

## Angewandte Informatik 2 - Tutorium 4

### Besprechung: Übungsblatt 4

Götz Bürkle  
(goetz@buerkle.org)

Institut für Angewandte Informatik und  
Formale Beschreibungsverfahren - AIFB  
Universität Karlsruhe (TH)

KW 25/26

## XML-Schema erstellen - Adresse in Deutschland

Enthalten sein sollen folgende Informationen:

- ▶ genau ein Firmenname
- ▶ optional ein Firmenzusatz (GmbH, AG oder Ltd.)
- ▶ genau eine Straße, es dürfen keine Zahlen vorkommen
- ▶ genau eine Hausnummer, positive Zahl
- ▶ genau eine Postleitzahl (5-stellige positive Zahl)
- ▶ genau eine Stadt (Buchstaben und „-“, wobei links und rechts eines „-“ mindestens ein Buchstabe stehen müssen)
- ▶ optional eine Telefonnummer:
  - Vorwahl (mit 0 beginnende Zahl, zwischen 3 und 5 Zeichen lang),
  - Nummer (nicht mit 0 beginnende Zahl, mindestens 4 Ziffern lang)

## Übersicht

### Heimarbeitsblatt 4

- Aufgabe 1 - XML-Schema
- Aufgabe 2 - XSL/XPath
- Aufgabe 3 - HTTP
- Aufgabe 4 - Webarchitektur
- Aufgabe 5 - Dynamische Webseiten

### Zusammenfassung

## Beispiel

```
<?xml version="1.0" encoding="UTF-8"?>
<adresse>
  <firmenname>Silverstroke</firmenname>
  <firmenzusatz>AG</firmenzusatz>
  <strasse>...</strasse>
  ...
</adresse>
```

## Reguläre Ausdrücke in XML-Schema

Reguläre Ausdrücke beschreiben reguläre Sprachen (Chomsky-3).

### ► Zeichenklassen

[a-z] → Zeichenklasse bestehend aus den Buchstaben a, b, ..., z

### ► Anzahlen

\* → keinmal oder beliebig oft

+ → mindestens einmal

? → keinmal oder einmal

{3, 7} → mindestens 3, höchstens 7 mal

### ► Negation

[^a-z] → Zeichenklasse bestehend nicht aus den Buchstaben a, b, ..., z

## Reguläre Ausdrücke in XML-Schema

### ► Maskierung

Zeichen mit besonderer Bedeutung müssen „maskiert“ werden, wenn sie als Zeichen und nicht mit ihrer Bedeutung benutzt werden.

Solche Zeichen sind z.B. ^, \$, +, \*, ?, [, ], (, ), {, }, ., \

Maskiert wird, indem ein „\“ vorangestellt wird (z.B. \]).

Mehr zu Regulären Ausdrücken:

<http://de.selfhtml.org/perl/sprache/regexpr.htm>

<http://www.lrz-muenchen.de/services/schulung/unterlagen/regul/>

<http://www.xml-xslt.de/index.php?id=146,0,0,1,0,0>

[http://de.wikipedia.org/wiki/Regulärer\\_Ausdruck](http://de.wikipedia.org/wiki/Regulärer_Ausdruck)

## XML-Schema - Adresse

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="adresse" type="adresseType"/>
  <xsd:complexType name="adresseType">
    <xsd:all>
      <xsd:element name="firmenname" type="xsd:string"/>
      <xsd:element name="firmenzusatz" type="zusatzType"
        minOccurs="0"/>
      <xsd:element name="strasse" type="strasseType"/>
      <xsd:element name="hausnummer" type="xsd:positiveInteger"/>
      <xsd:element name="postleitzahl" type="postleitzahlType"/>
      <xsd:element name="stadt" type="stadtType"/>
      <xsd:element name="telefonnummer" type="telefonnummerType"
        minOccurs="0"/>
    </xsd:all>
  </xsd:complexType>
  [...]
</xsd:schema>
```

## XML-Schema - Adresse

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema [...]
  <xsd:complexType name="telefonnummerType">
    <xsd:all>
      <xsd:element name="vorwahl" type="vorwahlType"/>
      <xsd:element name="nummer" type="nummerType"/>
    </xsd:all>
  </xsd:complexType>
  <xsd:simpleType name="zusatzType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="GmbH"/>
      <xsd:enumeration value="AG"/>
      <xsd:enumeration value="Ltd."/>
    </xsd:restriction>
  </xsd:simpleType>
  [...]
</xsd:schema>
```

