Layout-Modelle für Webseiten und Formulare

Ein Überblick über Cascading Stylesheets und JGoodies

Projektgruppe „Generierung von Web-Anwendungen aus visuellen Spezifikationen“
der AG-Kastens an der Universität Paderborn

erstellt von Stephan Winter

Inhaltsverzeichnis Seite

1. Cascading Stylesheets

1.1 Einführung und Überblick

Cascading Stylesheets ist eine einfache Markup-Sprache, um das Aussehen einer HTML-Datei oder auch XML-Datei zu beeinflussen. In HTML ist es auch ohne Cascading Stylesheets (kurz: CSS) möglich, das Aussehen verändernde Anweisungen einzufügen, wie z.B.:

```
<font color="red" size="1"><b>Dies ist Text</b></font>
```

Dies weist den Browser an, einen Text mit dem Inhalt „Dies ist Text" auf der Ausgabefläche (Canvas) auszugeben. Des Weiteren ist er aber auch angehalten, diese Schrift in roter Farbe, der Größe 1 und (**<b>**) in fett darzustellen.
Warum sollte man nun auf CSS umsteigen bzw. CSS-Tags lernen, wenn doch nahezu alles bereits in HTML integriert ist? Flexibilität, Konsistenz und Trennung von Inhalt und Layout sind die Antworten.
Mit Cascading Stylesheets kann auf allen Webseiten ein konsistentes Layout erzeugt werden, da in dem HTML-Quellcode hauptsächlich nur der Inhalt der Daten und nahezu keine formatierenden Informationen enthalten sind. Diese Informationen werden ausgelagert in so genannte CSS-Anweisungsblöcke oder sogar extern eingebundene CSS-Anweisungsdateien. Damit wird nicht nur der HTML-Quellcode wesentlich vereinfacht und verkleinert, sondern auch wesentlich flexibler. Eine Änderung des Layouts muss nur zentral in den CSS-Anweisungsblöcken (im Idealfall in nur einer einzigen externen CSS-Datei) geändert werden. Inhalt und Layout sind also voneinander getrennt.
Hier wird auch klar, dass dieses Konzept nicht weit von XML entfernt ist. XML selbst hat auch kein eigenes Erscheinungsbild nach außen, sondern repräsentiert lediglich Daten. Das Erscheinungsbild dieser Daten kann dann wiederum (analog zu CSS-Dateien) durch XSLT beeinflusst werden.
Im Folgenden werde ich mich jedoch im Kern auf HTML beziehen, da dies im Gegensatz zu XML ein Kernpunkt der Projektgruppe im Hinblick auf die Aufgabe ist.

Es gibt mehrere Versionen des Cascading Stylesheets Standards. Derzeit ist die aktuellste vom W3 Consortium verabschiedete Version die Version 2.1. 1996 wurde CSS1 bzw. CSS vom W3 Consortium verabschiedet, im Jahre 1998 folgte darauf als logische Folge im Hinblick auf einen größeren Funktionsumfang die Version 2. Versionsnummer 2.1 enthält Bugfixes zur Version 2 und Version 3 ist derzeit in Planung.

Cascading Stylesheets sollen Browser-/Plattform und Geräteunabhängig sein. Dies ist ein Ziel der Cascading Stylesheets, welches jedoch nicht immer umgesetzt werden konnte. In nahezu jedem erhältlichen Browser gibt es mindestens eine CSS-Anweisung, die anders interpretiert wird, als es vorgesehen ist. Jedoch ist zu bemerken, dass die Zeit der Browserkriege vorbei ist und somit auch die Zeit der absolut unterschiedlichen Darstellung. War es früher nicht unüblich, dass ein Webseiten-Ersteller 2 Versionen seiner Seite online stellte, so ist dies mittlerweile aufgrund der Konsistenz in der Darstellung von Webseiten in den populären Browsern (Internet Explorer, Mozilla/Firefox, Opera) nicht mehr nötig.

Cascading Stylesheets können jedoch weit mehr als das Aussehen von Inhalten bzw. Schrift beeinflussen. Auch die Positionierung von Inhalten sowie die Beeinflussung der Sprachausgabe sind mit CSS möglich. Weitere Informationen hierzu findet man in der Liste der kategorisierten CSS-Eigenschaften (Property) in Appendix A.

