DHBW
Duale Hochschule
Baden-Württemberg

HARMAN

Web- und Multimediabasierte Informationssysteme

Vorlesungsmitschrieb

des Studiengangs

Informationstechnik

von

Jan Ulses

15. September 2014

Dozent:	Jürgen Röthig
E-Mail:	jr@roethig.de
Vorlesungszeitraum:	29.09.14 - 31.03.14
Klausurtermin:	19.12.2014
Autor:	Jan Ulses
Kurs:	TINF12B3
Ausbildungsfirma:	Harman/Becker Automotive Systems GmbH
Studiengangsleiter:	Jürgen Vollmer

Inhaltsverzeichnis

KAPITEL 1

XML

- eXtensible Markup Language
- Mittel um konkrete Auszeichnungssprachen zu definieren
- XML selbst ist eine Metasprache, keine eigene (konkrete) Sprache

Auszeichnungssprache: Sprache um reinen Text weitere Eigenschaften mitzugeben

Designorientiert: Textbestandteile bekommen Aussehen (z.B. Fettdruck, rote Farbe). Beispiel: klassisches Word 1990, Druckformate (PostScript, PCL)

Strukturorientiert: Struktur oder spezielle Funktion (z.B. Überschrift, Absatz, Liste, Tabelle). Beispiel: HTML, LaTeX, SGML, die meisten XML-basierten

1.1 Beispiele für XML-basierte Sprachen

- XHTML (auch HTML5 in XML-Variante)
- SVG (Grafikformat)
- XSD (Sprache zur Definition XML-basierter Sprachen)
- Viele Konfigurationsdateien vieler Software-Pakete (z.B. Apache)
- Dokumentformate (aktuellere) von Microsoft Office (.docx) oder Open Office
- Austauschformate für Inhalte von relationalen Datenbanken

1.2 DOCTYPE-Deklaration

```
<!DOCTYPE html(root Tag) | PUBLIC(bzw. SYSTEM, PRIVATE) "Public-Id"(bei
    PUBLIC) "Syst-Id"(nicht bei PRIVATE)>
```

Listing 1.1: Syntax einer DOCTYPE-Deklaration

Public-ID: ungefähr wie bei HTML "–//W3C/DTD/XHTML1.1/EN"
System-ID: URL, verweist auf konkrete Grammatik in Form einer DTD

1.3 Wesentliche Eigenschaften von XML-Dateien

Es gibt zwei wesentliche Eigenschaften, welche jedes XML-basierte Dokument erfüllen muss bzw. sollte.

- Wohlgeformtheit (z.B. XML-Deklaration)
  - Wurzel-Tag, welcher das komplette Dokument umschließt
  - Tags paarweise, also Start- und Endtag
  - Korrekte Schachtelung (letzter geöffneter Tag zuerst schließen)
- Gültigkeit (z.B. DOCTYPE-Deklaration, insbesondere Verweis auf DTD)
  - Entspricht einer konkreten Grammatik (Tagnamen, Attributnamen und Zugehörigkeit, Enthaltenseinsmodell (Inhalt eines Tags)

1.4 Document Type Definition

Document Type Definitionen (kurz: DTD)

- Definiert eine konkrete Grammatik (XML-basiert)
- Besteht aus Text
- Besteht nur aus Deklarationen (wegen fehlendem Wurzeltag nicht XML-basiert)

1.5 ELEMENT-Deklaration

```
<!ELEMENT tagname inhaltsmodell>
```

Listing 1.2: Syntax einer ELEMENT-Deklaration

tagname: Name des Elements/Tags bestehend aus Buchstaben (Klein- und Großschreibung wird unterschieden, <bla> ist nicht gleich <bLa>) und Ziffern, 1. Zeichen muss Buchstabe oder Unterstrich sein, theoretisch beliebig lang, praktisch kleiner 256 Zeichen, keine Umlaute verwenden.

inhaltsmodell:

EMPTY	(Bsp. aus XHTML: <!ELEMENT br EMPTY> für leere Tags ohne Inhalt)
(#PCDATA)	für beliebige Zeichenfolgen (außer „<“, „>“, „&“ und „”“) insbesondere keine Tags z.B. <!ELEMENT title (#PCDATA)>
sequenz	z.B. (tagname1, tagname2) → Abfolge von tagname1 und tagname2 z.B. <!ELEMENT html (head, body)>
auswahl	z.B. (tagname1 — tagname2)
gemischt	((#PCDATA) — auswahl)*)

Alle Inhaltsmodelle können mit nachgestellten Quantoren versehen werden:

- (inhaltsmodell)* beliebig oft (inkl. Keinmal)
- (inhaltsmodell)+ beliebig oft, aber mindestens einmal
- (inhaltsmodell)? einmal oder keinmal

Entitäten:

&lt	„<“	Less than
&gt	„>“	Greater than
&amp	„&“	Ampersand
&quot	„”“	Quotation mark
&auml	„ä“	
&Auml	„Ä“	

1.6 ATTLIST-Deklaration

```
<!ATTLIST tagname attrname attrtype voreinstellung (optional) >
```

Listing 1.3: Syntax einer ATTLIST-Deklaration

attrname: Name des Attributs, genauso aufgebaut wie Tagnamen

attrtype:

CDATA	beliebige Zeichenfolgen inklusive „<“ und „>“, Einschränkung bei einfachen/doppelten Anführungszeichen
ID	dokumentweit eindeutiger Wert, Einschränkung an Werteraum wie bei Tag- und Attributnamen! d.h. z.B. Zahlenwerte sind keine gültigen Werte vom Typ ID! Beispiel aus HTML: <!ATTLIST a id ID>
IDREF	Referenz/Verweis auf einen ID-Wert, Einschränkung der Werte wie oben, aber keine Eindeutigkeit gefordert, da beliebig oft auf denselben ID-Wert verwiesen werden darf
IDREFS	beliebig viele ID-Werte, getrennt durch Leerzeichen
NMTOKEN(S)	„Name“, d.h. Zeichenfolge von beliebig vielen Buchstaben, Ziffern, manchen Sonderzeichen (insb. aber kein Leerzeichen), auch das erste Zeichen darf ein beliebiges Zeichen der Zahlenmenge sein bzw. mehrere Namen durch Leerzeichen getrennt aufzaehlung: (nmtoken1—nmtoken2—nmtoken3—. . .) der Attributwert kann nur einer der aufgeführten „Namen“ sein
ENTITY	Verweis auf Entitäten → externe (auch binäre) Daten
ENTITIES	Verweis auf Entitäten → externe (auch binäre) Daten
NOTATION	Daten mit spezieller Interpretation

voreinstellung:

#REQUIRED	Pflichtattribut
#IMPLIED	optionales Attribut
#FIXED	wert, Attribut mit festem Wert wert
wert	#IMPLIED-Attribut mit Default-Wert wert
[fehlt]	#IMPLIED-Attribut ohne Default-Wert

Die „Gültigkeit“ einer XML-Datei kann anhand der DOCTYPE-Deklaration und der darin referenzierten DTD überprüft werden → mittels einem Validator z.B. für HTML-Dateien "http://validator.w3.org/".
Problem: Zugriff des Validators auf die DTD? Muss die DTD auf einem öffentlich zugänglichen WebSpace liegen? → NEIN, Abhilfe: Inline-DTD, siehe das Beispiel aus Listing 3.1 auf Seite 12.

```
<?xml version="1.0" ?>
<!DOCTYPE bla [
  <!ELEMENT ...>
    .
    .
  <!ATTLIST ...>
    .
    .
]>
<bla>
    .
    .
</bla>
```

Listing 1.4: Inline-DTD Beispiel

1.6.1 Beispiel

Name	Vorname	Matrikelnummer	Kursbezeichnung	Wahlfach
Müller	Max	012345	TINF12B3	WuMBasis
Maier	Moritz	4711	TINF12B3	Shit
Schulze	Marta	0815	TINF12B5	Gaming

```
<studis>
  <studi matrikelnummer="012345">
    <name nach="Mueller" vor="Max" />
    <kurs> TINF12B3 </kurs>
    <wahlfach> WuMM </wahlfach>
  </studi>
  <!-- entsprechend fuer Maier und Schulze -->
</studis>
```

Listing 1.5: Listeneinträge

```
<!ELEMENT studis (studi)*>
<!ELEMENT studi (name, kurs, wahlfach)>
<!ELEMENT name EMPTY>
<!ATTLIST name nach CDATA REQUIRED attrtype evtl. NMTOKEN vor CDATA
    REQUIRED attrtype evtl. NMTOKENS>
```

Listing 1.6: Baumerstellung per !ELEMENT

Anzeige abstrakter XML-Daten (nicht HTML oder SVG) im XML-fähigen WebBrowser?

- strukturierte Liste (mit Einschränkungen, Elemente aus- und einklappbar)
- nicht besonders anschaulich
- kann mit CSS deutlich „aufgehübscht“ werden
- bessere Variante: XSLT (XML Style Sheet Language Transformation)
  Achtung: Trotz des Namensbestandteils “Stylesheet macht eine XSLT viel mehr als nur Aussehen festzulegen!

KAPITEL 2

XSL

2.1 Bestandteile

Die XML Stylesheet Language besteht aus:

- XSLT: XSL Transformation, Sprache zur Transformation von „XML-Konstrukten“ in andere XML-Konstrukte (oder auch „Konstrukte“ in textbasierten Sprachen)
- XPath: XML Path Language, Sprache zur Auswahl von spezifischen „XML-Konstrukten“ aus der XML-Quelldatei
- XML-FO: XML-Formatting Objects, spezielle XML-basierte Sprache zur layoutgetreuen Ausgabe (nicht struktur- sondern designorientierte Sprache)
- im Folgenden für uns in der Vorlesung interessant: XSLT, XPath nicht jedoch XML-FO

Bsp. für Anwendung: XSLT zur Wandlung der abstrakten „Studis-Datei“ in eine HTML-Datei mit entsprechender Tabelle der Studis

Wer führt die Transformation durch?

- *standalone-Tool:* XSLprocessor (in gängigen Linux-Distributionen enthalten) Apache xalan saxon (von Michael Kay) (unterstützt auch XSLT Version 2)
- *serverseitig:* Integration der XSLT in einem WebServer, d.h. der WebServer liefert bei Anforderung der XML-Datei bereits die mittels XSLT transformierte Datei aus! Apache Project Cocoon Perl-Modul AxKit
- *clientseitig:* integriert in gängige WebBrowser → Mozilla Firefox, MS Internet Explorer, Opera, Chrome, Safari können XSLT!

KAPITEL 3

XSLT

3.1 Transfomation

Werkzeug zur Transformation von XML-basierten Daten in (meist andere) XML-basierte Daten.

```
<?xml version="1.0" ?> <!--Hinweis: Attribut encoding="UTF-8" ist bei
    XML default-->
<!DOCTYPE studis SYSTEM "url/zur DTD"> //<?+<! sind Deklarationen wobei
    <! auf > endet.
[KEINE DOCTYPE-Deklaration!]
<xsl:stylesheet
  version="1.0" //version->Namensraumdeklaration fuer XSLT, Praefix->
      Postfix
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="Namespace der Ausgabesprache, z.B. HTML" //
      Namensraumdeklaration fuer Ausgabesprache, Verwendung ohne Postfix
      und Praefix (Grund: Ersparung von Schreibarbeit)
>
  <xsl:output method="xml" encoding ="UTF-8" //method->auch html( bitte
      nicht angeben!) oder text
  doctype-public"..." //Public Doctypes (Doctype definiert den
      HTML-Standart)
  doctype-system"url/zur/DTD" //fuer system Doctype deklaration
  <!--Template fuer die Definition der Transformation-->
</xsl:stylesheet>
```

Listing 3.1: Definition einer XML-Datei zur Transformation

```
  <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE studis SYSTEM "url/zur/DTD">
```

Listing 3.2: Transformierte XML-Datei

Transfoframtionsvorschriften in Form von Templates (Schablonen)

- Templates werden nacheinander notiert, d.h. sie können nicht geschachtelt werden.
- Templates ersetzen irgendwelche Knoten (Tags und Attribute) aus der Quelldatei.

```
<xsl:template match="XPath-Ausdruck">
2 <!--(wohlgeformte) Ausgabe des des Templates, also Text, Tags(inklusive
      Attribute) und  weitere Verarbeitungsanweisungen-->
</xsl:template>
```

Listing 3.3: Syntax einer xsl:template-Deklaration

KAPITEL 4

XPath

4.1 XPath Baumstruktur

Sprache zur Auswahl bestimmter Knoten eines XML-Dokuments. Meist relativ zur aktuellen Position im XML-Dokument.

