Forts.) Abgrenzung zu anderen Konzepten:

- Taxonomie:
  - Klassifizierungen, systematische Verschlagwortung
- Folksonomy: (neues Kunstwort aus folk und taxonomy)
  - Taxonomie durch Laien, i.d.R. mit "Sozialer Software"
- Glossar:
  - Zusammenstellung von Begriffen

Topic Maps (1): Konzept

- Semantisches Netz zur Repräsentation von Wissen
- Geeignet zur Repräsentation von Ontologien
- TAO:
  - Topics: Person, log. Einheit, Konzept, ...
  - Associations: Beziehungen zwischen Topics
  - Occurrences: Erscheinungsformen von Topics
- IFS:
  - Identity: eindeutige Adressierbarkeit eines Topic
  - Facets: Eigenschaften
  - Scopes: Kontext, in dem T/A/O interpretiert werden

Topic Maps (2): Metamodell



Standards zur Repräsentation (1)

- Topic-Maps:
  - ISO 13250
  - XTM: XML Topic Maps, Redefinition ISO 13250 mit XML
- RDF (W3C): Resource Description Facility
  - XML basiert, d.h.
    - Strukturierung mit einer einheitlichen Syntax
    - Eindeutige Definition von XML-Vokabularen mit XML-Namespaces
    - Syntax von XML-Vokabularen mit XML-Schema detailliert beschreibbar

Standards zur Repräsentation (2)

- (Forts.) RDF:
  - Beschreibung einzelner Ressourcen:
    - Sammlung von Aussagen in der Form *Subjekt Prädikat Objekt*
    - Objekte können wiederum Ressourcen sein, die weiter beschrieben werden
  - Es ergibt sich ein gerichteter Graph aus Ressourcen und ihren Eigenschaften
- RDF-Schema (W3C):
  - In RDF Klassen von Objekten bilden
  - Eigenschaften in Werte- / Gültigkeitsbereiche zu ordnen

Standards zur Repräsentation (3)

- Web Ontology Language (OWL) (W3C):
  - Erweitert RDF-Schema um Beschreibungsmöglichkeiten
    - Zur Einschränkung von Eigenschaften (z.B. Kardinalitäten)
    - Angabe gemeinsamer Eigenschafter unterschiedlicher RDF-Klassen (Schnittmengen)

- Werkzeuge, Anwendungen zu RDF, RDF-Schema und OWL:
  - z.B. *dbpedia.org* : Wikipedia aufbereitet
  - librdf.org: Redland RDF Libraries (C), mit Bindungen zu Perl, PHP, Python, Ruby

In der Praxis: Dublin-Core (1)

(Quelle: dublincore.org, wiki.dublincore.org)

- "Dublin Core Metadata Initiative"
  - Dublin: Konferenzort in Ohio (!) im Jahr 1995
  - Dokumente mit recherchierbaren Metadaten anreichern
- "set of fifteen generic elements for describing resources"
  - Creator, Contributor, Publisher, Title, Date, Language, Format, Subject, Description, Identifier, Relation, Source, Type, Coverage, and Rights
- Weitere Elemente zur optionalen Verfeinerung der Beschreibung

In der Praxis: Dublin-Core (2)

(Quelle: de.wikipedia.org)

```
<head profile="http://dublincore.org/documents/dcq-html/">
  <title>Dublin Core</title>
  <link rel="schema.DC" href="http://purl.org/dc/elements/1.1/" />
  <link rel="schema.DCTERMS" href="http://purl.org/dc/terms/" />
  <meta name="DC.format"              scheme="DCTERMS.IMT" content="text/html" />
  <meta name="DC.type"                scheme="DCTERMS.DCMIType" content="Text" />
  <meta name="DC.publisher"           content="Jimmy Wales" />
  <meta name="DC.subject"             content="Dublin Core Metadaten-Elemente, Anwendungen" />
  <meta name="DC.creator"             content="Björn G. Kulms" />
  <meta name="DCTERMS.license"        scheme="DCTERMS.URI" content="http://www.gnu.org/copyleft/fdl.html" />
  <meta name="DCTERMS.rightsHolder" content="Wikimedia Foundation Inc." />
  <meta name="DCTERMS.modified"     scheme="DCTERMS.W3CDTF" content="2006-03-08" />
</head>
```

In der Praxis: LinkedData (1)

(Quellen: linkeddata.org und linkeddatabook.com)

Prinzipien nach T. Berners-Lee:

- Use URIs as names for things.
- Use HTTP URIs, so that people can look up those names.
- When someone looks up a URI, provide useful information, using the standards.
- Include links to other URIs, so that they can discover more things.