1.2 Motivationsbeispiel

Hier nun ein erstes CSS-Beispiel in einer HTML-Datei. Die CSS-Anweisungen sind in rot dargestellt.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
<TITLE>Dies ist der Titel meiner Seite</TITLE>
<style type="text/css">
BODY {
        font-family:  "Verdana",  sans-serif;    font-size:
10pt;
}
.farbig {
        background: #AABBCC; padding: 1px 1px 1px 1px;
}
LI {
```

```
        font-size: 10pt; background: white;
        margin: 1px 1px 1px 1px; padding: 5px 5px 5px 5px;
}
</style>
<BODY>
<H1>Willkommen auf meiner Seite</H1>
<P class="farbig">
Schön, dass Du zu mir gefunden hast.
Dafür jetzt hier eine Aufstellung der besten Biere:
<UL>
<LI>Krombacher Pils
<LI>Jever
<LI>Becks
</UL>
<a href="http://www.bier.de">Klick hier</a>
</BODY>
</HTML>
```

Zu sehen sind hier 3 rot dargestellte verschiedene CSS-Anweisungsblöcke, welche von einem **<style>**-Tag umklammert sind. Dieser Tag zeigt dem Browser an, welche Art von Daten folgt. Ein CSS-Anweisungsblock beginnt mit einem Namen (Selector). Darauf folgt der eigentliche, von geschweiften Klammern umschlossene CSS-Anweisungssatz. Diese Anweisungen sind getrennt durch Semikola und immer im Format:

```
Property : value;
```

aufgebaut (z.B. **font-size: 10pt;**).

Der Selector eines Anweisungsblocks bezieht sich immer auf alle HTML-Tags derselben Seite oder ist ein Punkt gefolgt von einem selbst-definierten Namen (wie hier z.B. „farbig“). Damit ist es möglich, einen Anweisungsblock mit den entsprechenden Anweisungen auf verschiedene HTML-Tags zu beziehen. In diesem Beispiel beziehen sich die Anweisungen aus dem Anweisungsblock „farbig“ auf den **<P>**-Tag, dem über das class-Attribut der entsprechende Anweisungsblock zugeordnet wurde. Auch die Kombination ist möglich, sodass sich **P.rot** ausschließlich auf **<P>**-Tags bezieht, die als class-Attribut den Wert „rot“ enthalten.

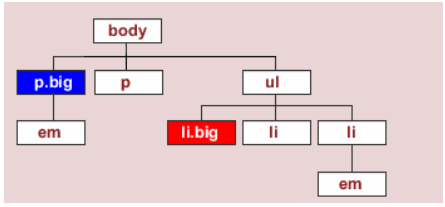
Um nun jedoch nicht in jeder HTML-Datei diese Definitionen einbinden zu müssen und um damit die Wartbarkeit zu erhöhen, ist es möglich die CSS-Anweisungen in eine externe Datei auszulagern. Diese kann dann wiederum in mehrere HTML-Dokumente eingebunden werden, was zu einem konsistenten und leicht wartbaren Design auf allen Seiten führt.

Eine externe CSS-Datei wird über folgenden Aufruf eingebunden:

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

Das rel-Attribut zeigt an, dass nun eine CSS-Datei eingebunden wird. Das href-Attribut gibt den Ort an, wo sich die Datei befindet (im lokalen Dateisystem oder auch per http-Aufruf) und das type-Attribut zeigt den Mime-Typ bzw. die verwendete Stylesheet-Sprache an.

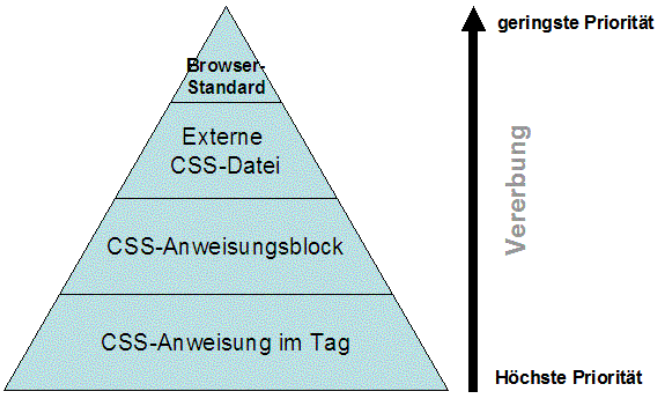
Das Auslagern von CSS-Anweisungen in externe Dateien sollte grundsätzlich erfolgen, da dies die Wartbarkeit und Konsistenz erhöht und den HTML-Quelltext besser vom Layout trennt. Die Vorteile bezüglich der Seitenaufbaugeschwindigkeit seien hier einmal vernachlässigt.



Wie geht der Browser nun vor, um die Elemente alle korrekt darzustellen? Hierzu baut der Browser zuerst einmal einen Document Tree mit n Elementen auf. Für jedes Element wird nun die Attribut-Liste gefüllt (z.B. die Schriftgröße eines Absatzes oder die Umrandung eines Bildes). Zuerst werden die Voreinstellungen für das Element gewählt. Danach überschreiben die vom Benutzer getroffenen Browser-Einstellungen (z.B. absichtlich vergrößerte Schrift) diese Einstellungen. Im nächsten Schritt werden die CSS-Anweisungen der externen Datei gelesen sowie die weiteren mit höherer Priorität folgenden CSS-Anweisungen gelesen und angewendet.

1.3 Warum Cascading?

Cascading Stylesheets haben Ihren Namen nicht ohne Grund. In CSS gibt es ein hierarchisches Vererbungskonzept bezüglich der Attribute. Dies bedeutet, dass es möglich ist, z.B. mit HTML einen Absatz zu erstellen, welcher vorerst das Aussehen analog der Browser-Standards bzw. vom Benutzer vorkonfigurierten Einstellungen erhält. Danach werden einige oder sogar alle Aussehensmerkmale durch



Anweisungen aus einer externen CSS-Datei überschrieben. Unter den CSS-Anweisungsmöglichkeiten ist die externe CSS-Datei die mit der geringsten Priorität. Ein CSS-Anweisungsblock, der direkt in der HTML-Datei definiert ist, überschreibt wiederum die Browser-Standards als auch möglicherweise schon definierte Anweisungen aus einer externen CSS-Datei. Steht also der Hintergrund einer HTML-Seite in den Browser-Standards auf weiß, in der externen CSS-Datei auf rot und in einem CSS-Anweisungsblock auf blau, so wird der Hintergrund blau dargestellt. Die letzte Möglichkeit zur Beeinflussung ist die CSS-Anweisung direkt im HTML-Tag, welches folgendes Aussehen hat:

```
<li style="border : 3px solid blue;">List item</li>
```

Mit dieser Anweisung wird ein Element einer Aufzählung (z.B. Unordered List **<ul>**) bezeichnet, welches eine 3 Pixel breite solide (durchgehende) blaue Umrandung erhält.
Neben dieser impliziten Prioritätenverteilung gibt es jedoch auch noch eine explizite Angabe von Prioritäten (um z.B. einen Block besonders herauszuheben). Hierfür werden die !-Anweisungen genutzt. Ein Beispiel hierfür:

```
! H2 { ... }
```

Hiermit erhält diese Definition des **<H2>**-Tags höchste Priorität. Selbst wenn diese Definition nur in einer externen CSS-Datei (also geringere Priorität besitzt) steht und im Quelltext der HTML-Datei durch einen weiteren H2-Block überschrieben wird, behält dieser Block Priorität.

1.4 Anwendungsmöglichkeiten von Cascasding Stylesheets

Die Anwendungsmöglichkeiten von CSS sind nahezu unbegrenzt, was die Formatierung von Text angeht. Appendix A gibt einen Überblick über die in CSS2 nutzbaren Eigenschaften (Property). Neben der Formatierung von Text, Schrift und Hintergründen ist jedoch auch die Beeinflussung anderer Medientypen möglich, wie z.B. die aureale (akustische) Unterstützung des Browsers, die Druckausgabe oder auch die Gestaltung von Benutzeroberflächen. Es ist also ersichtlich, dass sich CSS nicht nur auf HTML beschränken lassen will.
Die Eigenschaften von CSS lassen sich in folgende Kategorien einteilen:

- Hintergrund und Farbe (background, color)
- Rahmen (border, border-style)

- Abstände (margin, padding)
- Schrift (font-family, font-weight)
- Listen (list-style)
- Druckausgabe (page, widows)
- Positionierung (direction, height)
- Sprachausgabe (pitch, speak)
- Tabellen (border-collapse, table-layout)
- Textformation (text-decoration, text-indent)
- Benutzeroberflächen (cursor, outline)

Anzumerken ist, dass neben all den Möglichkeiten von CSS immer an die Konsistenz gedacht werden sollte. Es ist nicht Sinn der Sache, viele verschiedene Benutzerpanels besonders bunt darzustellen und besonders ausgefallene Textformatierungen zu wählen, sondern ein konsistentes (gern auch farbenfrohes) Panel mit einheitlicher Textformatierung. Dies erhöht den Wiedererkennungs- und damit Usabilitywert.

Zur Fehlerbehandlung von CSS ist zu sagen, dass im Falle eines Fehlers der Browser den kleinstumschließenden Block ignoriert.

```
H1 {
fontfamily: Verdana, Arial;
}
@import “style.css“;
```

Im obigen Beispiel sind 2 Fehler enthalten. Zum einen wird eine Property „fontfamily“ genutzt, welche jedoch (siehe Appendix A) nicht existiert. Da nun im Block H1 ein Fehler aufgetreten ist, wird dieser Block komplett ignoriert.
Der zweite Fehler bezieht sich auf ein noch nicht erwähntes Feature: die @-rule. Eine @-rule ist etwas wie eine Catch-All Regel. Sie kommt immer dann zur Anwendung, wenn kein Anweisungsblock denselben Tag beschreibt. Sie hat also geringste Priorität und überschreibt lediglich die Browser-Defaults. Diese @-rule muss immer vor allen Anweisungsblöcken stehen und ist also in diesem Beispiel am falschen Platz. Daher wird die @-rule ignoriert.

Kommentare werden in CSS wie in Java notiert, also folgendermaßen:

```
H1 {
font-family: Verdana, Arial;
/* font-size: 12px; */
}
```

Unter Values werden die den Properties zugewiesenen Werte (z.B. 12px, 14em, loud, bold) verstanden. CSS nutzt feste, dynamische und auch exotische Maßeinheiten, um die Properties zu beschreiben. Die folgenden Value-Typen sind in CSS2 definiert:

- Integer-Werte

- Längenangaben

relative Längenangaben:

em: Schriftgröße der Schrift proportional zum Tag