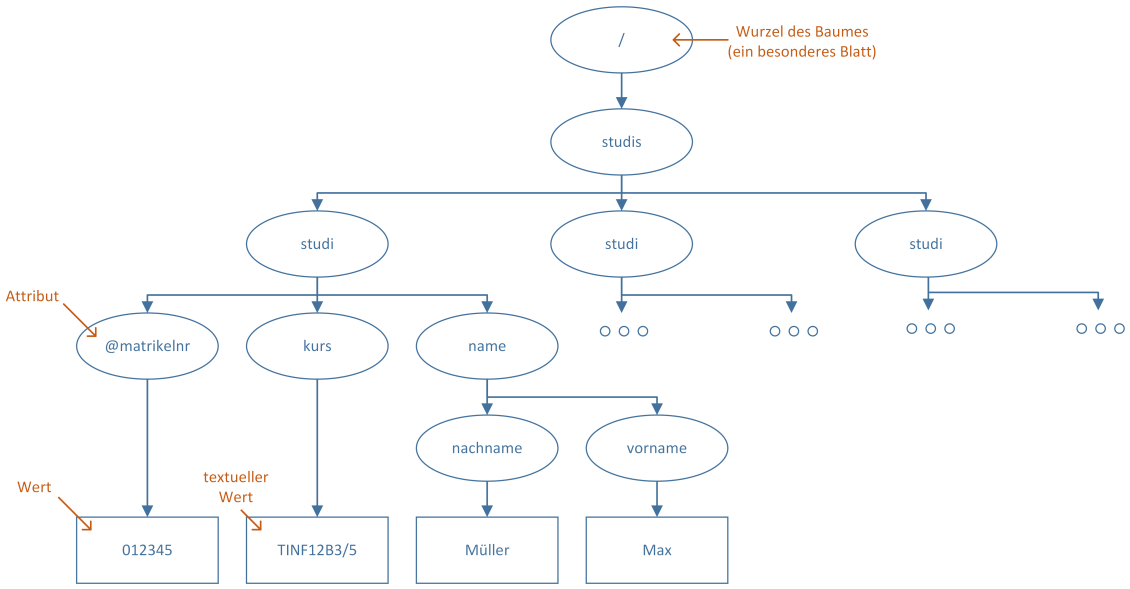

Abbildung 4.1: Baumdarstellung einer XML-Datei

Beschreibung der Knotensyntax:
„.“ → aktueller Knoten
„..“ → Elternknoten

„tagname“ → Kindelement mit ”tagname”
„@attrname“ → Attribut mit ättrname”
„text()“ → Textknoten
„/“ → Wurzel
„//“ → irgendwo im Baum
mehrere Lokalisierungsschritte werden durch „/“ verbunden nacheinander angegeben. Bsp.: /studis/studi/name/nachname/text()

XPath Ausdrücke liefern im allgemeinen eine Knotenmenge, d.h. mehrere Knoten (oder auch keinen)

4.2 Lokalisierungsschritte

bisher: „verkürzte Notation“
außerdem: ausführliche Notation
axis::nodetest[predicate]
(predicate ist optional)

4.2.1 Achsenausdrücke (ausführliche Notation)

root	Wurzelknoten	„/“
child	Kindknoten	„/“ (nicht am Anfang bzw. weglassen)
parent	Elternknoten	„..“
self	aktueller Knoten (Kontextknoten)	„.“
ancestor	Vorfahren, übergeordnete Knoten (Eltern, Großeltern,...)	
descendent	Nachkommen, untergeordnete Knoten (Kinder mit Kindeskinder)	
ancestor-or-self	Vorfahren inkl. Kontextknoten	
descendent-or-self	Nachkommen inkl. Kontextknoten	
following	nachfolgende Knoten (ohne Kinder und Kindeskinder des Kontextknotens)	
following-sibling	nachfolgende Geschwisterknoten (d.h. nachfolgende Knoten mit demselben Elternknoten wie der Kontextknoten)	
preceding	vorhergehende Knoten	
preceding-sibling	vorhergehende Geschwisterknoten (d.h. vorhergehende Knoten mit demselben Elternknoten wie der Kontextknoten)	
attribute	Attributknoten	„@“

4.2.2 Knotentest

- Knotenname/tagname/attrname
- „*“ als Joker für beliebige Knotennamen
- text(), comment() für Text- bzw. Kommentarknoten

4.2.3 Prädikate

Prädikate stehen immer in eckigen Klammern: „[Prädikatausdruck]“

- Zahl: Nummer des Knotens, Nummerierung beginnt bei 1
- Vergleich: z.B. = [@farbe = ”blau”] weitere: !=,>,<,>=,<=
- numerische Operatoren: +,-,*,div,mod (alles Ganzzahloperatoren)
- knotenmengen Funktionen: count (...) Anzahl der Elemente

4.3 XSL-Template Definition und Verwendung

Das Listing 4.1 zeigt ein Beispiel zur tabellarischenDdarstellung aller in der XML gespeicherten Studierenden jeweils mit Vor- und Nachname.

```
<?xml version="1.0"?>
<xsl:template match="studis">
  <html>
    <head>
      <title>Studis an der DHBW</title>
    </head>
    <body>
      <table><xsl:applytemplates /></table>
    </body>
  </html>
</xsl:template>

<xsl:template match="studi">
  <tr>
    <td><xsl:value-of select="name/nachname/text()" /></td>
    <td><xsl:value-of select="name/vorname" /></td>
  </tr>
</xsl:template>
```