In der Praxis: LinkedData (2)

- Identifizierung per URI:
  - für alle Zusammenhänge, auch Begriffe, Erscheinungsformen, Relationen
  - Nicht nur auf Web-Dokumente anwenden
- Zugriff per HTTP
- Bereitstellung der Informationen in standardisierter Form
  - HTML
  - RDF
- Beziehungen mit Hilfe von Hyperlinks herstellen
  - Nicht nur Web-Dokumente (s.o.)
  - Mit Bedeutung anreichern (welche Beziehung?) statt untypisierter Hyperlinks wie bei Web-Dokumenten

In der Praxis: LinkedData (3)

Bereitstellung der Informationen:

- Lesbar für Benutzer
- Maschinell verarbeitbar
- Unterschiedliche Auslieferungen
  - "content negotation" ("Verhandlung über den Inhalt"): im http-Header wird angegeben, welches Format erwartet wird
  - Für Benutzer z.B. HTML
  - Für Software z.B. RDF

Web-Engineering

Sicherheit

Hochschule Niederrhein
University of Applied Sciences
Elektrotechnik und Informatik
Faculty of Engineering and Computer Science

Vorbemerkung

- IT-Sicherheit:
  - Umfassendere Darstellung in einer eigenen Veranstaltung
  - Hier: spezielle Aspekte bei Web-Anwendungen

- OWASP (Open Web Application Security Project)
  - www.owasp.org
  - Top 10 Liste der Sicherheitsrisiken zu einzelnen Jahren

Sicherheitsprobleme (1)

- Aspekte des Thema "Sicherheit":
  - Schutz vor unbefugter Verwendung von Software
    - Autorisierung : wer darf was tun ?
  - Schutz vertrauenswürdiger Daten
    - Authenzität:
      - Originalität (ist es das ursprüngliche Dokument ?)
      - Rechtsgültigkeit
    - Personenbezogene Daten
  - Schutz vor Schädigung:
    - Ruf / Vertrauen zu einem Informationsanbieter
    - Gezielte Fehlinformationen
    - Beschädigung / Zerstörung von Informationsangeboten und Diensten

Sicherheitsprobleme (2)

- Speziell bei Web-Anwendungen:
  - Zerlegung in 2 unabhängige, einander nicht bekannte Systeme (Client-System, Server-System)
    - Gegenseitige Vertrauenswürdigkeit ist problematisch
  - Technologien sind nicht per se mit Schutzmechanismen ausgestattet:
    - Klartext-Übertragungen (HTTP)
    - Clientseitige dynamische Auslieferung / Installation von Software
    - Clientseitig Transparenz der verwendeten Datenstrukturen

Angriffsmöglichkeiten (Bsp.)

- Clientseitige Angriffe:
  - XSS: Cross-Site-Scripting
    - Injizieren von Schad-Code auf dem Client-System
  - Cross-Site Request Forgery:
    - Fälschen der Requests
  - Ausnutzen von Fehlern in den Webbrowsern
- Übertragungsweg:
  - Man-in-the-Middle
    - Abhören und Verfälschen des Nachrichtenaustauschs
- Serverseitige Angriffe
  - Injizieren von Schad-Code auf dem Server-System
  - Ausnutzen von Fehlern in den Web-Servern, Applikations- und Datenbankserver
  - Gezielte Überlastung (Denial of Service-, Distributed Denial of Service-Attack)

Merke

- Misstraue allen Nachrichten ! Seien es
  - Nachrichten vom Server
  - Nachrichten vom Client
  - Ressourcen aus verschiedenen Quellen (s.u.)
- Daraus folgt:
  - Nachrichteninhalte müssen beim Client und beim Server überprüft werden
    - Syntaktisch (Form)
    - Semantisch (Inhalt)

Clientseitige Angriffe

- Nutzen die Programmiermöglichkeiten beim Webbrowser aus
  - Zugriff / Veränderung des DOM
  - Dynamisches Nachladen / Erzeugen von Programmcode
  - Dynamische Erzeugung von Inhalten
    - Direkt serverseitig
    - Aufgrund gelieferter Daten (per AJAX)
- Auch bei statischen (X)HTML-Seiten, wenn auf Client z.B. javascript ausgeführt werden darf

XSS: Cross-Site-Scripting (1)

- "Same-Origin-Policy":
  - Wird eine Quelle durch den Aufruf einer Webseite als vertrauenswürdig angesehen, dann sind auch alle weiteren Informationen dieser Quelle vertrauenswürdig
  - Alle Daten einer Website (= Web-Standort) werden gemeinsam eingeschätzt
    - Anderen Quellen wird misstraut, d.h. die gelieferten Informationen werden nicht berücksichtigt
  - Also nur Unterscheidung in Pfad und Dateiname zulässig
- Beispiel: Website http://www.mustersoft.com
  - http://www.mustersoft.com/main.html $\Rightarrow$ vertrauenswürdig
  - http://www.anders.com/main.html $\Rightarrow$ nicht vertrauenswürdig

XSS: Cross-Site-Scripting (2)

- "Same-Origin-Policy": **Problem**
  - Wird nur bei Seiten-Requests und Zugriffe auf DOM-Bäume unterschiedlichen Ursprungs berücksichtigt
  - Wird **nicht** bei Requests berücksichtigt, die durch `src`-Attribute ausgelöst werden, z.B.