```
H1 { line-height: 1.2em }   /* 20% größer als H1 Größe */
```

ex: Höhe der Schrift

px: Größe der Schrift in Pixel abhängig vom Ausgabegerät

absolute Längenangaben:

in: inches

cm/mm:

pt: point (1 point = 1/72-tel eines inches)

pc: pica (1 pica = 12 points)

- Prozentwerte

```
P { line-height: 120% }         /* 120% der 'font-size' */
```

Die Prozentangaben beziehen sich auf einen property-spezifischen Default-Wert.

- URL / Counter

- Farben

```
EM { color: #f00 }              /* #rgb */
EM { color: #ff0000 }           /* #rrggbb */
EM { color: rgb(255,0,0) }      /* integer range 0 - 255 */
EM { color: rgb(100%, 0%, 0%) }
```

- Winkel

deg, rad, grad

- Zeiten

ms, s

- Frequenzen (für die akustische Ausgabe)

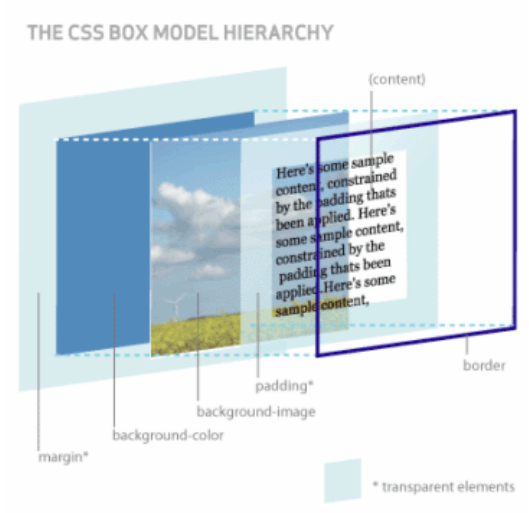
Hz, kHz

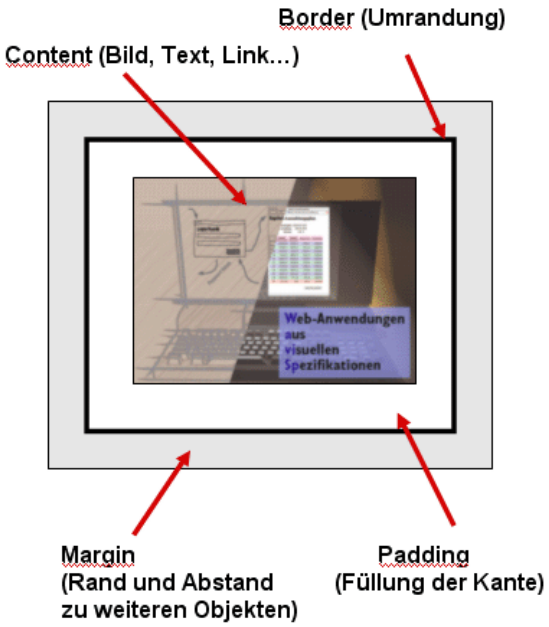
- Strings

1.5 CSS Box Model

Das CSS Box-Model ist eine ausgefeilte Beschreibungstechnik mit der Objekte angeordnet werden können. Mit Objekten sind jedoch nicht nur Bilder gemeint, sondern jegliche Art von Objekten, also vom Abschnitt, über dessen einzelne Sätze bis hin zu jedem einzelnen Buchstaben (z.B. kann man mit der Eigenschaft **`letter-spacing`** die Laufweite der Schriftart beeinflussen). Jedes Objekt erhält eine umschließende Box. Diese Box kann nun beeinflusst werden in Bezug auf ihre Füllung zwischen Umrandung und Objekt (**`Padding`**), die Umrandung (**`Border`**) und den Abstand zu weiteren Objekten (**`Margin`**). Damit ist nahezu jede Ausrichtung möglich. Das Box-Model ist also an sich keine Funktionalität, welche man aktiv nutzen kann, sondern eine rein deklarative Beschreibung bzw. ein Modell von CSS, welches man indirekt über Properties wie „**`margin-width`**“ beeinflusst.

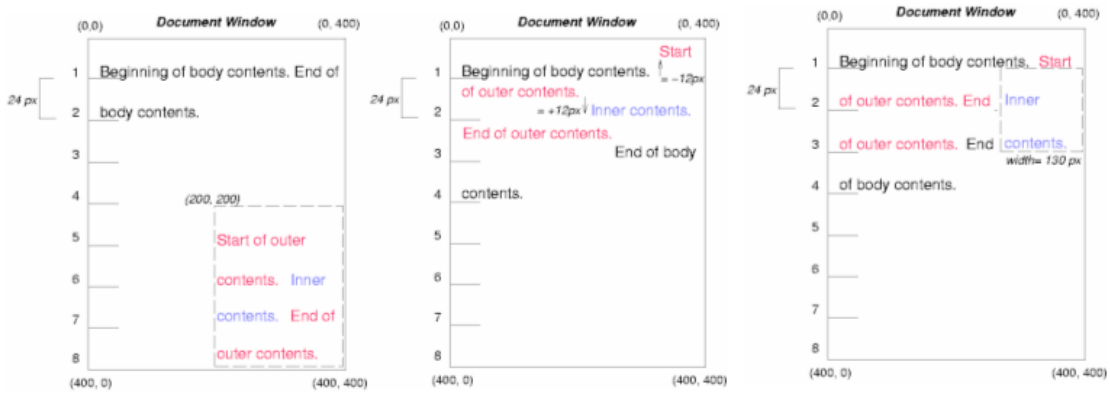
Die folgenden Abbildungen illustrieren das Konzept und seine Bestandteile noch einmal visuell.





1.6 Positionierung mit CSS

Positionierung in CSS ist durch das vorher erläuterte Box-Model besonders einfach. Nahezu jedes Objekt lässt sich in einen Container bzw. Abschnitt stellen, welchen man wiederum frei auf der Arbeitsfläche/Ausgabefläche (canvas) anordnen kann. Ein typisches Beispiel für eine solche Box ist die Navigationsleiste einer Webseite. Diese kann man in einen eigenen Container platzieren, um dann nachträglich noch die konkrete Anordnung auf dem Bildschirm zu wählen (z.B. am linken oder rechten Rand).



Absolut

