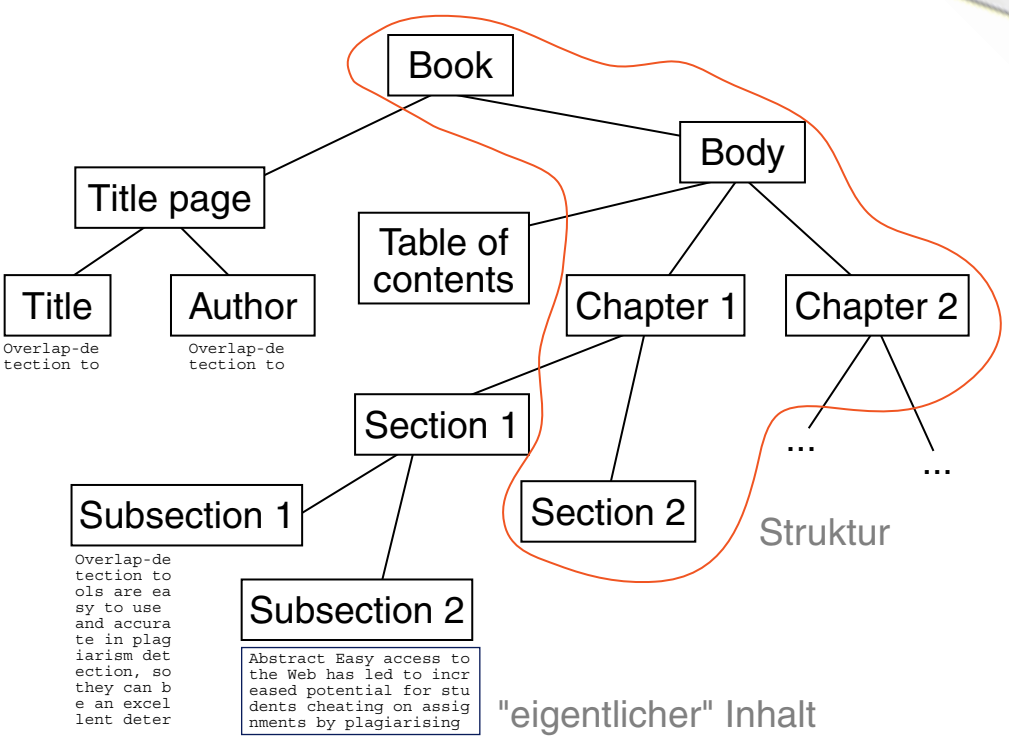
Kapitel WT:III

Auszeichnungssprachen

Einführung

Trennung von

1. Dokumenteninhalt
2. Dokumentendarstellung bzw. Layout



Layout

Auszeichnungssprachen

Einführung

Beispiel LaTeX:

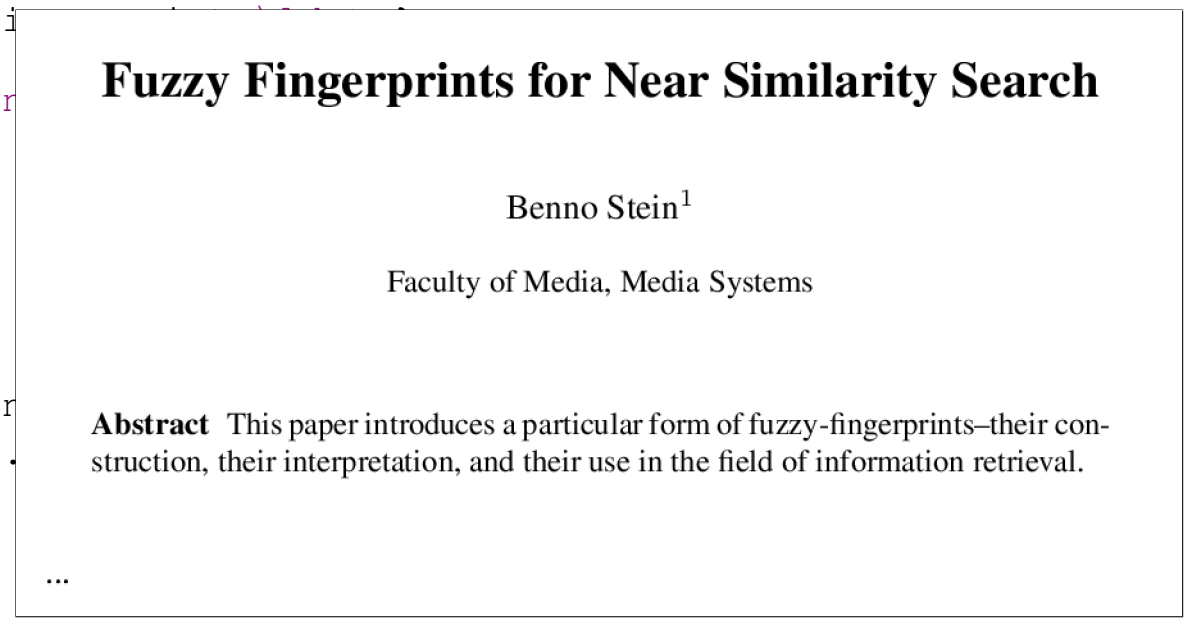
```
\documentclass{llncs}
\usepackage[T1]{fontenc}
\usepackage[german,american]{babel}
\usepackage{graphicx}

\begin{document}

\title{Fuzzy Fingerprints for Near Similarity Search}
\titlerunning{Fuzzy Fingerprints\ldots}

\author{Benno Stein\inst{1}}
\institute{Faculty of Media, Media Systems}

\maketitle

\begin{abstract}
This paper introduces a particular form of fuzzy-fingerprints--their
construction, their interpretation, and their use in the field of
information retrieval.
\end{abstract}

...
```

Auszeichnungssprachen

Einführung

Beispiel LaTeX:

```
\documentclass{llncs}
\usepackage[T1]{fontenc}
\usepackage[german,american]{babel}
\usepackage{graphicx}

\begin{document}

\title{Fuzzy Fingerprints for Near Similarity Search}
\titlerunning{Fuzzy Fi
\author{Benno Stein\ir
\institute{Faculty of

\maketitle

\begin{abstract}
This paper introduces
construction, their ir
information retrieval.
\end{abstract}

...
```



Fuzzy Fingerprints for Near Similarity Search

Benno Stein[1]

Faculty of Media, Media Systems

Abstract This paper introduces a particular form of fuzzy-fingerprints–their construction, their interpretation, and their use in the field of information retrieval.