    Image-Element `<img src="http://fremdedomain.com/Bild">`

    Iframe-Element `<iframe src="http://fremdedomain.de/x.htm">`

    Script-Element `<script src="http://fremdedomain.de/x.js">`

XSS: Cross-Site-Scripting (3)

- Injection von Schad-Code:
  - Nutzt die Möglichkeit aus, z.B. Code-Abschnitte (<script>, i.d.R. javascript) an beliebigen Stellen einzufügen
  - Einfache Variante:
    - Direktes Speichern externer Eingaben
    - Erzeugung einer Ausgabe aufgrund dieser Eingaben
    - Mögliche Abhilfe: Meta-Zeichen so umwandeln, dass ihre besondere Bedeutung verloren geht
      - Spezielle Module der Web-Frameworks verwenden
      - Mit regulären Ausdrücken prüfen / bearbeiten

XSS: Cross-Site-Scripting (4)

- Injection von Schad-Code (Forts.):
  - Seiten mit präparierten Links
    - Direkt mit angehängten Script-Teilen
    - Sind ggf. in Statuszeilen sichtbar
  - Anstelle einfacher Links: präparierte src-Angaben, z.B. für Script-Bereiche, die generell unsichtbar bleiben
  - Veränderung von GET-Anfragen, wenn URLs von dort direkt übernommen werden
    - GET-Parameter sind i.d.R. sichtbar in URL-Eingabefeldern

XSS: Cross-Site-Scripting (5)

- Mögliche Schäden:
  - Auslesen von Cookies/clientseitig gespeicherten Daten, Übermittlung an Dritte
    - Insbesondere Session-ID
  - Auslesen von clienseitig intern gespeicherten Daten (z.B. Formulardaten), Übermittlung an Dritte
    - Insbesondere Passwörter (auch bei verschlüsselten Verbindungen!)

Man-in-the-Middle

- Zwischen die Kommunikationspartner Client und Server schaltet sich ein *Angreifer*:
  - Beobachtet den ursprünglichen Datenverkehr
  - Simuliert dann zu beiden Seiten hin den passenden Datenverkehr
    - Um sensible Informationen zu erlangen
    - Um die Kommunikation zu verändern
    - Ggf. sogar die Kommunikation weiter umzulenken
  - Abhilfe: Verschlüsselung der Kommunikation

Cross-Site Request Forgery (1)

- Requests fälschen:
  - Datenverkehr eines berechtigten Benutzers verwenden, um Schad-Code beim Webserver (und damit ggf. später bei den Web-Clients) zu injizieren
  - i.d.R. wird der Benutzer dazu veranlasst, einen manipulierten Link zu verwenden, der die gewünschte Wirkung hervorruft:
    - Z.B. in Cookies gespeicherte Sitzungsdaten benutzen
    - Z.B. mit XSS Seite des Benutzers passend verändern

Cross-Site Request Forgery (2)

- Mögliche Abhilfen:
  - Zunächst sicherstellen, dass kein XSS möglich ist
  - Bei Requests, die serverseitig Daten ändern sollen:
    - Ids auch auf anderem Weg übertragen
    - Einmalige Austausch-Ids verwenden, die dann serverseitig geprüft werden können

Serverseitige Angriffe

- Bekanntes Beispiel:
  - "SQL-Injection":
    - SQL-Anweisungen in Eingaben verstecken
    - Wenn die Eingabe direkt zum Aufbau einer SQL-Anweisung verwendet wird, kann der Schad-Code ausgeführt werden
    - Vertrauliche Inhalte können selektiert und zum Client übertragen werden
- Ausnutzen von Fehler in Interpretern / Laufzeitbibliotheken

Sicherheitstests

- Im Rahmen der Tests (siehe dort) auszuführen
- Überprüfen (Beispiele):
  - Welche Sicherheitsanforderungen bestehen?
  - Hardware / Sicherheitssoftware
  - Verschlüsselung
  - Passwortschutz
  - Benutzerführung
  - Session Handling
  - Protokollierung
  - Zugriffsrechte
- Penetrationstest durchführen: unerlaubter Systemzugriff möglich ?