```
<STYLE type="text/css">
.inner {
  position: absolute;
  top: 200px;
  left: 200px;
  width: 200px;
  color: red;
}
</STYLE>
```

Relativ

```
<STYLE type="text/css">
.inner {
  position: relative;
  top: -12px;
  color: red
}
</STYLE>
```

Fließend (Float-Box)

```
<STYLE type="text/css">
.inner {
  float: right;
  width: 130px;
  color: blue
}
</STYLE>
```

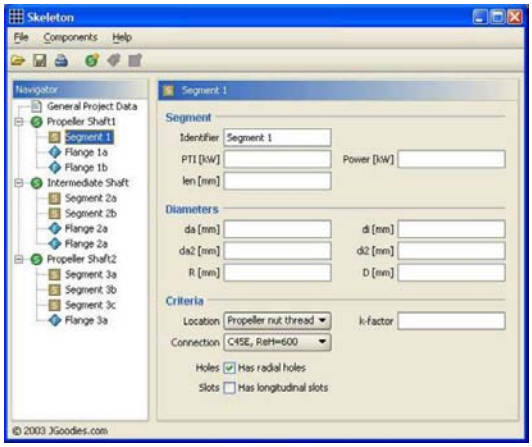
Die Positionierung eines solchen Containers kann absolut, relativ oder fließend (analog zu dem Flow Layout in Java) geschehen.

Wie im obigen Beispiel ersichtlich kann die absolute Ausrichtung über Spezifikation der x- und y-Koordinaten (abhängig von der 0,0-Koordinate links oben im Browser) erfolgen. Die relative Ausrichtung erfolgt über Angabe von relativen Werten. Im obigen Beispiel wird in der Mitte „-12px“ angegeben, was bedeutet, dass der Text um 12 Pixel erhöht gegenüber der normalen Schriftposition dargestellt wird. Beim Flow-Layout wird lediglich die Ausrichtung (z.B. Ausrichtung am rechten Rand wie im Beispiel rechts) angegeben. Die übrigen Objekte arrangieren sich um diesen Container bzw. um diese Box herum.

2. JGoodies

2.1 Einführung und Überblick

JGoodies ist ein Java-Framework, welches von Karsten Lentzsch aus Kiel entworfen worden ist. Der geistige Vater dieses Frameworks stammt aus dem Business Bereich und beschäftigte sich mit der Gestaltung von GUIs. Daher sind die meisten Beispiele und Gestaltungsmöglichkeiten auf das Aussehen von Business Software ausgelegt. Dies ist daran erkennbar, dass sich das Aussehen der GUIs stark an dem Aussehen von Formularen orientiert. Objekte orientieren sich an Fluchten, die Größen von Objekten (wie z.B. Eingabefelden, Buttons) werden aneinander angeglichen, Eingabebereiche sind sinnvoll getrennt und es handelt sich oft um die typische Explorer-ähnliche Sicht mit einer Navigation auf der linken Seite, dem Inhalt auf der rechten Seite und einer Anwendungsleiste (Button-Bar) am oberen Rand.



Das JGoodies Framework hat sich mittlerweile (zumindest im Business Bereich) zum Standard für die Gestaltung von GUIs entwickelt. Dies liegt zum einen an den konsistenten Layouts mit denen man die Usability-Experten befriedigen kann und zum anderen auch an der Einfachheit der Programmierung dieser GUIs. Sind ohne JGoodies mehrere hundert Zeilen Code nötig, um ein Layout wie oben gezeigt zu erzeugen, so ist dies in JGoodies mit einem Zehntel des Codes möglich. Insbesondere weist der Autor darauf hin, dass mit JGoodies gutaussehende GUIs leicht zu erstellen und schlechtaussehende GUIs nahezu unmöglich („make simple things easy and the hard stuff impossible“) sind. Das Framework ist nach Aussage des Autors für 90% der Fälle ideal bzw. ausreichend und für 10% Sonderfälle bewusst nicht nutzbar.

2.2 Konzepte

Das JGoodies Framework basiert auf den folgenden 5 Prinzipien:

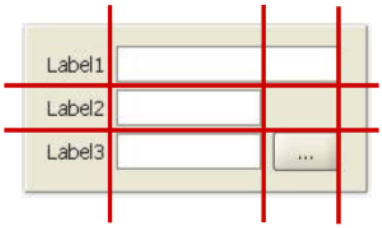
- JGoodies-GUI-Entwicklung orientiert sich an dem typischen Vorgehen eines Mediengestalters bei der Erstellung von Formularen (z.B. Fluchten suchen, günstige Farbkombinationen wählen etc.)

- Mit dem Stichwort „separate concerns“ meint der Autor, dass inhaltlich zusammenhängende Objekte (z.B. Eingabefelder) gruppiert gehören (z.B. in Boxen), jedoch umgekehrt auch keine 2 Angelegenheiten innerhalb eines zusammenhängenden Raums stehen sollten. Als Beispiel sei hier die klare Trennung von Navigation, Inhalt, Infobox bei Webseiten genannt.
- JGoodies nutzt (ähnlich wie die einzelnen Werkzeuge innerhalb des ELI-Systems der AG Kastens) eine umfangreiche Spezifikationssprache
- Es sollten der Einfachheit halber Abkürzungszeichenketten genutzt werden, wie z.B. **`br`** anstelle von **`bottom right`**. Hier sollte jedoch beachtet werden, inwiefern diese Abkürzungen die Lesbarkeit verschlechtern.
- Beispiele - insbesondere wenn wieder verwendbar - sind sehr wichtig, um die Nutzung des Frameworks zu verinnerlichen. Dies wird besonders durch die Factory Classes unterstützt. Factory Klassen sind vorgefertigte Code-Beispiele wie z.B. eine spezielle Button Bar oder ein oft genutztes wieder verwendbares Schema für ein Panel.

Neben diesen 5 Prinzipien ist es die Hauptaufgabe von JGoodies ein konsistentes, benutzerfreundliches Aussehen der GUI für den Anwender zu erstellen und dies mit geringem Aufwand für den Entwickler zu erreichen. Dies erhöht die Produktivität auf der Seite des Anwenders sowie der des Entwicklers.

2.3 Motivationsbeispiel

Im Folgenden nun ein Beispiel zur Motivation für JGoodies. Dies zeigt gleich die Einfachheit bei der Erstellung von GUIs.



Das nebenstehende Panel soll in der gezeigten Anordnung in Java erstellt werden. Das Hinzufügen der Panelelemente ist in Java selbst kein Problem, die Ausrichtung jedoch dagegen ist schon schwieriger. Hier kommt JGoodies ins Spiel. Werfen wir einen Blick auf den Quellcode zum Beispiel:

```
1: FormLayout layout = new FormLayout(
2: “pref, 4dlu, 50dlu, 4dlu, min”,
3: “pref, 2dlu, pref, 2dlu, pref”);
4: layout.setRowGroups(new int[][]{{1, 3, 5}});
5: JPanel panel = new JPanel(layout);
6: CellConstraints cc = new CellConstraints();
```