Listing 4.1: Praktisches Beispiel für xsl:template

xsl:apply-templates Syntax

```
<xsl:apply-templates select="XPath-Ausdruck" />
```

Listing 4.2: xsl:apply-templates Syntax

- sucht abhängig vom Kontextknoten nach weiteren passenden Templates und führt diese aus (für Kindelemente, kann weiter eingeschränkt und auch ausgeweitet werden über optionales select-Attribut mit XPath-Ausdruck)
- rekursiver Aufruf

xsl:value-of Syntax

```
<xsl:value-of select="XPath-Ausdruck" />
```

Listing 4.3: xsl:value-of Syntax

liefert den textuellen Wert eines Knotens bzw. einer Knotenmenge zurück textueller Wert:

- eines Textknotens ⇒ Text selbst eines Attributknotens ⇒ Wert des anhängenden Textknotens

- eines Elementknotens (eines „Tags“) ⇒ Konkanetation der Werte aller Elemente und Textknoten, welche Kinder des Elementknotens sind

xsl:for-each Syntax

```
<xsl:for-each select="XPath-Ausdruck">
  <!-- Block, welcher fuer jeden Knoten in der durch den XPath-Ausdruck
       bestimmten Knotenmenge ausgegeben wird - relative Position im
       Dokumentbaum entspricht diesem Knoten -->
</xsl:for-each>
```

Listing 4.4: xsl:for-each Syntax

- Schleife → iterativer Aufruf

xsl:if Syntax

```
<xsl:if select="XPath-Ausdruck">
  <!-- Block, welcher ausgegeben wird, falls Bedingung wahr ist -->
  <!-- Achtung: Es gibt kein else! -->
</xsl:if>
```

Listing 4.5: xsl:if Syntax

Bedingung: Wie bei Prädikaten
z.B. einfacher XPath-Ausdruck ⇒ wahr, falls Knotenmenge nicht leer.
„echte“ Bedingungen, z.B. xpath1 &lt xpath2 (oder &gt oder = oder ! = oder &lt= oder &gt=).

xsl:choose Syntax <xsl:choose> ist ähnlich dem switch-case normaler Programmiersprachen.

```
<xsl:choose>
  <xsl:when test="bedingung1">
    <!-- Ausgabe falls Bedingung1 wahr ergibt -->
  </xsl:when>
  <xsl:when test="bedingung2"> ..... </xsl:when>
  <!-- beliebig viele xsl:when moeglich -->

  <xsl:otherwise>
    <!-- Ausgabe, falls keine der vorigen Bedingungen wahr ergibt.
    Otherwise ist Optional -->
  </xsl:otherwise>
</xsl:choose>
```

Listing 4.6: xsl:choose Syntax

xsl:sort Syntax Innerhalb von xsl:apply-templates¿ und ¡xsl:for-each¿

```
<xsl:sort select="XPath" (oder="descending", default=ascending) [
    data-type="number" (default=text)] />
```

Listing 4.7: xsl:sort Syntax

<xsl:sort> ist das erste Kind von <xsl:for-each> (bzw. einziges Kind von <xsl:apply-templates>) auch mehrere <xsl:sort> unmittelbar hinter einander sind möglich für Mehrfachsortierung (bei Gleichheit des vorigen Sortierkriteriums relevant)

Benamte xsl:template Syntax Innerhalb von <xsl:apply-templates> und <xsl:for-each>

```
<xsl:template name="bla">
  <!-- irgendeine Ausgabe -->
</xsl:template>
```

Listing 4.8: xsl:template mit name Syntax

Diese benamten Templates können überall (in jedem Template) aufgerufen werden. Die Syntax dazu ist in Listing 4.9 aufgeführt.

```
<xsl:call-template name="bla" />
```

Listing 4.9: xsl:call-template Syntax

normale Templates

```
<xsl:template match="XPath" [mode="fasel"]>

</xsl:template>
```

Listing 4.10: Normale xsl:template Syntax

- werden per <xsl:apply-templates> nur dann aufgerufen, wenn dieses ebenfalls ein mode-Attribut mit demselben Wert hat!
- damit sind mehrfache Templates für denselben Knoten und damit auch mehrfache Durchläufe durch den Knotenbaum mit verschiedenen Ausgaben möglich

Variable?

```
<xsl:variable name="meine_var" select="wert" />
```

Listing 4.11: xsl:variable Syntax

- Der Variablenwert kann nur einmal gesetzt und nicht mehr verändert werden
- Wert der Variablen kann in jedem XPath-Ausdruck mit $meine_var verwendet werden
- Verwendung z.B. zum „Zwischenspeichern“ eines Wertes abhängig von der Knotenposition und Wiederverwenden auch dann, wenn die Knotenposition verändert wurde

```
<bla wert="fasel">
  <blubb wert="fas" />
  <blubb werd="el" />
  <blubb wert="fasel" />
</bla>
```

Listing 4.12: Beispiel für Variablendeklaration

Bearbeite nur die Knoten blubb, deren Attribut „wert“ denselben Wert hat wie das Attribut „wert“ des Elternknotens bla!

```
<xsl:template match="bla">
  <xsl:variable name="blawert" select="@wert" />
  <xsl:for-each select="blubb[@wert=$blawert]">
    <!-- Wir haben einen blubb-Knoten mit identischem Wert wie der
        bla-Knoten! -->
  </xsl:for-each>
</xsl:template>
```

Listing 4.13: Beispiel für Variable in Template

Ausgabe ins Zieldokument:

- direkte Übernahme von Tags, Texten, Werten von Attributen ins Zieldokument, wie im Template notiert
  aber: Whitespace wird auf ein Trennzeichen reduziert!
- Text innerhalb von <xsl:text> ⇒ Whitespace bleibt erhalten

```
<xsl:template match="bla">
  <fuba>
    <xsl:attribute name="automobil">
      <xsl:value-of select="@wert" />
    </xsl:attribute>
  </fuba>
</xsl:template>
```