Web-Engineering

Test von Web-Applikation / Web-Content

Hochschule Niederrhein
University of Applied Sciences
Elektrotechnik und Informatik
Faculty of Engineering and Computer Science

Literatur

- Franz, Klaus: Handbuch zum Testen von Web-Applikationen, Springer Xpert.press, 2007

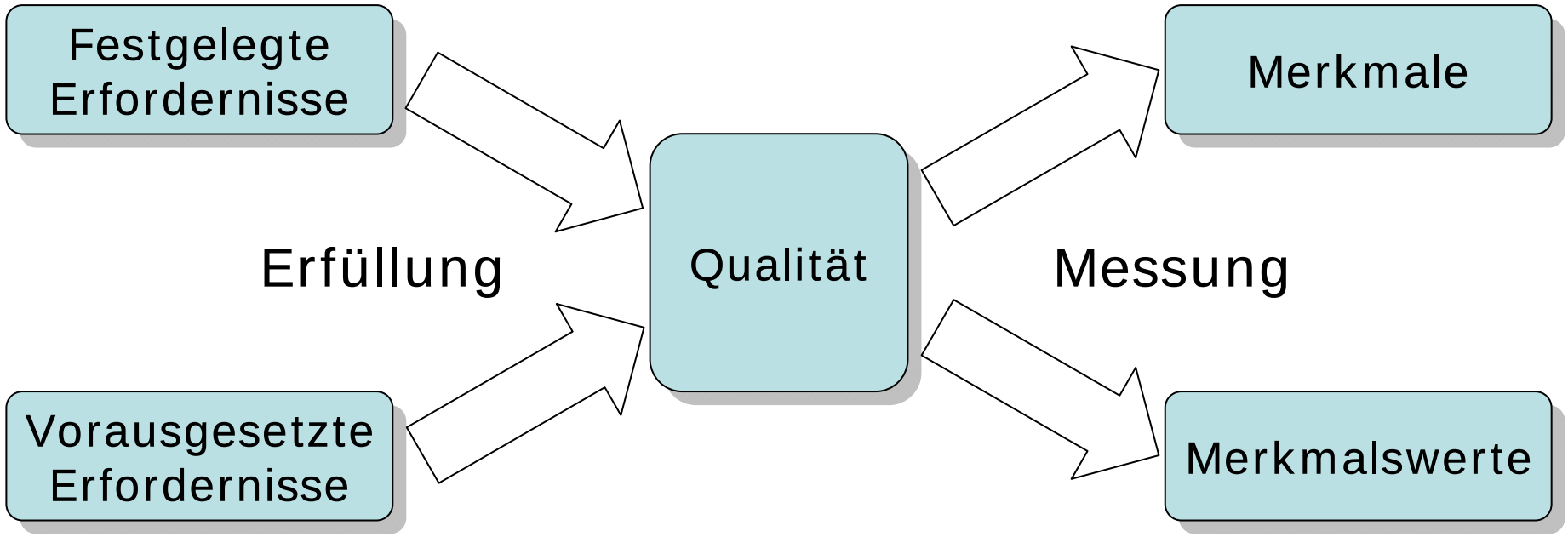
Der Unverbesserliche

Wer testet, ist feige !

Was ist ein Test (nicht) ?

- Nicht:
  - 'läuft schon ... irgendwie'
  - Einzelne Effekte beobachten
  - Mit einem Debugger exemplarisch einige Programmzeilen überprüfen
  - Herumprobieren
- Sondern:
  - **Messen**, in welchem Umfang vorgegebene Qualitätskriterien erfüllt werden
  - Somit: **Soll-Ist-Vergleich**

Was ist Qualität ?



Begriffe (1)

- Fehler: Nichterfüllung einer festgelegten Forderung
- Testobjekt: wird einem Test unterzogen
- Statischer Test: Prüfung Testobjekt ohne Rechnerunterstützung (insbesondere: ohne Ausführung)
- Dynamischer Test: Prüfung Testobjekt mit Rechnerunterstützung (insbesondere: mit Ausführung)

Begriffe (2)

- Testfall: Anweisung zur Durchführung eines Tests
  - Notwendige Vorbedingungen definieren / herstellen
  - Eingabedaten ("Testdaten")
  - Notwendige Eingabeaktionen (*)
  - Erwartete Ergebnisse – Werte wie Systemzustände
  - Aktionen, die zur Überprüfung der Ergebnisse notwendig sind (*)
  - (*) sind Bestandteil des Testrahmens / Testtreibers
- Testspezifikation: (vereinfacht) Beschreibung und Begründung der durchzuführenden Testfälle

Begriffe (3)

- Überdeckungsgrad : Maß für die Vollständigkeit eines Tests
- Regressionstest : Wiederholung von Tests nach Änderungen

- *Blackbox-Test*: kein Einblick in die Interna des Testobjekts
- *Whitebox-Test*: innere Struktur Testobjekt prüfen, d.h. alle Interna sind bekannt

Funktionalität testen (1)