```
 7: panel.add(new JLabel("Label1"), cc.xy (1, 1));
 8: panel.add(textField1, cc.xyw(3, 1, 3));
 9: panel.add(new JLabel("Label2"), cc.xy (1, 3));
10: panel.add(textField2, cc.xy (3, 3));
11: panel.add(new JLabel("Label3"), cc.xy (1, 5));
12: panel.add(textField3, cc.xy (3, 5));
13: panel.add(detailsButton, cc.xy (5, 5));
```

In Zeile 1 bis 3 wird ein neues Formlayout erstellt. Hiermit wird die grobe Struktur tabellenartig festgelegt. Man überlegt sich also vorher im Schritt des „separating concerns", wie viele Container für die Elemente benötigt werden. Diese Anzahl von Zeilen und Spalten gebe ich nun als Parameter an das neue Formlayout. In diesem Beispiel werden in der 2. und 3. Quelltextzeile das Zeilen- und Spaltenlayout festgelegt. Hier sei angemerkt, dass auch Größenangaben wie „min" oder „pref" gültig sind. Min bzw. Max bedeutet, dass die minimale Größe bzw. maximale Größe des später einzufügenden Objekts gewählt werden soll. Die Angabe pref bedeutet, dass die eingefügten Elemente, wenn möglich, ihre bestmögliche Größe selbst wählen sollen. Ein Button z.B. hat eine optimale Höhe und Breite (die sich nach der Beschriftung richtet). Hier kann man bestimmen, dass der Button in seiner bestmöglichen Darstellung angezeigt wird. Dies geschieht natürlich nur, solange kein anderes Objekt dadurch verdrängt wird. Die Größenangabe **`dlu`** bedeutet „Dialog line units" und ist eine dynamische Größe, abhängig von dem Ausgabemedium und den Ausgabeeinstellungen (Schriftgröße etc.).

In der 4. Quelltextzeile werden die Zeilen 1,3 und 5 gruppiert. Damit erhalten diese Zeilen auch dieselbe Höhe.

Das in Zeile 5 definierte JPanel erhält als Parameter das nun definierte Formlayout als Parameter übergeben. Im nächsten Schritt müssen die einzufügenden Objekte in das Formlayout (also dem Tabellenkonstrukt) eingefügt werden. Dazu wird ein Cellconstraints-Objekt erzeugt, über das man mit den x- und y-Werten die exakte Zelle angeben kann.

In Zeile 7 bis 13 werden die Objekte mit Hilfe des Cellconstraint Objekts in die virtuelle Tabelle eingefügt.

Als spätere Referenz noch einmal in Kurzform die Schritte zur Erzeugung eines Forms:

1) FormLayout erzeugen

```
new FormLayout("right:pref, 10px, left:pref:grow",   // 3 Spalten
               "pref, 4px, pref, pref:grow");        // 4 Zeilen
```

pref = preferred Size (Breite, die für das Element am besten passt)

right = rechtsbündig ausgerichtet

grow = zur entspr. Seite sich vergrößernd (falls Platz übrig)

2) Panelbuilder mit erzeugtem Layout aufrufen (es können also versch. Panelbuilder genutzt werden -> höhere Abstraktion / Wiederverwendbarkeit)
3) Zeilen/Spalten gruppieren (erzeugt gleich Höhe, Breite)

```
layout.setRowGroups(new int [] [] { {1,4}, {2,3}});  // Zeilen 1+4 & 2+3 selbe Höhe
```

4) CellConstraints Object erzeugen (dient zur Positionierung nicht einzelner Elemente, sondern Elementgruppen, wie z.B. allen Feldern auf einer Zeile)