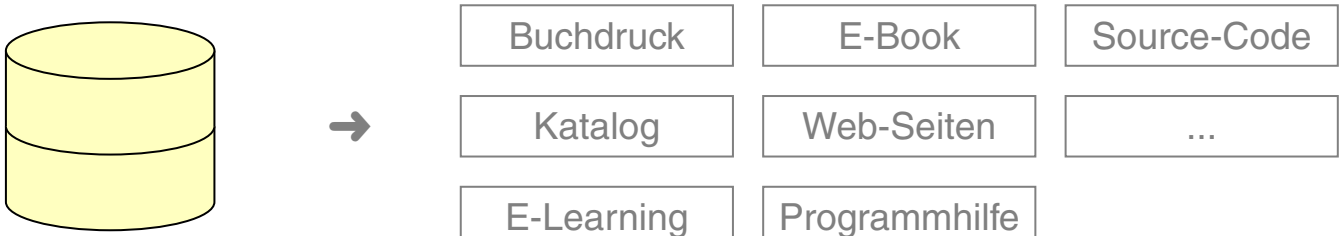
...

Auszeichnungssprachen

Einführung

Trennung von Inhalt und Darstellung ermöglicht:

- Layout- und geräteunabhängige Archivierung
- maschinelle Analyse und Verarbeitung von Strukturinformation. Beispiele: Indexe, Seitenzahlen, Verweise, Zitate
- Single-Source-Prinzip: die Änderung an *einer* Quelle wird in allen nachfolgenden Layout-Prozessen nachvollzogen
- Database Publishing, Cross Media Publishing



Bemerkungen:

- Mögliche Zielformate eines Layout-Prozesses:
  - Postscript PS
  - Portable Document Format PDF
  - Rich Text Format RTF
  - Microsoft Help Format HLP
  - Hypertext Markup Language HTML
  - Compiled HTML CHM
  - Programm-Code

Auszeichnungssprachen

Einführung

Merkmale von Auszeichnungssprachen:

- Strukturinformation wird in den „eigentlichen“ Inhalt integriert.
  ➜ Meta-Sprache zur Auszeichnung von Strukturinformation
- Auszeichnung = Markup; Auszeichnungssprache = Markup-Sprache
- Markup-Symbol *(Tag)* = Wort aus der Markup-Sprache;
  insbesondere: Unterscheidung von Start-Tags und End-Tags

Auszeichnungssprachen

Einführung

Merkmale von Auszeichnungssprachen:

- Strukturinformation wird in den „eigentlichen“ Inhalt integriert.
  ➜ Meta-Sprache zur Auszeichnung von Strukturinformation
- Auszeichnung = Markup; Auszeichnungssprache = Markup-Sprache
- Markup-Symbol *(Tag)* = Wort aus der Markup-Sprache;
  insbesondere: Unterscheidung von Start-Tags und End-Tags

Forderungen an Auszeichnungssprachen:

- Syntax und Semantik von Markup-Symbolen definierbar
- erweiterbar hinsichtlich neuer Strukturelemente und Dokumententypen
- von Menschen schreib- und lesbar
- einbettbar in Programmiersprachen
- offen für zukünftige Entwicklungen: neue Medientypen, Medienintegration

Auszeichnungssprachen

SGML

Historie:

60er	einheitliches Datenformat soll Datenverarbeitung flexibler machen
70er	C. Goldfarb entwickelt bei IBM die Generalized Markup Language GML
1986	Standardisierung von GML ➜ SGML = Standard GML, ISO/IEC 8879

Auszeichnungssprachen

SGML

Historie:

60er einheitliches Datenformat soll Datenverarbeitung flexibler machen

70er C. Goldfarb entwickelt bei IBM die Generalized Markup Language GML

1986 Standardisierung von GML ➜ SGML = Standard GML, ISO/IEC 8879

Konzepte von SGML:

1. SGML-Deklaration

2. SGML Document Type Definition, DTD

3. SGML-Dokument

Auszeichnungssprachen

SGML

Historie:

60er einheitliches Datenformat soll Datenverarbeitung flexibler machen

70er C. Goldfarb entwickelt bei IBM die Generalized Markup Language GML

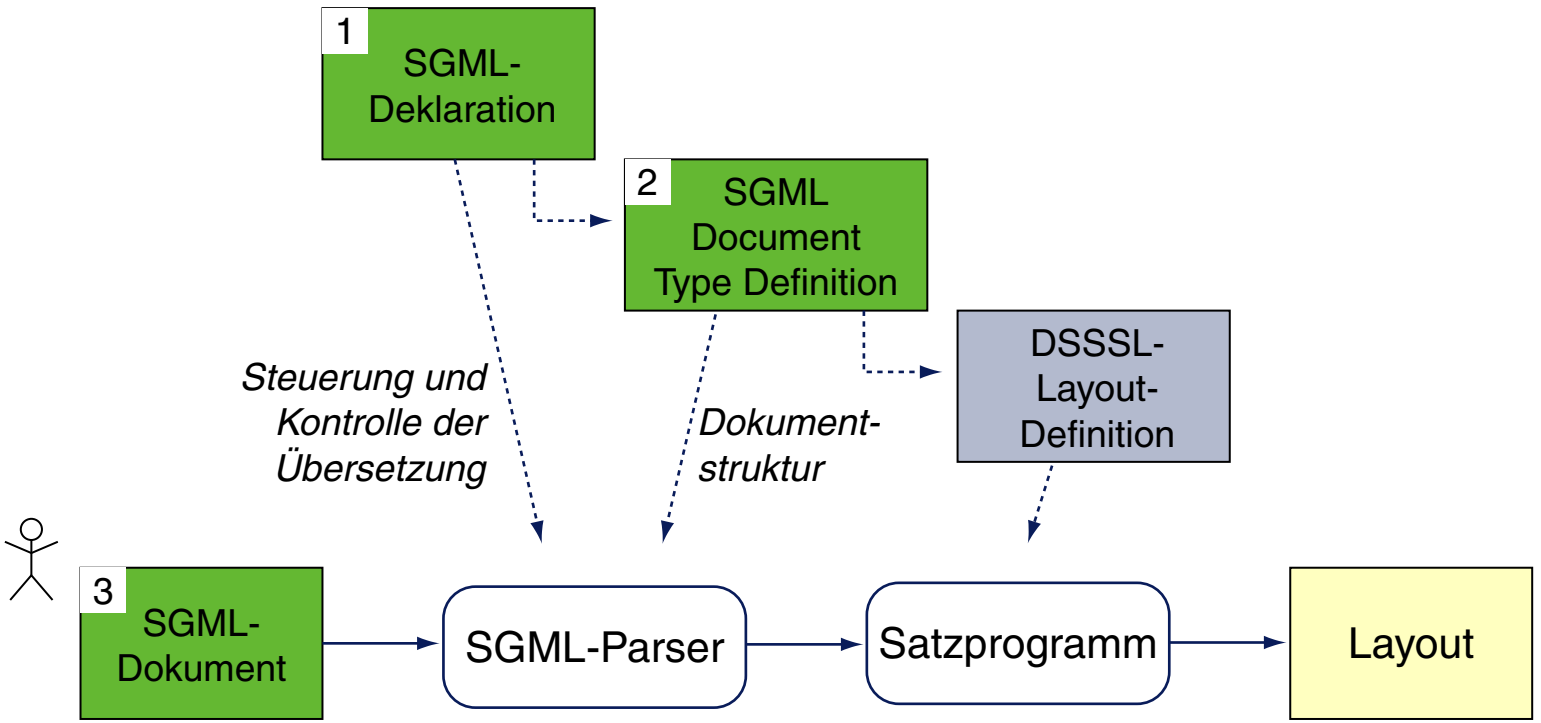
1986 Standardisierung von GML ➜ SGML = Standard GML, ISO/IEC 8879

Konzepte von SGML:

1. SGML-Deklaration
   Definiert Zeichenvorrat, Steuerzeichen, Auszeichnungsregeln für Parser.
2. SGML Document Type Definition, DTD
   Dokumentenklasse; definiert die strukturellen Elementtypen eines SGML-Dokuments, „gegen“ die der Parser analysiert.
3. SGML-Dokument
   Enthält eine Instanz des Strukturbaums gemäß einer DTD; die Blätter des Strukturbaums bilden den eigentlichen Inhalt.

Auszeichnungssprachen

SGML Dokumentenverarbeitung



Bemerkungen:

- Das Formatieren ist nicht Bestandteil von SGML.
- Für Layout-spezifische und geräteabhängige Definitionen zur Darstellung der in SGML beschriebenen Strukturelemente wurde eine spezielle Sprache, die *Document Style Semantics and Specification Language* DSSSL entwickelt. [Wikipedia]
- DSSSL wird „Dissel“ ausgesprochen.

Auszeichnungssprachen

SGML-Dokument

Allgemeine Form einer SGML-Elementinstanz [HTML] :

`<elementName Attributliste>` *eigentlicher Inhalt* `</elementName>`

Beispiel:

```
<table width="100%" cellspacing="0" cellpadding="0" border="0">
 ...
</table>
```

Auszeichnungssprachen

SGML-Dokument

Allgemeine Form einer SGML-Elementinstanz [HTML] :

```
<elementName Attributliste> eigentlicher Inhalt </elementName>
```

Beispiel:

```
<table width="100%" cellspacing="0" cellpadding="0" border="0">
 ...
</table>
```

Abstrakte Syntax für Markup-Symbole *(Tags)*:

```
<start-tag>    =  <stag-open> <identifier> {<attribute>}* <tag-close>
<end-tag>      =  <etag-open> <identifier> <tag-close>
<attribute>    =  <identifier> = "<value>"
```

Konkrete Syntax für Markup-Symbole:

```
<stag-open>    =  <
<etag-open>    =  </
<tag-close>    =  >
<identifier>   =  {xchar}+
<value>        =  {char}+
```

Auszeichnungssprachen

SGML-Dokument

Allgemeine Form einer SGML-Elementinstanz [HTML] :

`<elementName Attributliste>` *eigentlicher Inhalt* `</elementName>`

Beispiel:

```
<table width="100%" cellspacing="0" cellpadding="0" border="0">
 ...
</table>
```

Abstrakte Syntax für Markup-Symbole *(Tags)*:

```
<start-tag>    =  <stag-open> <identifier> {<attribute>}* <tag-close>
<end-tag>      =  <etag-open> <identifier> <tag-close>
<attribute>    =  <identifier> = "<value>"
```

Konkrete Syntax für Markup-Symbole:

```
<stag-open>    =  <
<etag-open>    =  </
<tag-close>    =  >
<identifier>   =  {xchar}+
<value>        =  {char}+
```

Auszeichnungssprachen

SGML-Dokument

Allgemeine Form einer SGML-Elementinstanz [HTML] :

```
<elementName Attributliste> eigentlicher Inhalt </elementName>
```

Beispiel:

```
<table width="100%" cellspacing="0" cellpadding="0" border="0">
 ...
</table>
```

Abstrakte Syntax für Markup-Symbole *(Tags)*:

```
<start-tag>    =  <stag-open> <identifier> {<attribute>}* <tag-close>
<end-tag>      =  <etag-open> <identifier> <tag-close>
<attribute>    =  <identifier> = "<value>"
```

Konkrete Syntax für Markup-Symbole:

```
<stag-open>    =  <
<etag-open>    =  </
<tag-close>    =  >
<identifier>   =  {xchar}+
<value>        =  {char}+
```