- Qualitätsmerkmal Funktionalität überprüfen:
  - Web-Anwendung realisiert geforderte Funktionen
    - Angemessen
    - Vollständig
    - Korrekt
  - Komponenten fehlerfrei integriert zu Gesamtsystem
  - Sicherheit der Daten gewährleistet

Funktionalität testen (2)

- "Klassische" funktionale Tests:
  - Klassentest:
    - Einzeltest
    - Integrationstest
    - Z.B. mit Überdeckungstests (Zweig-, Pfad-Überdeckung)
  - Komponententest (Modultest):
    - Nachweisen, dass die Komponente der Spezifikation entspricht
    - Überprüfen der Schnittstellen
    - Überprüfen der Zustände der Komponente

Funktionalität testen (3)

- "Klassische" funktionale Tests (Forts.):
  - Integrationstest:
    - Funktionales und technisches Zusammenwirken von HW- und SW-Komponenten prüfen
    - Fehlerwirkungen in Schnittstellen und Kommunikation aufdecken
  - Integrationsstrategien:
    - Top-Down, Bottom-Up: bei Web-Anwendungen ungeeignet
    - "Big Bang": Alles direkt zusammentesten: NIE !
    - Anwendungsfallorientiert vorgehen !

Funktionalität testen (4)

- "Klassische" funktionale Tests (Forts.):
  - Funktionaler Systemtest:
    - Gesamtsystem in Bezug auf die funktionalen Anforderungen testen
- Web-spezifische Tests:
  - Link-Test:
    - Fehlerfreiheit, Rechtmäßigkeit von internen und externen Links
    - Checklisten verwenden
    - Link-Checker, z.B. vom W3C, einsetzen

Funktionalität testen (5)

- Web-spezifische Tests (Forts.):
  - Cookie-Test
    - Fachliche, technische Funktionalität eingesetzter Cookies prüfen
    - (hier sind Ergänzungen notwendig: WebStorage etc.)
  - Plugin-Test
    - Fehlerfreie und korrekte Benutzerführung im Hinblick auf benötigte Plugins prüfen
  - Sicherheitstest:
    - Prüft die Eignung, Korrektheit, Unumgänglichkeit und Wirksamkeit der eingesetzten Sicherheitsmaßnahmen

Benutzbarkeit testen (1)

- Content-Test:
  - Erfüllung von Benutzererwartungen
  - Rechtskonformität
  - Erfüllung von Aufklärungspflichten, z.B.
    - Anbieterkennzeichnung
    - Nennung von Autoren, Vertretungsberechtigten
    - Gesetzliche Anforderungen
- Oberflächentest:
  - Einhaltung Dialogrichtlinien, Korrektheit von Standardfunktionen überprüfen

Benutzbarkeit testen (2)

- Browser-Test
- Usability-Test:
  - Gebrauchstauglichkeit überprüfen
  - Unter Beachtung der jeweiligen Zielgruppe !
- Zugänglichkeitstest: (Barrierefreiheit testen)
- Auffindbarkeitstest:
  - Direkter Aufruf per URI möglich?
  - Problemloses Auffinden in Suchmaschine

Effizienz und Zuverlässigkeit testen (1)

- 'Performance' testen:
  - Werden geforderten Funktionen im Rahmen geforderter Bedingungen erbracht?
    - Zeitverhalten
    - Mengenverarbeitung
    - Ressourcenverbrauch

    bei normalen Systembedingungen.
- Lasttest:
  - Verhalten bei steigender Systemlast

Effizienz und Zuverlässigkeit testen (2)

- Skalierbarkeit testen
- Auf Speicherlecks hin prüfen
- Ausfallsicherheit testen:
  - "Failover" prüfen: wird problemlos auf eine vorhandene Sekundärkomponente umgeschaltet?
- Verfügbarkeitstest:
  - Verfügbarkeit / Ausfallrate überprüfen

Werkzeuge (Beispiele)

- Treiber und Platzhalter
- Capture Replay Tools:
  - Aufzeichnung von Tests
  - Wiederholung und damit Automatisierung von Tests
- Code Coverage Tools:
  - Überdeckungen bestimmen
- Monitore:
  - Schnittstellen beobachten
- Logging-Tools:
  - Protokolle zu Abläufen erstellen und auswerten
- Automatisierung der Benutzerinteraktionen:
  - z.B. Selenium IDE / Selenium WebDriver

Der Einsichtige

Wer testet, ist mutig !