```
CellConstraints cs = new CellConstraints();
cc.xy(2, 1); // zweite Spalte, erste Reihe
cc.xy(2, 1, "r, b")  // wie oben, aber rechts unten ausgerichtet
```

5) Elemente hinzufügen: builder.addLabel("Identifier", cc.xy(1,3));

3. Fazit

Cascading Stylesheets und JGoodies sind zwei sehr verschiedene Ansätze, was den Nutzen für die Projektgruppe betrifft. Cascading Stylesheets ist ein absolutes Muss bei der Gestaltung von Webseiten. Trotz der Inkompatibilitäten bei den Browsern was die Interpretation von Stylesheets angeht sind die größten Probleme mittlerweile in den neueren Versionen behoben und daher vernachlässigbar. Stylesheets sind sehr wichtig, um ein konsistentes Design zu erzeugen und HTML-Code von Gestaltungsmerkmalen zu trennen. Dies gilt trotz der Tatsache, dass in der Projektgruppe ein Generator die HTML-Seiten erzeugt und es damit vordergründig irrelevant ist, wo die Definitionen das Aussehen betreffend gemacht werden. Die Vorteile bezüglich der geringeren Größe der Webseiten sowie die Auslagerung des Designs in nur eine Datei sind wertvoll für Anwender und ggf. einen Entwickler, der über den vom Generator erstellten Code hinaus das Projekt weiterentwickelt.
In Verbindung mit Devil ist die Nutzung von CSS an sich recht einfach zu bewerkstelligen. In Devil ist es möglich, erzeugten Objekten Properties zuzuordnen. Hier können dann die meisten Eigenschaften (wie Rahmendicke, Schriftgröße etc.) bei eben diesen Properties gesetzt werden. Statt das Setzen der Hintergrundfarbe direkt als HTML-Tag auszugeben, wird diese Eigenschaft während der Generierung in die externe CSS-Datei geschrieben.

Was JGoodies angeht, so sind lediglich die theoretischen Ansätze nutzbar, da unsere Anwendung nicht in Java geschrieben wird. Insbesondere das formular-basierte Layout ist der Übersichtlichkeit zuträglich und sollte angewandt werden. Auf den meisten Webseiten findet sich das übliche Layout wieder: Navigation links, Inhalt rechts und ggf. Funktionsleiste oben. Diese Anordnung sollte beibehalten werden, da der Benutzer an ein solches Layout gewohnt ist.
Interessant ist auch die Einführung neuer Maßeinheiten wie die dlu (dialog line units), die sich der gewählten Schriftgröße des Benutzers anpassen oder eine bevorzugte/optimale Wunschgröße für Objekte. Damit wird die Ausrichtung von Objekten wesentlich leichter. Eine tabellenartige Anordnung der Objekte ist dafür Voraussetzung. Dynamische Maßeinheiten haben zudem den Vorteil, dass der Benutzer später noch Einfluss auf das Design nehmen kann, was insbesondere für sehbehinderte Menschen wichtig ist, die die Schriftgröße stark erhöhen müssen.

Auch der Punkt „Separate concerns“ der 5 Regeln der JGoodies sollte Beachtung finden, jedoch erschließt sich dies meist aus einer konsequenten Trennung von z.B. Navigation, Inhalt und Infoleiste.
Es sollte dem Benutzer in jedem Falle immer möglich sein, die gewünschten Sinnabschnitte im Editor zu verwirklichen.

Literaturverzeichnis

Cascading Style Sheets Level 2 Specification, W3C Recommendation 12-May-1998, URL: http://www.w3.org/TR/REC-CSS2

Introduction to CSS3, URL: http://www.w3.org/TR/2001/WD-css3-roadmap-20010523/

W3C Web Style Sheets, URL: http://www.w3.org/Style/

JGoodies Karsten Lentzsch, URL: http://www.jgoodies.com/

Appendix A – Übersicht über Eigenschaften in CSS2

Text Properties

Property	Description	Possible Values
letter-spacing	Controls the amount of space between each letter in a section of text.	normal (default) number of pixels
line-height	Controls the amount of vertical space between lines of text	normal (default) number of pixels percentage
text-align	Controls the alignment for a section of text	browser decides (default) left right center
text-decoration	Controls what the text looks like	none (default) underline overline line-through blink
text-indent	Controls the indentation of the first line in a section of text	0 (default) number of pixels percentage