Auszeichnungssprachen

SGML-Dokument

Allgemeine Form einer SGML-Elementinstanz [HTML] :

```
<elementName Attributliste> eigentlicher Inhalt </elementName>
```

Beispiel:

```
<table width="100%" cellspacing="0" cellpadding="0" border="0">
 ...
</table>
```

Abstrakte Syntax für Markup-Symbole *(Tags)*:

```
<start-tag>    =  <stag-open> <identifier> {<attribute>}* <tag-close>
<end-tag>      =  <etag-open> <identifier> <tag-close>
<attribute>    =  <identifier> = "<value>"
```

Konkrete Syntax für Markup-Symbole:

```
<stag-open>    =  <
<etag-open>    =  </
<tag-close>    =  >
<identifier>   =  {xchar}+
<value>        =  {char}+
```

Auszeichnungssprachen

SGML-Dokument

Allgemeine Form einer SGML-Elementinstanz [HTML] :

```
<elementName Attributliste> eigentlicher Inhalt </elementName>
```

Beispiel:

```
<table width="100%" cellspacing="0" cellpadding="0" border="0">
 ...
</table>
```

Abstrakte Syntax für Markup-Symbole *(Tags)*:

```
<start-tag>    =  <stag-open> <identifier> {<attribute>}* <tag-close>
<end-tag>      =  <etag-open> <identifier> <tag-close>
<attribute>    =  <identifier> = "<value>"
```

Konkrete Syntax für Markup-Symbole:

```
<stag-open>    =  <
<etag-open>    =  </
<tag-close>    =  >
<identifier>   =  {xchar}+
<value>        =  {char}+
```

Auszeichnungssprachen

SGML-Dokument

Allgemeine Form einer SGML-Elementinstanz [HTML] :

```
<elementName Attributliste> eigentlicher Inhalt </elementName>
```

Beispiel:

```
<table width="100%" cellspacing="0" cellpadding="0" border="0">
 ...
</table>
```

Abstrakte Syntax für Markup-Symbole *(Tags)*:

```
<start-tag>    =  <stag-open> <identifier> {<attribute>}* <tag-close>
<end-tag>      =  <etag-open> <identifier> <tag-close>
<attribute>    =  <identifier> = "<value>"
```

Konkrete Syntax für Markup-Symbole:

```
<stag-open>    =  <
<etag-open>    =  </
<tag-close>    =  >
<identifier>   =  {xchar}+
<value>        =  {char}+
```

Auszeichnungssprachen

SGML-Dokument

Allgemeine Form einer SGML-Elementinstanz [HTML] :

`<elementName Attributliste>` *eigentlicher Inhalt* `</elementName>`

Beispiel:

```
<table width="100%" cellspacing="0" cellpadding="0" border="0">
 ...
</table>
```

Abstrakte Syntax für Markup-Symbole *(Tags)*:

```
<start-tag>    =  <stag-open> <identifier> {<attribute>}* <tag-close>
<end-tag>      =  <etag-open> <identifier> <tag-close>
<attribute>    =  <identifier> = "<value>"
```

Konkrete Syntax für Markup-Symbole:

```
<stag-open>    =  <
<etag-open>    =  </
<tag-close>    =  >
<identifier>   =  {xchar}+
<value>        =  {char}+
```

Auszeichnungssprachen

SGML Document Type Definition [XML]

Die DTD definiert:

1. Art und Aufbau von Elementtypen für eine Klasse von Dokumenten
   = die Inhaltsmodelle der Elementtypen
2. die in Elementeninstanzen verwendbaren Attribute und ihre Datentypen
3. verschiedene Arten von Textkonstanten, sogenannte *Entities*

Auszeichnungssprachen

SGML Document Type Definition (Fortsetzung)

Beispiel für eine SGML-Elementtypdeklaration:

```
<!ELEMENT book              (titlepage, body)>
<!ELEMENT titlepage         (title, author)>
<!ELEMENT body              (table-of-contents, chapter+)>
<!ELEMENT chapter           (chapterhead, section+)>
<!ELEMENT title             (#PCDATA)>
```

Auszeichnungssprachen

SGML Document Type Definition (Fortsetzung)

Beispiel für eine SGML-Elementtypdeklaration:

```
<!ELEMENT book          (titlepage, body)>
<!ELEMENT titlepage     (title, author)>
<!ELEMENT body          (table-of-contents, chapter+)>
<!ELEMENT chapter       (chapterhead, section+)>
<!ELEMENT title         (#PCDATA)>
```

`<!ELEMENT`	Beginn der Deklaration des Elementtyps
`chapter`	Name der Elementtypdeklaration
`(`	Beginn des Inhalts der Deklaration
`chapterhead,`	genau ein Kapitelkopf muss vorkommen
`section+`	mindestens ein Abschnitt muss vorkommen
`)`	Ende des Inhalts
`>`	Ende der Deklaration
`#PCDATA`	Datentyp “Parsed Character Data” [www.w3schools.com 1 2]

Auszeichnungssprachen

SGML Document Type Definition (Fortsetzung)

Beispiel für eine SGML-Elementtypdeklaration:

```
<!ELEMENT book              (titlepage, body)>
<!ELEMENT titlepage         (title, author)>
<!ELEMENT body              (table-of-contents, chapter+)>
<!ELEMENT chapter           (chapterhead, section+)>
<!ELEMENT title             (#PCDATA)>
```

`<!ELEMENT`	Beginn der Deklaration des Elementtyps
`chapter`	Name der Elementtypdeklaration
`(`	Beginn des Inhalts der Deklaration
`chapterhead,`	genau ein Kapitelkopf muss vorkommen
`section+`	mindestens ein Abschnitt muss vorkommen
`)`	Ende des Inhalts
`>`	Ende der Deklaration
`#PCDATA`	Datentyp “Parsed Character Data” [www.w3schools.com 1 2]

Beispiel für die Deklaration einer Textkonstante *(Entity)*:

```
<!ENTITY euro           "&#8364;">
```
[www.w3schools.com]

Bemerkungen:

- Die Elemente einer DTD können in einem SGML-Dokument instantiiert werden und dienen so im eigentlichen Inhalt als Markup.
- Entities werden durch den Aufruf &*Entityname*; referenziert.
- DTDs lassen sich auf zwei Arten einsetzen:
  1. Zur Analyse, um vorgegebene Dokumente zu validieren.
  2. Zur Synthese, um neue Dokumente zu generieren.