Web-Engineering

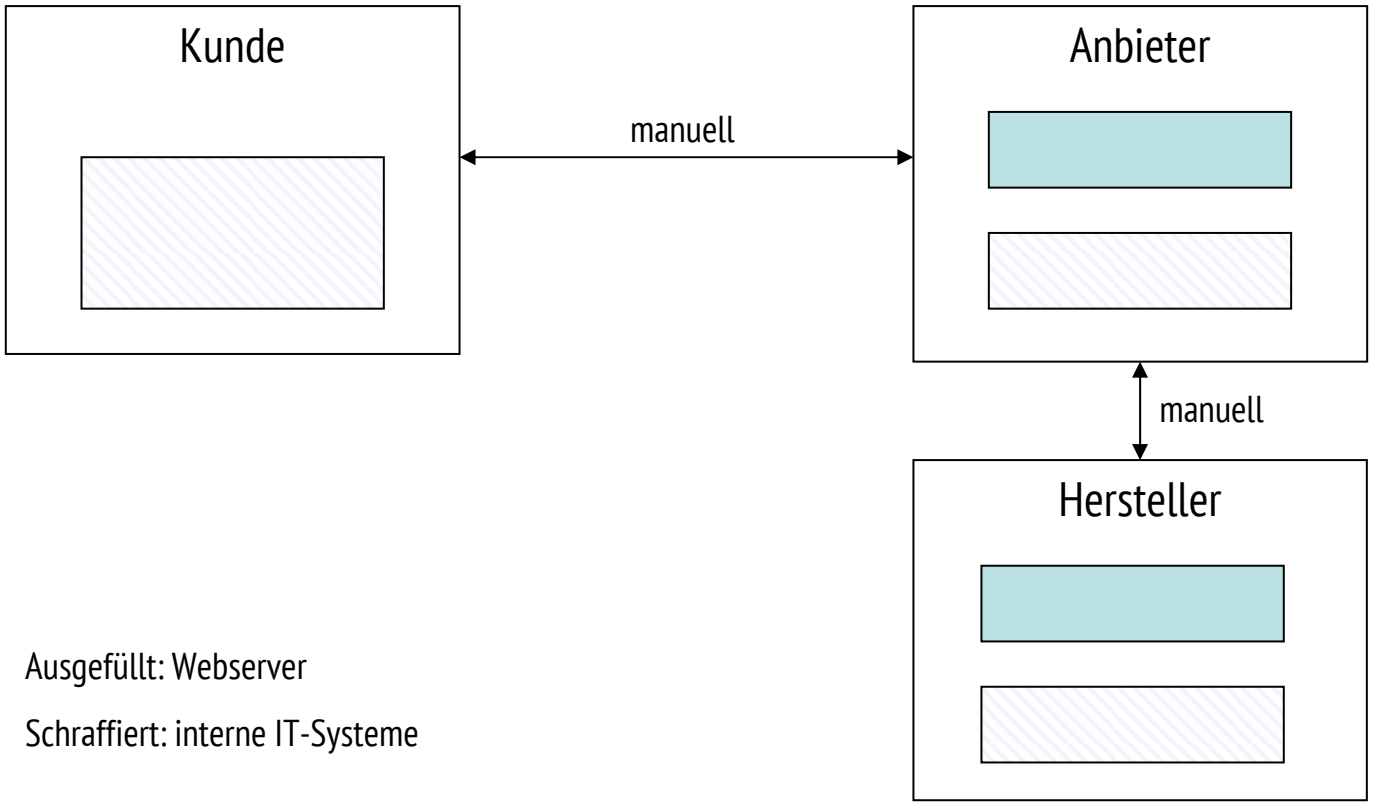
Webservices

Hochschule Niederrhein
University of Applied Sciences
Elektrotechnik und Informatik
Faculty of Engineering and Computer Science

Motivation (1)

- B2B-Integration
  - Leistungsaustausch zwischen Unternehmen (B: Business)
    - Angebotsanforderung
    - Auftrag
    - Lieferung
    - Rechnungswesen
  - Unterschiedliche IT-Systeme bei den Beteiligten
    - Manuelle Bearbeitung von Schnittstellen
    - Automatisierung erfordert einheitliche Protokolle (Abläufe, Datenformate)

Motivation (2)