## XML-Schema - Adresse

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema [...]
  <xsd:simpleType name="strasseType">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[^\0-9]+"\>
        <!-- - <xsd:pattern value="\D"/> - -->
      </xsd:restriction>
    </xsd:simpleType>
  <xsd:simpleType name="postleitzahlType">
    <xsd:restriction base="xsd:positiveInteger">
      <xsd:totalDigits value="5"/>
      <!-- - <xsd:pattern value="[0-9]{5}"/>,
           dann aber restriction base="xsd:string"! - -->
    </xsd:restriction>
  </xsd:simpleType>
  [...]
</xsd:schema>
```

## XSL-Stylesheet erstellen

Gegeben: XML-Datei (Personenverzeichnis)

### Was ist zu tun?

- ▶ Transformation XML → HTML
- ▶ HTML-Datei mit vier Spalten  
(Name, Vorname, Raumnummer, Telefonnummer)
- ▶ Pro Person eine Zeile
- ▶ Raumnummern, die mit 2 beginnen oben  
(Raumnummern sind dreistellig)

## XML-Schema - Adresse

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema [...]
  <xsd:simpleType name="stadtType">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[A-Z][a-z]*(\-[A-Z][a-z]*)*\>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="vorwahlType">
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="5"/>
      <xsd:pattern value="0[1-9][0-9]+"\>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="nummerType">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[1-9][0-9][0-9][0-9]+"\>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

## Beispiel: personenverzeichnis.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<personenverzeichnis>
  <person>
    <name>Bonn</name>
    <vorname>Matthias</vorname>
    <raumnummer>-121</raumnummer>
    <telefonnummer>+49 (721) 608 6034</telefonnummer>
  </person>
  [...]
  <person>
    <name>Richter</name>
    <vorname>Urban</vorname>
    <raumnummer>225</raumnummer>
    <telefonnummer>+49 (721) 608 6586</telefonnummer>
  </person>
</personenverzeichnis>
```

## personenverzeichnis.xslt, Teil 1

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <head>
        <title>Personenverzeichnis</title>
      </head>
      <body>
        <h1>Personenverzeichnis</h1>
        <table border="1">
          <tr>
            <th>Name</th>
            <th>Vorname</th>
            <th>Raumnummer</th>
            <th>Telefonnummer</th>
          </tr>
          [...]
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

## personenverzeichnis.xslt, Teil 3

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet [...]
  <xsl:template match="/">
    [...]
    <xsl:for-each select="personenverzeichnis/person
      [raumnummer > 199 and raumnummer < 300]">
      <xsl:sort select="raumnummer"
        order="ascending" data-type="number"/>
      <tr>
        <td><xsl:value-of select="name"/></td>
        <td><xsl:value-of select="vorname"/></td>
        <td><xsl:value-of select="raumnummer"/></td>
        <td><xsl:value-of select="telefonnummer"/></td>
      </tr>
    </xsl:for-each>
    [...]
  </xsl:template>
</xsl:stylesheet>
```

## personenverzeichnis.xslt, Teil 2

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet [...]
  <xsl:template match="/">
    [...]
    <!-- Schleifenanweisung/Wiederholungseffekt -->
    <!-- Pfadname und Sortierbedingung mit XPath
      groesser 199; kleiner 300 -->
    <xsl:for-each select="personenverzeichnis/person
      [raumnummer > 199 and raumnummer < 300]">
      <!-- aufsteigende Sortierung nach raumnummer -->
      <xsl:sort select="raumnummer"
        order="ascending" data-type="number"/>
      <!-- Alternative XPath-Anfrage -->
      <!-- <xsl:for-each select="personenverzeichnis/person
        [starts-with(raumnummer,2)]"> -->
      <!-- <xsl:sort select="raumnummer"
        order="ascending" /> -->
      [...]
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

## personenverzeichnis.xslt, Teil 4

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet [...]
  <xsl:template match="/">
    [...]
    <!-- und jetzt alle anderen Personen;
      kleiner 200; groesser 299 -->
    <xsl:for-each select="personenverzeichnis/person
      [raumnummer < 200 or raumnummer > 299]">
      <!-- aufsteigende Sortierung nach raumnummer -->
      <xsl:sort select="raumnummer"
        order="ascending" data-type="number"/>
      <!-- Alternative XPath-Anfrage -->
      <!-- <xsl:for-each
        select="personenverzeichnis/person
          [not(starts-with(raumnummer,2))]"> -->
      <!-- <xsl:sort select="raumnummer"
        order="ascending" /> -->
      [...]
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

## personenverzeichnis.xslt, Teil 5

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet [...]
  <xsl:template match="/">
    [...]
    <xsl:for-each select="personenverzeichnis/person
      [raumnummer &lt; 200 or raumnummer &gt; 299]">
      <xsl:sort select="raumnummer"
        order="ascending" data-type="number"/>
      <tr>
        <td><xsl:value-of select="name"/></td>
        <td><xsl:value-of select="vorname"/></td>
        <td><xsl:value-of select="raumnummer"/></td>
        <td><xsl:value-of select="telefonnummer"/></td>
      </tr>
    </xsl:for-each>
    [...]
  </xsl:template>
</xsl:stylesheet>
```

## Anmerkungen

Es gibt natürlich verschiedene Möglichkeiten XPath-Ausdrücke zu formulieren.