Auszeichnungssprachen

Zusammenhang SGML, XML, HTML, XHTML

XML, *Extensible Markup Language* [W3C], ist eine Teilmenge von SGML, die speziell auf die Bedürfnisse des WWW zugeschnitten und stark vereinfacht ist:

- XML hat eine fixe, nicht veränderbare SGML-Deklaration.

Auszeichnungssprachen

Zusammenhang SGML, XML, HTML, XHTML

XML, *Extensible Markup Language* [W3C], ist eine Teilmenge von SGML, die speziell auf die Bedürfnisse des WWW zugeschnitten und stark vereinfacht ist:

- XML hat eine fixe, nicht veränderbare SGML-Deklaration.

HTML, *Hypertext Markup Language* [W3C], ist eine Teilmenge von SGML und, verglichen mit XML, noch weiter eingeschränkt:

- HTML hat eine fixe, nicht veränderbare SGML-Deklaration.
- HTML verwendet eine fixe Dokumentstruktur (und hat folglich nur *eine* DTD).
  ➜ Kein Austausch von SGML-Deklaration und DTD erforderlich.
- Vergleiche hierzu die SGML Dokumentenverarbeitung.

Auszeichnungssprachen

Zusammenhang SGML, XML, HTML, XHTML

XML, *Extensible Markup Language* [W3C], ist eine Teilmenge von SGML, die speziell auf die Bedürfnisse des WWW zugeschnitten und stark vereinfacht ist:

- XML hat eine fixe, nicht veränderbare SGML-Deklaration.

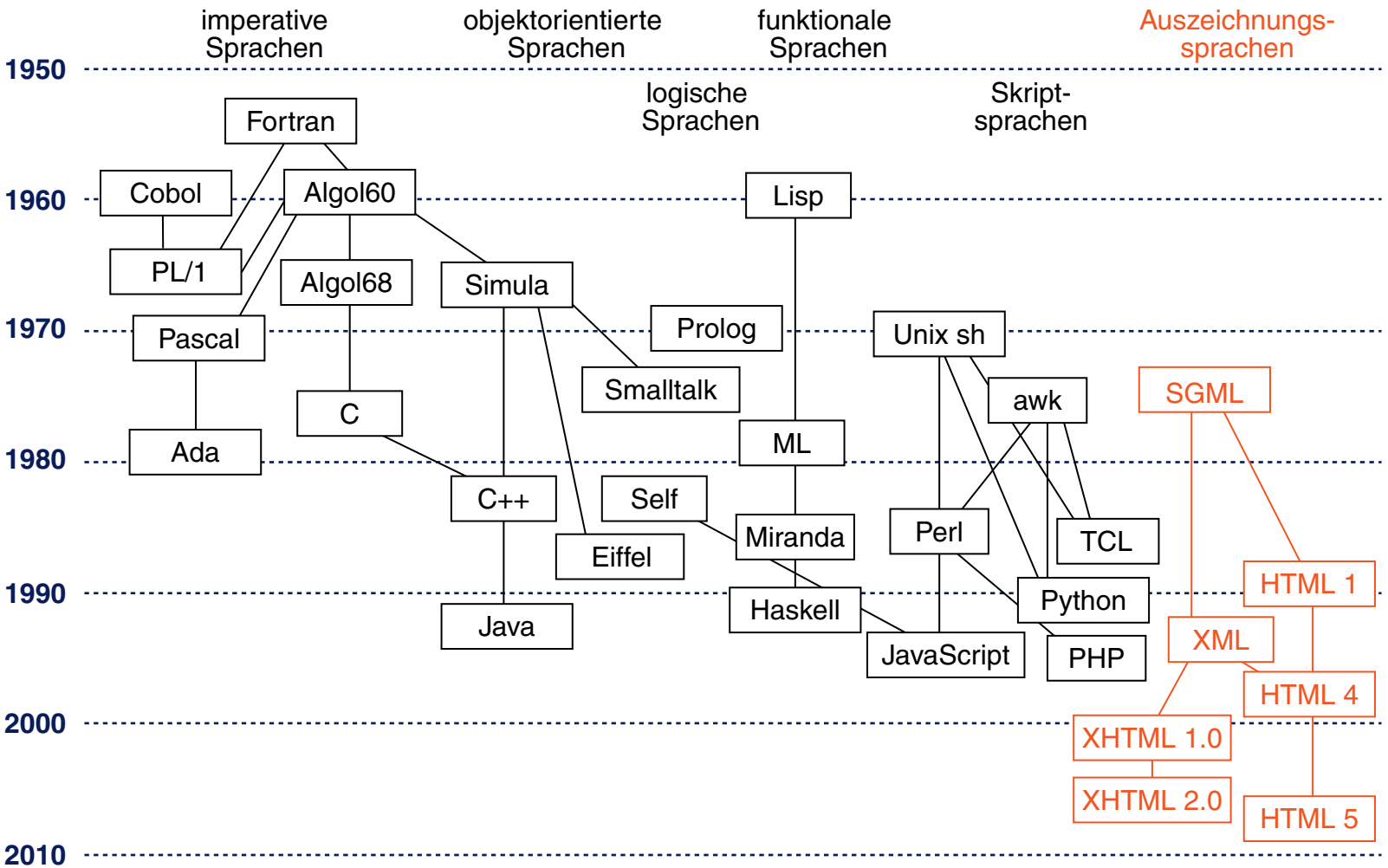
HTML, *Hypertext Markup Language* [W3C], ist eine Teilmenge von SGML und, verglichen mit XML, noch weiter eingeschränkt:

- HTML hat eine fixe, nicht veränderbare SGML-Deklaration.
- HTML verwendet eine fixe Dokumentstruktur (und hat folglich nur *eine* DTD).
  ➜ Kein Austausch von SGML-Deklaration und DTD erforderlich.
- Vergleiche hierzu die SGML Dokumentenverarbeitung.