Ausgefüllt: Webserver

Schraffiert: interne IT-Systeme

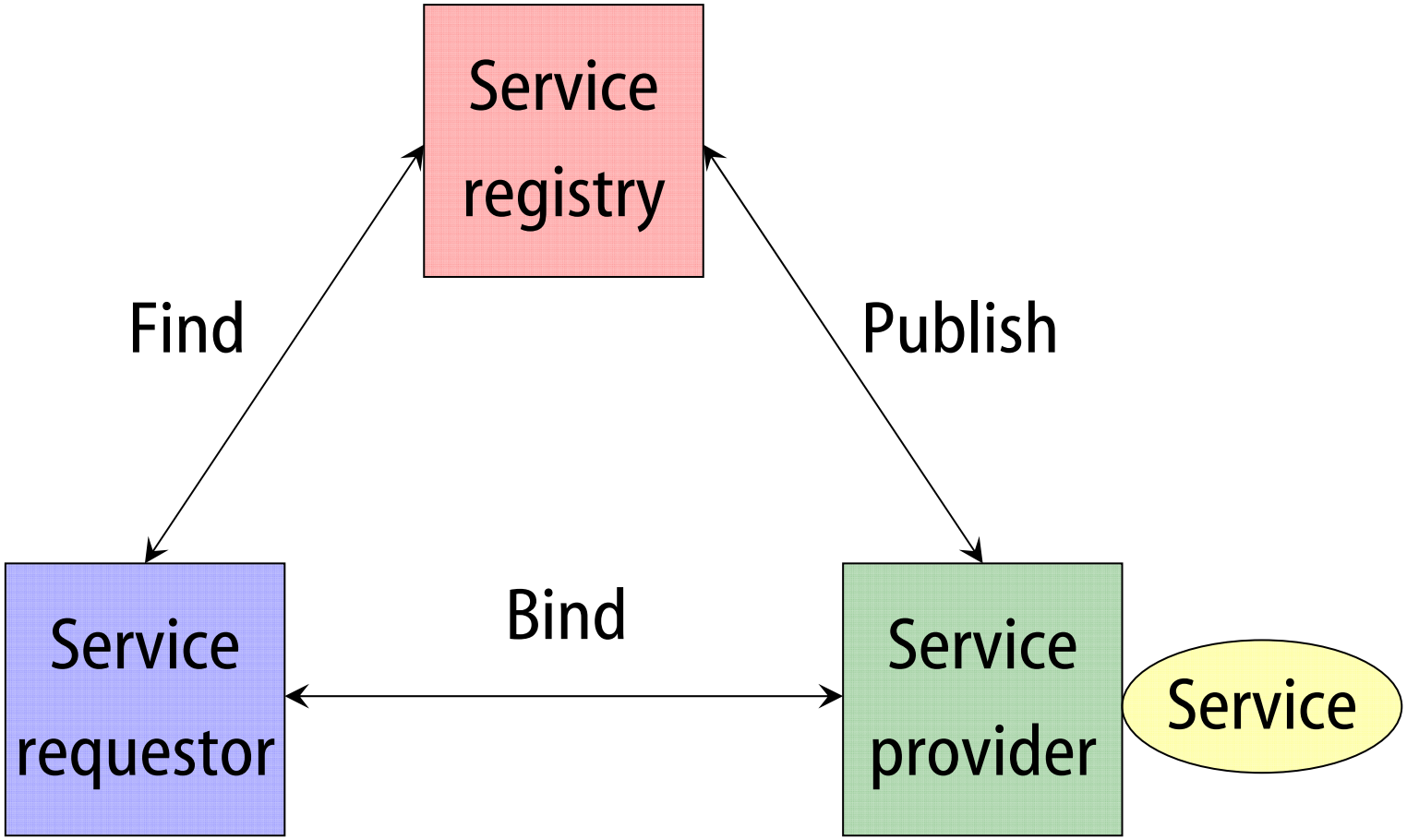
Motivation (3)

- Manuelle Eingriffe ersetzen
- "Services" zur Verfügung stellen:
  - IT-Leistungen eines Unternehmens
  - Automatisierbare Nutzung
- Service:
  - "... a service is a procedure, method or object with a stable, published interface that can be invoked by clients"
  - Aufruf durch andere Programme, nicht manuell
  - Standardisierung:
    - Veröffentlichung (wer bietet was an?)
    - Nutzung

Webservices (1)

- Beispiel einer "Service Oriented Architecture"
  - "Middleware": Integration von Anwendungslandschaften
- Anwendungsübergreifend definierte Schnittstellen
- Lose Kopplung
- Registrierung von Services (= Angebote, Leistungen zu nutzen)
- Bereitstellung von Services

Webservices (2)



Webservices (3)