Listing 4.14: Beispiel für Variable in Template

xsl:attribute Syntax

```
<xsl:attribute name="aname">
  <!--Wert des Attributs -->
</xsl:attribute>
```

Listing 4.15: xsl:attribute Syntax

- Gibt dem unmittelbar vorher erzeugten Elemntknoten (und noch offenem Knoten) einen Attributknoten „@aname“ mit Wert „Wert des Attributs“

4.4 Übung

http://dh.jroethig.de → WuMMbasIS
⇒XML und XSLT

KAPITEL 5

Grafikformate im Web

Rastergrafiken	**Vektorgrafiken**
Aufteilung der Darstellungsfläche in meist gleichgroße und quadratische Teilflächen.	Etablieren eines Koordinatensystems auf der Darstellungsfläche
Zuweisung (Attributierung) von Farbwerten, Helligkeitswerten, Transparenzwerten zu den einzelnen Teilflächen.	Darstellung von geometrischen Grundformen.
+ einfache Darstellung auf herkömmlichen rasterbasierten Ausgabegeräten (LCD, Laser, Tintenstrahldrucker)	Attributierung der Grundformen mit Position, Größe, Farbe, Helligkeit, Transparenz, Muster, Strichdaten, ... (letztere optional)
+ einfache Übernahme von natürlichen Darstellungen möglich	+ nahezu unbegrenzte Skalierbarkeit
− Skalierbarkeit schlecht: Vergrößerung des Rasters	+ auch vektorbasierte Ausgabegeräte verfügbar: Plotter, vektorbasierte CRT-Bildschirme
	− Rendering ist recht aufwändig

5.1 Rasterbasierte Grafikformate

Zuweisung von Farbe → Farbmodell: RGB (Rot+Grün+Blau)
Helligkeit ergibt sich durch die Farbwerte: größere Werte → mehr Helligkeit.
Üblicherweise wird jedem Farbanteil ein 8bit-Wert zugewiesen ⇒ ein Bildpunkt benötigt $3 * 8bit = 3Byte$
Bsp: Digitalkamera mit 25Megapixel ⇒ 75MByte Speichervolumen pro Bild, 16GByte Speicherkarte ⇒ „nur“ 200 Bilder ⇒ Kompression der Speicherdaten!!!

5.1.1 GIF

Graphics Interchange Format wurde in den 1980er Jahren erfunden und verwendet das RGB-Farbmodell. Das Format ist für computergenerierte schematische Darstellung mit großen einfarbigen Flächen gut geeignet.

Manko zu Beginn wird eine Farbtabelle mit maximal 256 Farben definiert und fortan jede Teilfläche nur noch mit einem 8bit Wert als Index auf diese Tabelle kodiert

- ⇒ Beschreibung der gleichzeitig verwendeten Farbenzahl auf 256 Farben (aus insgesamt ≈ 16Mio möglichen Farben)
- ⇒ Reduktion der Datenmenge durch Verlust an Information (welcher klar sichtbar ist). Anschließende Kompression der Folge von Indexwerten: modifizierte Lauflängenkodierung (Wert und Anzahl)
- ⇒ funktioniert bei häufig aufeinanderfolgenden Werten
- ⇒ verlustfreies Kompressionsverfahren bei GIF!!!
- ⇒ aber Voraussetzung für das Funktionieren des Kompressionsverfahrens ist die vorherige Reduktion auf 256 Farbwerte
- ⇒ Wahrscheinlichkeit gleicher aufeinanderfolgender Werte damit recht hoch!

Transparenz eine Farbe kann als transparent markiert werden, jedoch ist keine Teiltransparenz möglich.

Animated GIF erlaubt kurze Sequenzen von Bildern, welche unmittelbar nacheinander dargestellt werden ⇒ „Filmeffekt“

5.1.2 JPEG

Joint Photographic Expert Group verwendet das RGB-Farbmodell oder (alternativ) CMYK (für Druck)

- ⇒ keine Transparenz möglich
- ⇒ keine Reduktion der Farbanzahl, also 16Mio. (bei RGB) bzw. 4Mrd. (bei CMYK) Farben gleichzeitig (theoretisch) möglich

DCT: Discrete Cosine Transform

a) Mittelwert über gesamtes (zunächst 8x8 Pixel) Areal

b) Halbierung des (8x8 Pixel-) Blockes in x- und y-Richtung auf vier (4x4 Pixel-) Blöcke

c) Berechnung der Mittelwerte dieser kleineren Blöcke

d) Abspeichern als Differenzen der kleinen Blöcke zum großen Block

e) weiter bis 1x1 Pixel-Block erreicht

DCT sorgt für denselben Informationsgehalt bei gleicher Datenmenge (keine Kompression!)
„Qualitätseinstellung“ durch Zusammenfassen von Koeffizienten mit ähnlichen Werten auf den gleichen Wert ⇒ Informationsverlust, hilft der Effizienz des nachfolgenden Huffman

Huffman-Codierung: verlustfrei abhängig von der Häufigkeit ihres Vorkommens werden die zu kodierenden Werte mit kürzeren oder längeren Codewerten dargestellt ⇒ variabel lange Codeworte für die Werte!
Prinzip: Binärbaum
Beispiel:

0	⇒	fertig
1	⇒	kommt noch was
„0“	⇒	für häufigsten Wert
„10“	⇒	für zweithäufigsten Wert
„110“	⇒	für dritthäufigsten Wert
bis zu „11... 10“ (255 Einsen)	⇒	für seltensten Werte

⇒funktioniert ganz gut bei DCT-Koeffizienten („Abweichungswerten“), da die Abweichungen häufig kleine Werte darstellen ⇒ kurze Codelänge für diese kleinen Werte
⇒verlustfreie Komprimierung!

Nachfolgeformat: JPEG2000

⇒ unterstützt RGBA (zusätzlicher Alpha-Kanal/Transparenz mit 8bit)

⇒ nur mit Lizenzgebühren zu verwenden

⇒ keine weite Verbreitung bislang

5.1.3 PNG

Portable Network Graphics wurde in den 1990er Jahren als Ersatz für GIF entwickelt (wg. Patentproblematik)

2 Modi:

- indizierte Farben wie bei GIF
- unreduzierte RGBA-Farben

⇒beliebige (auch Teil-)Transparenz möglich. In jedem Fall verlustfreie Kompression nachgeschaltet.
⇒für photorealistische Darstellungen of deutlich größeres Dateivolumen als bei JPEG! Dafür evtl. JNG → keine große Verbreitung.