Auf das Erzeugen von perfektem (also zumindest validem) HTML-Code wurde hier bewußt verzichtet, z.B. fehlt die DOCTYPE-Angabe.

Siehe auch in SELFHTML:

- ▶ <http://de.selfhtml.org/xml/darstellung/xpathsyntax.htm>
- ▶ <http://de.selfhtml.org/xml/darstellung/xsltbeispiele.htm>
- ▶ <http://de.selfhtml.org/xml/darstellung/xsltelemente.htm>

## personenverzeichnis.xslt, Teil 6

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet [...]
  <xsl:template match="/">
    <html>
      [...]
      <body>
        <table>
          [...]
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

## Laden von Dateien über HTTP

**Wieviel Zeit kann bei komplettem Download durch Einsatz von Pipelining und Keep-Alive eingespart werden?**

- ▶ **Dateien:**
  - ▶ eine HTML-Datei
  - ▶ zwei eingebundene Bilder
  - ▶ eine Flash-Animation
- ▶ **Größen:**
  - ▶ HTML: 3 kB
  - ▶ Bilder: 10 kB
  - ▶ Flash: 80 kB

## Laden von Dateien über HTTP

Weitere Annahmen:

- ▶ **Overhead:** pro Request und pro Response: 250 Byte
- ▶ **Nettobandbreite:**
  - ▶ 760 kBit/s Downstream
  - ▶ 120 kBit/s Upstream
- ▶ **TCP-Verbindungsaufbau:** 150 ms
- ▶ **Datenoverhead** der Transport- und Vermittlungsprotokolle können vernachlässigt werden
- ▶ **Genauigkeit:** in ganzen Millisekunden rechnen

## Berechnung - Übertragungsdauern

- ▶  $\text{httpRequest} = \frac{2000 \text{ Bit}}{120000 \frac{\text{Bit}}{\text{s}}} \cdot 1000 \frac{\text{ms}}{\text{s}} = 17 \text{ ms}$
- ▶  $\text{htmlResponse} = \frac{26576 \text{ Bit}}{760000 \frac{\text{Bit}}{\text{s}}} \cdot 1000 \frac{\text{ms}}{\text{s}} = 58 \text{ ms}$
- ▶  $\text{imgResponse} = \frac{83920 \text{ Bit}}{760000 \frac{\text{Bit}}{\text{s}}} \cdot 1000 \frac{\text{ms}}{\text{s}} = 110 \text{ ms}$
- ▶  $\text{flashResponse} = \frac{657360 \text{ Bit}}{760000 \frac{\text{Bit}}{\text{s}}} \cdot 1000 \frac{\text{ms}}{\text{s}} = 865 \text{ ms}$
- ▶ TCP-Aufbau = 150 ms

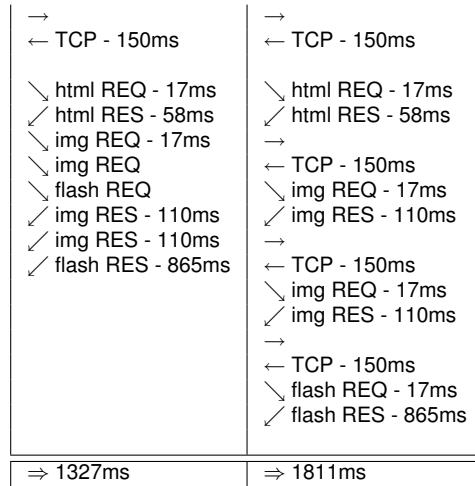
## Berechnung - Datenmengen bestimmen

- ▶  $\text{httpRequest} = 250 \text{ B} = 2000 \text{ Bit}$   
(Request besteht nur aus Overhead)
- ▶  $\text{htmlResponse} = 3 \text{ kb} = 3072 \text{ B} = 24576 \text{ Bit}$   
+ Overhead = 24576 Bit + 2000 Bit = 26576 Bit
- ▶  $\text{imgResponse} = 10 \text{ kB} = 10240 \text{ B} = 81920 \text{ Bit}$   
+ Overhead = 81920 Bit + 2000 Bit = 83920 Bit
- ▶  $\text{flashResponse} = 80 \text{ kB} = 81920 \text{ B} = 655360 \text{ Bit}$   
+ Overhead = 655360 Bit + 2000 Bit = 657360 Bit

## Sequenzdiagramm

...