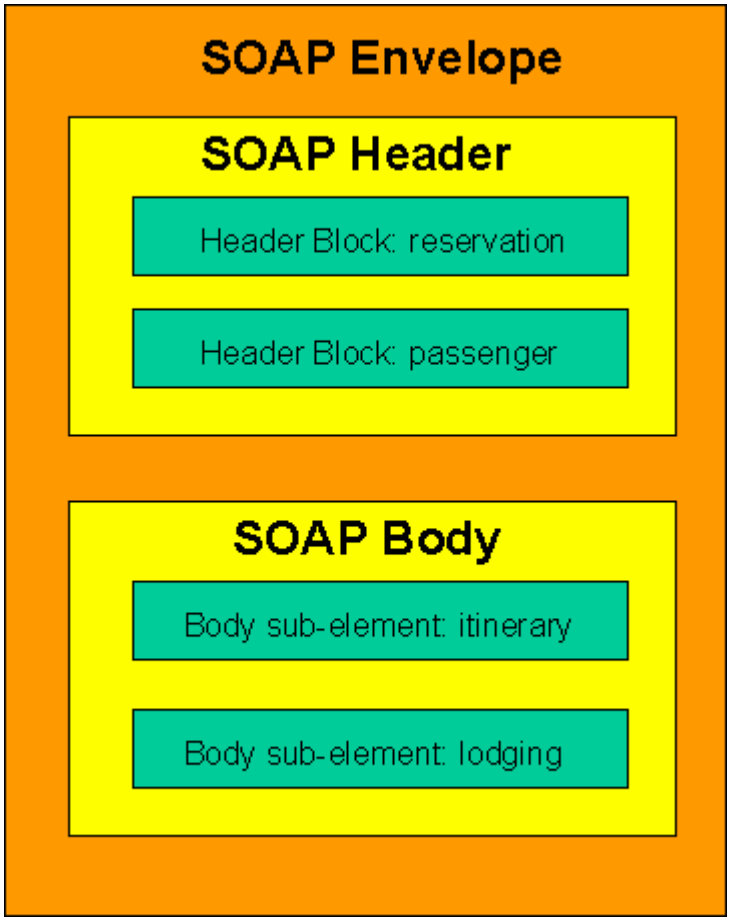
- Vorteile
  - Öffentliche Beschreibung der Services
  - Late Bindung (verzögerte Bindung)
    - Bis zur Auslieferung
    - Bis zur Ausführung
    - Ermöglicht Wechsel des Providers zur Laufzeit des Systems
  - Abrechnungsmodelle
- Kommunikationsstandards (Bsp.)
  - SOAP (Simple Object Access Protocol)
  - WSDL (Web Services Description Language)
  - UDDI (Universal Description, Discovery and Integration)

SOAP (1)

- Standardisierung durch W3C
- XML-basierte Syntax
- Informationskapselung: objektbasierte Verbindungsform
- Einfach aufgebaut, frei verfügbar
- Nachteile:
  - Overhead
  - Sicherheit : keine Verschlüsselung

SOAP (2) (Quelle: W3C)

- Aufbau einer SOAP-Nachricht (siehe auch: `soap_beispiel1_w3c.xml`)
- Header : optional zur Aufnahme von Zusatzinformationen
- Body : die wesentliche Information



WSDL

- Standardisierung durch W3C
- XML-basierte Syntax
- Service-Beschreibung:
  - `types`: welche Arten von Nachrichten kann der Service senden/empfangen
  - `interface`: abstrakte Funktionalität des Service
  - `binding`: wie erfolgt der Zugriff
  - `service`: wie ist der Service erreichbar
- Siehe Beispiel (`wsdl_beispiel1_w3c.xml`)

UDDI

- Standardisierung durch OASIS (`www.oasis-open.org`)
- XML-basierte Syntax
- Registrierung von Webservices
- Dienstebeschreibungen für Kunden:
  - White-Pages: beteiligte Firmen
  - Yellow-Pages: die Dienste
  - Green-Pages: Geschäftsmodelle/Geschäftsbedingungen

Beispiel: Amazon S3 (1)

S3 = Simple Storage Service

- Dokumentation: http://aws.amazon.com/de/documentation/s3/
- Bereitstellung Online-Speicher
- SOAP-Interface:
  - Operationen über SOAP-Messages
  - XML
- REST-Interface
  - Standardisierung durch Nutzung HTTP-Standard

Beispiel: Amazon S3 (2)

S3-Struktur:

- "Buckets" (wörtl.: "Eimer")
  - Enthalten benannte Objekte
  - Keine weitere Hierarchie
  - Komplexität der Strukturierung wird durch die Komplexität der Objekte erreicht (z.B. "Verzeichnisse")
- Authorisierungs- und Versionierungsmechanismen

Beispiel: Amazon S3 – REST (1)

Beim REST-Interface Nutzung der HTTP-Methoden

Anforderung	GET	HEAD	PUT	DELETE
Bucket-Liste /	Liste der Buckets	-	-	-
Ein Bucket /{bucket}	Liste der Objekte im Bucket	-	Bucket anlegen	Bucket löschen
Ein Objekt /{bucket}/{object}	Objekt inkl. Metadaten lesen	Metadaten des Objekts lesen	Objekt inkl. Metadaten schreiben	Objekt löschen

Beispiel: Amazon S3 – REST (2)

Nutzung der HTTP-Statuscodes, z.B.:

- 200: (ok) Anforderung konnte ausgeführt werden
- 404: (not found) Ressource nicht gefunden
  - Z.B. unbekannter Name für Objekt oder Bucket
- 403: (forbidden) Keine Autorisierung vorhanden für die Anforderung
- 400: (bad request) Anfrage kann nicht interpretiert werden
- 409: (conflict)
  - Z.B. Versuch Bucket zu löschen, das noch Objekte enthält