text-transform	Changes the case of a section of text	none (default) uppercase lowercase capitalize
vertical-align	Controls the vertical alignment of a section of text	baseline (default) sub super top text-top middle bottom text-bottom
word-spacing	Controls the amount of space between words (doesn't work as of yet)	normal (default) number of pixels

Font Properties

Property	Description	Possible Values
font-family	Controls the type of font shown on the page	browser decides (default) font family name
font-size	Controls the size of the font	medium (default) number of pixels percentage

font-style	Controls the style of the font	normal (default) italic oblique
font-size	Controls the size of the font	medium (default) number of pixels percentage
font-variant	Controls the variant of the font	normal (default) small-caps
font-weight	Controls the boldness of the font	normal (default) lighter bold bolder 100 200 300 400 500 600 700 800 900

Color/Background Properties

Property	Description	Possible Values
color	Controls the color of the text	browser decides (default) color name
background-attachment	Controls the scrolling of the background	scroll (default) fixed
background-color	Controls the color of the background	transparent (default) color name
background-image	Allows you to set a background image	none (default) image url
background-repeat	Allows different patterns of background repetition	repeat (default) repeat-x repeat-y no-repeat
background-position	Controls the position of the background on the page	0% 0% (default) postion in pixels ie {20,20} percentage ie {5%,7%} top bottom left right center

Box Properties

Property	Description	Possible Values
width	Controls the width of a section	auto (default) number of pixels percentage
height	Controls the height of a section	auto (default) number of pixels percentage
border-color	Controls the border color of a section	default text color (default) color name
border-style	Controls the style of a border	none (default) solid double
border-width	Controls the width of a border	undefined (default) number of pixels thin medium thick
border-top-width	Controls the width of a border side	medium (default) number of pixels thin thick
border-left-width	Controls the width of a border side	medium (default) number of pixels thin thick
border-right-width	Controls the width of a border side	medium (default) number of pixels thin thick
border-bottom-width	Controls the width of a border side	medium (default) number of pixels thin thick
margin-top	Controls the width of a margin from the specified side	0 (default) number of pixels percentage
margin-left	Controls the width of a margin from the specified side	0 (default) number of pixels percentage
margin-right	Controls the width of a margin from the	0 (default)

	specified side	number of pixels percentage
margin-bottom	Controls the width of a margin from the specified side	0 (default) number of pixels percentage
padding-top	Controls the amount of padding from the specified side	0 (default) number of pixels percentage
padding-left	Controls the amount of padding from the specified side	0 (default) number of pixels percentage
padding-right	Controls the amount of padding from the specified side	0 (default) number of pixels percentage
padding-bottom	Controls the amount of padding from the specified side	0 (default) number of pixels percentage
float	Controls the floating of a section	none (default) left right
clear	Defines whether a section disallows other sections on its sides	none (default) left right

Classification Properties

Property	Description	Possible Values
white-space	Controls the white space formatting of a section	normal (default) pre nowrap
display	Controls the display of a section	block (default) inline list-item none
visiblity	Controls the visibility of an element	inherit (default) visible hidden
z-index	Controls the layering of an element	auto (default) number