## Sequenzdiagramm - schematisch



## Zu beachten

- ▶ Umrechnungsfaktoren: bei Byte 1024, bei Bit 1000
- ▶ Pipelining: Neue Requests können bereits verschickt werden, bevor die Antwort auf den vorherigen Request eingetroffen ist
  - nur bei Bilder- und Flash-Requests möglich
- ▶ Requests und Responses können nur nacheinander übertragen werden
- ▶ Bild- und Flash-Dateien können nur nacheinander verschickt werden

## Ersparnis

Aus dem Sequenzdiagramm sieht man einfach:  
man spart 484 Millisekunden.

### Alternativrechnung

Man spart 3 TCP-Aufbauten und 2 HTTP-Requests.

$$\Rightarrow 3 \cdot 150 + 2 \cdot 17 = 484 \text{ Millisekunden}$$

## Formularverarbeitung

### Welche Daten werden in welcher Reihenfolge zwischen Client und Server ausgetauscht?

- ▶ Anmeldung an Web-Forum
- ▶ Benutzername und Paßwort in Formular eintragen
- ▶ Benutzerdaten korrekt: neuste Beiträge
- ▶ Benutzerdaten falsch: Fehlerseite

## Ablaufskizze

...

## Ablaufskizze - Reihenfolge

1. → Anfrage Forum-URL
2. ← HTML-Formular zur Anmeldung
3. → `user=<user>&pass=<pass>`
4. ↓ Script mit user, pass aufrufen
5. ✓ `check(user, pass)`
6. ↗ `check=true` oder `check=false`
7. ↑ a) wenn `check=false`: Fehlerseite erzeugen  
und an Server senden  
    ✓ b) wenn `check=true`: Forenseiten abfragen
8. ← a) Fehlerseite ausliefern  
    ↗ b) Forenseiten an Script liefern
9. ↑ b) aus Forenseiten Webseite erzeugen  
und an Server senden
10. ← b) Forenseite ausliefern

## Ablaufskizze - Reihenfolge

### Erklärungen zu den Pfeilen

- Client (Browser) zu Webserver
- ← Webserver zu Client (Browser)
- ↓ Webserver zu CGI-Script
- ↑ CGI-Script zu Webserver
- ✓ CGI-Script zu DB-System
- ↗ DB-System zu CGI-Script

## Formularverarbeitung mit Perl

Folgende Daten sollen an ein Perlscript `calc.pl` übergeben werden:

- ▶ Zahl  $x$
- ▶ Zahl  $y$
- ▶ Operator `op` (+, -, \* oder /)

Ausgabe: HTML-Ausgabe der Art

$x \text{ op } y = z$  mit korrekt berechnetem  $z$

## HTML-Formular

```
<form action="/cgi-bin/calc.pl" method="get">
  x: <input type="text" name="x">
  op: <input type="text" name="op">
  y: <input type="text" name="y">
  <input type="submit" value="">
</form>
```

Um das Formular herum sollte natürlich noch ein gültiges HTML-Gerüst.

Als Übertragungsmethode für die Formulardaten wurde `get` gewählt.

D.h. die Variablen werden „in der Adreßleiste“ übergeben, was dann in etwa so aussieht:

```
[...]calc.pl?x=1&op=+&y=1
```

## calc.pl

```
#!/usr/bin/perl
print "content-type: text/plain \n \n";

$query_string = $ENV('QUERY_STRING');
($xPar, $opPar, $yPar) = split(/&/, query_string);
($xVal, $yVal) = split(/=/, $xPar);
($yVal, $yVal) = split(/=/, $yPar);
($opVal, $opVal) = split(/=/, $opPar);

if ($opVal eq "+") {
  $Res = $xVal + $yVal;
} elsif ($opVal eq "-") {
  $Res = $xVal - $yVal;
} elsif ($opVal eq "*") {
  $Res = $xVal * $yVal;
} elsif ($opVal eq "/") {
  $Res = $xVal / $yVal;
} else {
  $Res = "???";
}
print $xVal, $opVal, $yVal, "=", $Res, "\n";
```

## Zusammenfassung

- ▶ **XML-Schema:** Ein Schema erstellen können, auch mit regulären Ausdrücken in Typen
- ▶ **XSL/XPath:** XSLT und XPath-Ausdrücke erstellen, verstehen und anwenden können.  
Im Besonderen: XML → HTML
- ▶ **HTTP:** Keep-Alive und Pipelining erklären können; Datenmengen und Übertragungszeiten berechnen und vergleichen können
- ▶ **Webarchitektur:** Datenaustausch zwischen Client und Server skizzieren und erklären können
- ▶ **Dynamische Webseiten:** einfache Skripte erstellen können