5.2 Vektorgrafikformate

5.2.1 SVG

Das Scalable Vector Graphics Format gibt es seit deren Einführung im Jahre 1998. Dabei gab es zwei unterschiedliche Vorschläge für das Vektorgrafikformat:

- VML (Vector Markup Language) von Microsoft
- PGML (Precision Graphics Markup Language) von Adobe, IBM, NS/Netscape, Sun

Historie

10/1998: Entwurf für Anforderungen an VektorGrafikformat SVG
09/2001: SVG 1.0 als W3C-Standard
01/2003: SVG 1.1 mit errata, modularization
04/2005: SVG 1.2 verschiedene Profile („tiny“, „full“, „basic“). Bis auf „tiny“ wieder zurückgegangen derzeit Arbeit an SVG2 ...

SVG ist XML-basiert und benutzt als übliche File-Extension „.svg“.
MIME-Type: image/svg+xml

SVG Viewer

früher: Adobe SVG Viewer
heute: native Unterstützung in allen gängigen Browsern (Mozilla Firefox, Google Chrome, Apple Safari, Opera Browser, Microsoft Internet Explorer)

SVG Editor

Texteditor
XML-Editor
Inkscape

```
<?xml version="1.0" [encoding="UTF-8"]?>
<!DOCTYPE
    svg <!--root-Tag-->
    PUBLIC <!--entspr. standardisierte Sprache-->
    "-//W3C//DTD SVG 1.1//EN" <!--Public Identifier-->
    "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd" <!--
        System-Identifier-->
>
<svg
    xmlns="http://www.w3.org/2000/svg"
    [xmlns:xlink="http://www.w3.org/1999/xlink"]
    width="breite" height="hoehe" [viewBox="xlo ylo kbreite khoehe"]
>
  <title>Name der Grafik</title>
  <desc>Beschreibung als Text</desc>
  <defs>
    <!--Platz fuer CSS-Definition, Javascript-Code, Definitionen von
        graph Objekten, welche zunaechst nicht angezeigt werden-->
  </defs>
  <!--eigentlicher Inhalt in Form von graph. Grundformen und unter
      Benutzung von komplexen (auch ind <defs> definierten) graph.
      Objekten-->
```

Listing 5.1: Grundgerüst einer SVG-Datei

breite, hoehe: dimensionsbehaftete Angabe (z.b. 30mm, 500px) dimmensionslose Angabe äquivalent zu Pixel 3000≙3000px
⇒durch breite und hoehe wird auch das Koordinatensystem aufgespannt...

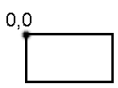

Abbildung 5.1: implizites Koordinatensystem in SVG durch Breiten- bzw. Höhenangaben

...wenn nicht auch das Attribut viewBox verwendet wird: viewBox=„xlo ylo kbreite khoehe“

Grafische Grundformen

Tags mit Attributen für Koordinaten und/oder Größe

```
<line x1="..." y1="..." x2="..." y2="..." />
<!--Linie von Punkt(x1,y1) zu Punkt(x2,y2)-->

<circle cx="..." cy="..." r="..." />
<!--Kreis mit Mittelpunkt(cx,cy) und Radius r-->

<rect x="..." y="..." width="..." height="..." />
<!--lotrechtes Rechteck mit Eckpunkt(x,y), Breite width und Hoehe height
    -->

<ellipse cx="..." cy="..." rx="..." ry="..." />
<!--lotrechte Ellipse mit Mittelpunkt(cx,cy) und horizontalen bzw.
    vertikalen Radius rx bzw. ry-->

<polyline points="x1 y1 x2 y2 x3 y3 ..." />
<!--Streckenzug, welcher die Punkte(x1,y1),(x2,y2),(x3,y3),...
    verbindet, deren Koordinaten als paarweise Werte in einem Attribut
    angegeben werden-->
<!--Koordinaten werden durch Leerzeichen und/oder Komma getrennt.
    Sinnvoll ist es jeweils aufeinanderfolgende x- und y-Koordinate
    desselben Punktes durch Komma und die Koordinaten des naechsten
    Punktes davon durch Leerzeichen zu trennen-->

<polygon points="..." />
<!--geschlossener Streckenzug, bis der letzte Punkt wieder mit dem
    ersten verbunden wird-->
```

Listing 5.2: Syntax von grafischen Grundformen in SVG

bislang fehlen uns „runde Kurven“ bzw. „Kurvenzug“
Voraussezung: Funktion stetig und Funktion stetig differenzierbar d.h. die erste Ableitung=Steigung muss ebenfalls stetig sein.
Lösung: abschnittsweise Definition über (perse) stetig differenzierbare Funktionen; an den Schnittstelllen muss jeweils die Steigung der linken gleich der Steigung der rechten Kurve sein.
⇒Kurveninterpolation
bei SVG: Kurveninterpolation mittels „Bezierkurven“

Bezierkurven

quadratische Bezierkurve geometrische Herleitung von quadratischen Bezierkurven: quadr.Bez.-Kurve wird vollständig über drei Punkte definiert:

- Anfangspunkt (AP)
- Stützpunkt (SP)

- Endpunkt (EP)

Eigenschaften: Die Kurve beginnt im Anfangspunkt und endet im Endpunkt. Der Stützpunkt liegt im allgemeinen nicht auf der Kurve.
Kurve hat im AP die Steigung der Geraden AP-SP und im EP die Steigung der Geraden SP-EP.

kubische Bezierkurve AP-SP1-SP2-EP
beginnt in AP und endet in EP, Steigung in AP $\hat{=}$ Steigung AP-SP1, Steigung in EP $\hat{=}$ Steigung SP2-EP.