XHTML, *Extensible HyperText Markup Language*:

- XHTML 1 ist die Definition von HTML auf Basis von XML. [W3C]
- XHTML 2 ist die Weiterentwicklung von XHTML 1 und ist nicht HTML-5-kompatibel. [W3C]

Auszeichnungssprachen



[Kastens 2005, Watt 1996, www.levenez.com/lang]

Kapitel WT:III (Fortsetzung)

HTML

Einordnung

SGML hat alle notwendigen Konzepte.

Warum überhaupt HTML?

1. ein guter Kompromiss zwischen Einfachheit und Ausdrucksstärke
2. Verfügbarkeit und Plattformunabhängigkeit

Das „klassische HTML“ macht keine strikte Trennung zwischen Dokumenteninhalt und Dokumentendarstellung.

HTML

Historie

1991 Vorstellung von frühen Versionen von URL, HTTP und HTML.

1994 HTML 2.0, mit standardkonformer SGML-DTD.

1997 HTML 3.2, mit Frames, Tabellen, textumflossenen Bilder, Java-Applets.

1998 HTML 4.0, führt Cascading Stylesheets CSS ein.

2000 XHTML 1.0, Neudefinition von HTML auf XML-Basis. [W3C] [Wikipedia]

2001 XHTML 1.1, Modularisierung von XHTML. [W3C]

2010 XHTML 2.0, Working Group Note. [W3C]

HTML

Historie

1991 Vorstellung von frühen Versionen von URL, HTTP und HTML.

1994 HTML 2.0, mit standardkonformer SGML-DTD.

1997 HTML 3.2, mit Frames, Tabellen, textumflossenen Bilder, Java-Applets.

1998 HTML 4.0, führt Cascading Stylesheets CSS ein.

2000 XHTML 1.0, Neudefinition von HTML auf XML-Basis. [W3C] [Wikipedia]

2001 XHTML 1.1, Modularisierung von XHTML. [W3C]

2010 XHTML 2.0, Working Group Note. [W3C]

2008 HTML 5, Working Draft. Loslösung von SGML, Fehlerbehandlung, neue Struktur- und Multimedia-Elemente.

2012 HTML 5, Candidate Recommendation. [W3C] [Wikipedia]

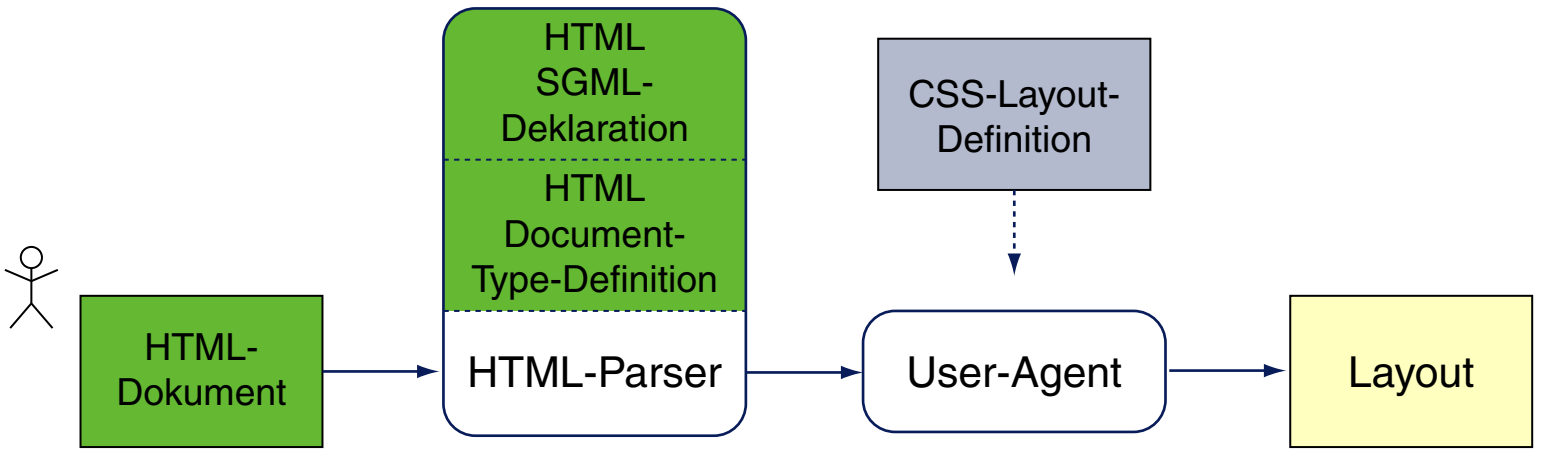
2013 HTML, Living Standard [whatwg]

Bemerkungen:

- Beispiele für die fehlende Trennung zwischen Dokumenteninhalt und Dokumentendarstellung sind Formatierungsangaben wie `<FONT>`, `<CENTER>`, etc.
- Mit der Einführung von Cascading Stylesheets in HTML 4.0 existiert ein Mechanismus, um Formatierungsangaben aus dem Dokumenteninhalt auszugliedern.
- Um Abwärtskompatibilität zu existierenden HTML-Standards zu gewährleisten, wurden drei Kompatibilitäts-DTDs entwickelt: Transitional-DTD, Strict-DTD, Frameset-DTD.
- XHTML 1.0 bringt keine neue Funktionalität gegenüber HTML 4.0, enthält aber die (kleineren) syntaktischen Anpassungen für den XML-Standard.
- Bei der Weiterentwicklung von XHTML 2.0 konnte keine Einigung zwischen W3C und der Industrie (WHATWG-Konsortium) erzielt werden. Mittlerweile arbeiten W3C und WHATWG gemeinsam an HTML 5. [W3C]
- HTML 5 reagiert auf die große Menge nicht valider Dokumente im Web (Stichwort: *tag soup*), die bislang jeder Browser auf eigene Weise behandelt, mit einer standardisierten Fehlerbehandlung (Stichwort: *quirks mode*). [W3C] [Wikipedia]
- HTML 5 führt eine Reihe neuer Elemente ein. Beispielsweise ermöglichen `<canvas>`, `<video>` und `<audio>` die native Medieneinbindung und sollen Plugin-Technologien wie Flash überflüssig machen. Strukturelemente wie `<article>`, `<section>`, `<header>` oder `<nav>` sollen die Semantik eines Elements im Dokument explizit machen und die Lesbarkeit (auch für Maschinen) erhöhen. [W3C] [w3schools]
- HTML 5 oder HTML5? [1, 2]