De-Casteljau-Algorithmus allgemeiner Tag zum Zeichnen von u.a. auch Bezier-Kurven:

```
<path d="...." />
```

Listing 5.3: Path-Tag

Das d- (wie „data“)-Attribut enthält eine Art „Zeichenbefehle“, mit denen das Ergebnis des path-Tags spezifiziert wird, sowie die entsprechenden (Punkt-)Koordinaten.

Zeichenbefehle: 1Buchstaben, gefolgt von den zugehörigen Koordinaten
→Großbuchstaben: absolute Koordinaten
→Kleinbuchstaben: relative Koordinaten, d.h. relativ zur aktuellen (letzten) Position
⇒Wir bewegen uns während des Abarbeitens des path-Tags über die Darstellungsfläche.
⇒zu Beginn des path-Tags befinden wir uns um Ursprung (bei (0,0)).
„M“ („More“)⇒bewegt den Zeichenstift, ohne etwas zu zeichnen

```
<path d="M10,50 m30,-20" />
```

Listing 5.4: Beispiel für ”Mïm Path-Tag

Im Beispiel aus Listing 5.4 ändert sich die Position zu (40,30), ohne etwas zu zeichnen.
„L“ („Line“)⇒zeichnet eine Strecke vom aktuellen zum neuen Punkt

```
<path d="M10,50 m30,-20 L70,20 m10,10 l-20,-50" />
```

Listing 5.5: Beispiel für ”Lïm Path-Tag

Das Baispiellisting 5.7 zeichnet eine Strecke (40,30)-(70,20) und eine Strecke (80,30)-(60,-20)

$Mx1y1$	$\Rightarrow$	$x_a = x1, y_a = y1$
$mdxdy$	$\Rightarrow$	$x_a = x_a + dx, y_a = y_a + dy$
$Lx1y1$	$\Rightarrow$	zeichnet Strecke von (x_a, y_a) zu (x1,y1) und verändert $x_a = x1, y_a = y1$
$ldxdy$	$\Rightarrow$	zeichnet Strecke von x_a, y_a nach $(x_a + dx, y_a + dy)$ und verändert $x_a = x_a + dx, y_a = y_a + dy$
$Hx1$	$\Rightarrow$	zeichnet horizontale Strecke von $(x_a, y_a$ nach $(x1, y_a)$ und verändert $x_a = x1, y_a = y_a$ h dx $\Rightarrow$Strecke von $(x_a, y_a) - (x_a + dx, y_a)$ und $x_a = x_a + dx, y_a = y_a$
$Vy1\vee$ v dy	$\Rightarrow$	zeichnet eine vertikale Strecke entsprechend

```
  <polyline points="30,20 50,10 70,70" />
2 <path d="M30,20 L50,10 L70,70" />
  <!--oder-->
4 <path d="M30,20 l20,-10 l20,60" />
```

Listing 5.6: Beispiel für "Lïm Path-Tag

Q sx,sy ex,ey	$\Rightarrow$	quadratische B-K mit AP(x_a, y_a), SP(sx, sy), EP(ex, ey)
q sdx,sdy edx,edy	$\Rightarrow$	quadratische B-K mit AP(x_a, y_a), SP$(x_a + sdx, y_a + sdy)$, EP$(x_a + edx, y_a + edy)$

Achtung: „Während" der Abarbeitung des einzelnen Befehls bleibt (x_a, y_a) auf dem Anfangspunkt.

(s1x,s1y s2x,s2y ex,ey)	$\Rightarrow$	kubische B-K mit AP(x_a, y_a), SP1$(s1x, s1y)$, SP2$(s2x, s2y)$, EP(ex, ey)
c s1dx,s1dy s2dx,s2dy ex,ey	$\Rightarrow$	kubische Bezier-Kurve entsprechend mit relativen Koordinaten
T ex,ey	$\Rightarrow$	quadratische Bezier-Kurve mit von der vorherigen B-K an AP gespiegelten SP
S s2x,s2y ex,ey	$\Rightarrow$	kubische B-K mit von der vorigen kubischen B-K an AP zu SP1 gespiegelten SP2

$\Rightarrow$sinnvoll (nur) bei voriger quadr. bzw. kubischer Bezier-Kurve
falls nicht$\Rightarrow$SVG-Spezifikation gibt an, wie der zu spiegelnde Punkt zu bestimmen ist

A,a:„elliptical arc"	$\Rightarrow$zeichnet Ausschnitt aus Ellipsen $\Rightarrow$relativ schwer zu verwenden (Parameter selbst nachlesen!)
Z,z:	schließt den aktuellen Streckenzug mit einer Strecke zum entsprechenden ersten Punkt (nach dem letzten M-Befehl)

```
  <path d="M10,10 L20,10 L15,20 Z M60,30 L70,40 L50,60 Z" />
2 <!-- Z=zurueck zu Anfangspunkt -->
```

Listing 5.7: Beispiel zwei Dreiecke mit nur einem Path-Tag

Text

```
<text x="x" y="y">Text der dargestellt wird</text>
```

Listing 5.8: Syntax des Text-Tag

⇒erstellt einen einzeiligen Text, beginnend an der Position (x,y)
⇒Zeichenumbrüche sind zunächst nicht möglich
aber: innerhalb ¡text¿ kann der Text mit ¡tspan¿ logisch unterteilt werden

```
<tspan[x="x" y="y"]>Text, welcher an einer Position (durch die
    optionalen Attribute x und y gegeben) beginnen kann</tspan>
<tspan [dx="dx" dy="dy"]>Text,welcher an einer Position mit Versatz (
    dx,dy) zur letzten Position gesetzt wird<\tspan>
```

Listing 5.9: Syntax des Text-Tag

⇒mehrzeiliger Text mit etwas Aufwand und ungewohnt, aber machbar

```
<text x="0" y="0">
  <tspan>eins</tspan>
  <tspan x="0" dy="10">anderthalb</tspan>
  <tspan x="0" dy="10">zwei</tspan>
  <tspan x="0" dy="10">drei</tspan>
</text>
```

Listing 5.10: Beispiel für ein tspan-Tag

Universalattribute

style: für CSS-Deklarationen (wie bei HTML, allerdings größtenteils andere Properties)

```
<g><!-- beliebige andere Tags fuer grafische Grundelemente, Text, fuer
    weitere Gruppen -->
```

Listing 5.11: Gruppierung von Inhalts-Tags bei SVG

CSS-Properties für SVG

stroke: Strichfarbe ⇒ z.b. „line {stroke: red;}“
stroke: #ABCDEF;
stroke: #AFE; ≙ #AAFFEE

stroke: rgb(0,200,0);

opacity: 0.5; („Deckkraft“)
0: voll-transparent
1: voll-deckend/intransparent

fill: Füllfarbe

opacity bestimmt die Deckkraft gleichermaßen für Strich und Füllung, getrennt mögich über
stroke-opacity:1.0;
fill-opacity:0.2;

stroke-width: Strichdicke
strocke-width:2px;
⇒Strichdicke von 2„Pixeln“=Einheiten im User Coordinate System

Präsentationsattribute

style=ßtroke-width: 2px;”/¿
stroke-width=”2px”
Präsentationsattribute haben niedrigere Priorität als CSS-Angaben! (IMHO ein Design-Bug!)

transform

für affine Abbildungen der entsprechenden Objekte

- Drehung
- Skalierung
- Verschiebung
- ...

Verschiebung

```
transform="translate(dx [dy])"
```

Listing 5.12: Verschiebung mit der transform Syntax

z.B. translate(10 -3)
falls dy fehlt, dy=0 (also horizontale Verschiebung)

Skalierung

```
transform="scale(sx [sy]"
```

Listing 5.13: Skalierung mit der transform Syntax

falls sy fehlt, sy=sx (also maßstabsgetreue Skalierung)

Drehung

```
transform="rotate(winkel [cx cy])"
//winkel in Altgrad (360 Grad = Umdrehung)
//falls cx,cy fehlt dann gilt: cx=cy=0
```

Listing 5.14: Drehung mit der transform Syntax

Winkel in Altgrad (360 Grad = Umdrehung)
falls cx,cy fehlt dann gilt: cx=cy=0

Scherung „Schrägstellung“

```
transform="skewX(winkel)"
//oder
transform="skewY(winkel)"
```

Listing 5.15: Scherung mit der transform Syntax

Anwendung mehrerer affiner Abbildungen nacheinander

```
<g transform="translate(20 10)
              scale(2)
              translate(-20 -10)">
          .......
</g>
```

Listing 5.16: Scherung mit der transform Syntax

⇒ bewirkt zuerst eine Verschiebung um (-20,-10), dann eine Skalierung mit 2 und zuletzt eine Verschiebung um (20,10)

⇒ zuerst auszuführende affine Abbildung wird als letztes notiert

weitere (universale) affine Abbildung Matrix

$$\begin{pmatrix} xneu \\ yneu \\ 1 \end{pmatrix} = \begin{pmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} xorig \\ yorig \\ 1 \end{pmatrix} = \begin{pmatrix} a * xorig + cyorig + e \\ b * xorig + d * yorig + f \\ 0 * xorig + 0 * yorig + 1 \end{pmatrix} = \begin{pmatrix} xorig + dx \\ yorig + dy \\ 1 \end{pmatrix}$$

z.B. translate (dx dy)

$$\begin{pmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{pmatrix}$$

Syntax in SVG:
transform=”matrix(a b c d e f)”
Wir können also statt ”translate(dx dy)äuch schreiben: matrix(1 0 0 1 dx dy)
scale(sx sy) $\hat{=}$ matrix(sx 0 0 sy 0 0)

$$\begin{pmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Skalierung:

$$\begin{pmatrix} sx * xorig \\ sy * yorig \\ 1 \end{pmatrix}$$

ebenso existieruen Matrizen für alle affinen Abbildungen sowie die Konkatenation von mehreren (beliebig vielen) affinen Abbildungen

$$\begin{pmatrix} xneu \\ yneu \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 20 \\ 0 & 1 & 10 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 0 & -20 \\ 0 & 1 & -10 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} xorig \\ yorig \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} * \begin{pmatrix} xorig \\ yorig \\ 1 \end{pmatrix}$$

weiteres „grafisches“ Objekt:

```
<use xlink:href="hashtagkackZeichen anderes Objekt" [x="..." y="..."]/>
```

Listing 5.17: Scherung mit der transform Syntax

x,y: optionale Parameter zur Positionierung (Verschiebung) des verwendeten Objekts
verweist auf Objekt

```
<g id="anderes Objekt">
    .....
</g>
```

Listing 5.18: Scherung mit der transform Syntax

KAPITEL 6

Web-Suchmaschinen

6.1 Entwicklung

0. Kommentierte Linklisten

⇒ Ausnutzen des „Urprinzips“ des Web ⇒ Hyperlink

- hoher Pflegeaufwand
- nur sinnvoll bei kleiner Anzahl WebSites
- potentiell hohe Qualität der Suchergebnisse möglich

1. Webkataloge (Yahoo, 1994)

⇒ Linklisten nach Kategorien sortiert

⇒ hierarchisch geordnet

- hoher Pflegeaufwand
- benutzbar bei deutlich höherer Anzahl an WebSites, aber Grenze durch Pflegeaufwand in den am feinsten spezifischen Unterkategorien

2. Volltextindizierer & WebCrawler (AltaVista, 1995)

⇒ Volltext der WebSite wird nach allen Begriffen in DB aufgenommen und „indiziert“ (ausgenommen Trivial„begriffe“ wie z.B. „ein/eine/einer/der/die/das/...“)

⇒ Seite wird nach links auf andere Seiten durchsucht; andere Seiten werden ebenfalls in den Index aufgenommen

- kein manueller Aufwand (außer initialer Programmierung)

- hoher maschineller Aufwand
  - Rechenzeit
  - Speicherplatz
  - hoher Netzwerktraffic

Anmerkung: AltaVista war ein Demonstrator der Firma DEC für die Leistungsfähigkeit seiner Rechner

Abbildungsverzeichnis

Tabellenverzeichnis

Listings