HTML

HTML Dokumentenverarbeitung



Vergleiche hierzu die SGML Dokumentenverarbeitung.

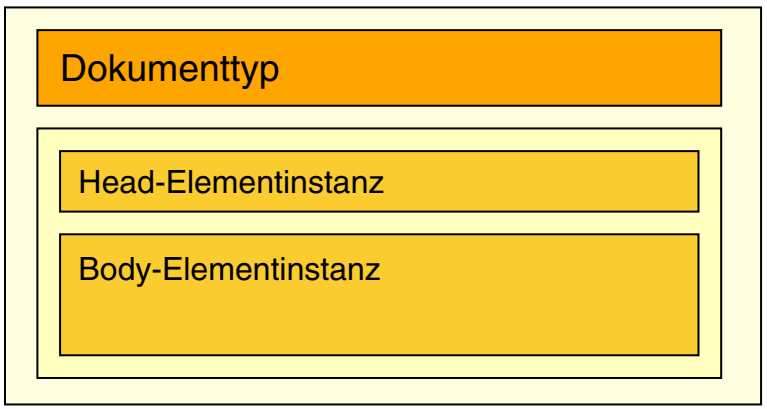
Bemerkungen:

- CSS (Cascading Stylesheets) ist eine Formatierungssprache, die für HTML – genauso wie DSSSL (Document Style Semantics and Specification Language) für SGML – einem mit Markup versehenen Dokument eine ausgabegeräteabhängige Layout-Vorschrift zuordnet.

HTML

HTML-Dokument

HTML-Dokument




```
<!DOCTYPE HTML PUBLIC ... >

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

- Die Angabe des Dokumenttyps (einschließlich einer DTD-Deklaration bis HTML 4) dient zur korrekten Interpretation des Dokuments.
- Der Dokumenten-Head enthält Meta-Informationen über das Dokument.
- Der Dokumenten-Body enthält HTML-Elementinstanzen.
- Vergleiche hierzu die XML-Dokumentstruktur.

HTML

DTD-Deklaration [W3C] [SELFHTML]

HTML verwendet eine fixe Dokumentstruktur, die bis HTML 4 als DTD spezifiziert ist. Unterscheidung von drei DTD-Varianten (HTML 4.01):

1. Strict

   ```
   <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
   ```

   Trennung zwischen Inhalt und Darstellung: keine Formatierungsangaben erlaubt; strenge Verschachtelungsregeln; kein Inhalt ohne Block-Auszeichnung.

2. Transitional

   ```
   <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
       "http://www.w3.org/TR/html4/loose.dtd">
   ```

   Hat nicht die Beschränkungen der Strict-DTD.

3. Frameset

   ```
   <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
       "http://www.w3.org/TR/html4/frameset.dtd">
   ```

   Für HTML-Dokumente mit Framesets.

Bemerkungen:

- Ein HTML-4-Dokument darf nur *eine* DTD besitzen. Bei der Verwendung von Frames ermöglicht die Frameset-DTD für jedes Frame die Einbindung einer DTD.
- Ein HTML-4-Dokument ohne DTD-Deklaration wird häufig nach den Regeln der Transitional-DTD für HTML 4.01 verarbeitet.
- HTML 5 ist weitgehend kompatibel zu HTML 4.01 und XHTML, basiert aber nicht mehr auf SGML; folglich ist die Dokumentstruktur nicht mehr in Form einer DTD spezifiziert. Ein HTML-5-Dokument sollte mit der Angabe eines Dokumentyps `<!DOCTYPE html>` beginnen. [W3C] [Wikipedia]

HTML

Dokumenten-Head

- Titel [SELFHTML]

```
<head>
  <title>Lemmy Caution's Strange Adventures</title>
</head>
```

Der Titel ist obligatorisch; der Titel erscheint nicht im dargestellten HTML-Dokument, wird aber als Fenstertitel, Lesezeichen, von Robots, etc. ausgewertet.

- Meta-Tags [SELFHTML]

```
<head>
  <title>...</title>

  <meta name="author" content="Judea Pearl">
  <meta name="keywords" content="Heuristics, Search, Bayes">
  <meta http-equiv="Content-Script-Type" content="text/javascript">
  <meta http-equiv="Content-Style-Type" content="text/css">

</head>
```

Meta-Tags haben meist zwei Attribute „Eigenschaft = Wert“ (`name` bzw. `http-equiv` = `content`); sie dienen zur Information von Web-Browsern, Robots und Web-Servern.

Bemerkungen:

- Seit Version 4.0 schreibt der HTML-Standard keine Meta-Tags mehr vor, sondern definiert nur deren Aufbau. Zur Standardisierung von Meta-Tags hat das W3C die Sprache RDF (*Resource Description Framework*) entworfen.
- Meta-Tags, die mit `name` definiert werden, richten sich an Client-Programme und Robots. Meta-Tags, die mit `http-equiv` definiert werden, sind für den Web-Server gedacht.

HTML

Dokumenten-Head (Fortsetzung)

- Adressbasis [SELFHTML]

```
<head>
  <title>...</title>

  <base href="http://www.my-webserver.de/absolute/path">

</head>
```

Definiert absoluten Bezugspfad und ermöglicht so die Verwendung von relativen Pfaden im Dokument.

- Links [SELFHTML]

```
<head>
  <title>...</title>

  <link rel="stylesheet" type="text/css" href="../share/bib.css">

</head>
```

Ermöglicht die Referenzierung (keine Hyperlinks) von Dokumenten; wird meist zur Angabe von externen Stylesheets verwendet.

HTML

Dokumenten-Body

Allgemeine Form einer HTML-Elementinstanz [SGML] :

<*Elementname* `Attributliste`> *eigentlicher Inhalt* </*Elementname*>

Elementinstanzen fallen in zwei Klassen [SELFHTML] :

1. Block-Elemente

2. Inline-Elemente

HTML

Dokumenten-Body

Allgemeine Form einer HTML-Elementinstanz [SGML] :

<*Elementname* `Attributliste`> *eigentlicher Inhalt* </*Elementname*>

Elementinstanzen fallen in zwei Klassen [SELFHTML] :

1. Block-Elemente
   Instanzen von Block-Elementen erzeugen einen eigenen Absatz im Textfluss; sie können normalen Text und Instanzen von Inline-Elementen enthalten; einige dürfen auch Instanzen anderer Block-Elemente enthalten.

   Beispiele für Block-Elemente:
   `<center>, <div>, <form>, <h1>, <noframes>, <p>, <table>, <ul>`

2. Inline-Elemente
   Instanzen von Inline-Elementen werden in derselben Zeile wie der vorhergehende Text gesetzt; sie können normalen Text und Instanzen weiterer Inline-Elemente enthalten.

   Beispiele für Inline-Elemente:
   `<a>, <br>, <cite>, <em>, <font>, <img>, <small>, <span>`

Bemerkungen:

- HTML 5 erweitert die Möglichkeiten der Inhaltsmodelle über die strikte Unterscheidung von Block- und Inline-Elementen hinaus. [W3C 1 2]
- Die Verarbeitung von Block-Elementen aus Sicht des Layout-Programms (beispielsweise mit einem Web-Browser) ist mit dem Verhalten von LaTeX im `\vmode` vergleichbar. Die Verarbeitung von Inline-Elementen ist mit dem Verhalten von LaTeX im `\hmode` vergleichbar.

HTML

Elemente zur Textstrukturierung

- Überschriften [SELFHTML]

```
<h1>&Uuml;berschrift 1. Ordnung</h1>
...
<h6>&Uuml;berschrift 6. Ordnung</h6>
```

- Listen [SELFHTML]

`<ol>`	geordnete Liste
`<ul>`	ungeordnete Liste
`<li>`	Listeneintrag
`<dl>`	Definitionsliste
`<dt>`	Definitionsüberschrift
`<dd>`	Definitionseintrag

HTML

Elemente zur Textstrukturierung

- Überschriften [SELFHTML]

```
<h1>&Uuml;berschrift 1. Ordnung</h1>
...
<h6>&Uuml;berschrift 6. Ordnung</h6>
```

- Listen [SELFHTML]

`<ol>`	geordnete Liste
`<ul>`	ungeordnete Liste
`<li>`	Listeneintrag
`<dl>`	Definitionsliste
`<dt>`	Definitionsüberschrift
`<dd>`	Definitionseintrag

- Tabellen [SELFHTML]

`<table>`	Tabelle
`<caption>`	Tabellenüberschrift
`<thead>`	Tabellenkopf
`<tbody>`	Tabellenkörper
`<tfoot>`	Tabellenfuß
`<tr>`	Tabellenzeile
`<td>`	einzelne Zelle
`<th>`	Zelle mit Überschrift
`<col>`	Tabellenspalte

Spalte 1	Spalte 2	Spalte 3
Zelle 1.1	Zelle 1.2	Zelle 1.3
Zelle 2.1	Zelle 2.2	Zelle 2.3

HTML

Elemente zur Textstrukturierung (Fortsetzung)

- Semantische Strukturelemente (HTML 5) [W3C]

`<header>`	Kopfinformationen einer Website oder eines Artikels
`<footer>`	Fußzeile einer Website oder eines Artikels
`<nav>`	Navigationsmenü oder andere Navigationsmöglichkeiten
`<article>`	eigenständiger Inhalt, ggf. mit eigenem <header> und <footer>
`<section>`	Gruppierung, typisch mit Überschrift für Inhalte in <article> oder Themenbereiche (<article>)
`<aside>`	Gruppierung von verwandter Information mit Bezug zum Hauptinhalt

HTML

Universalattribute [SELFHTML]

Universalattribute *(Core Attributes)* sind in fast allen Start-Tags erlaubt und werden nicht elementtypspezifisch behandelt. Unterscheidung von drei Arten:

1. Allgemeine

2. Zur Internationalisierung

3. Zum Event-Handling

HTML

Universalattribute [SELFHTML]

Universalattribute *(Core Attributes)* sind in fast allen Start-Tags erlaubt und werden nicht elementtypspezifisch behandelt. Unterscheidung von drei Arten:

1. Allgemeine

`class`	ordnet der Elementinstanz eine Stylesheet-Klasse zu
`id`	ordnet der Elementinstanz einen individuellen Namen zu
`style`	definiert CSS-Angaben zur Formatierung der Elementinstanz
`title`	definiert einen Titel für die Elementinstanz

2. Zur Internationalisierung

`dir`	gibt die Schreibrichtung für Text in der Elementinstanz an
`lang`	gibt die verwendete Landessprache (nach RFC 1766) an

3. Zum Event-Handling

`onclick`	Ausführen von Script-Code beim Anklicken der Elementinstanz
`onmouseover`	Ausführen von Script-Code beim Überfahren der Elementinstanz
`...`	

HTML

Textauszeichnungen

Textauszeichnungen sind vom Typ „inline“ und lassen sich hinsichtlich ihrer Konkretheit unterscheiden:

1. Physische Auszeichnungen (HTML 4) [SELFHTML]

2. Logische Auszeichnungen [SELFHTML]

HTML

Textauszeichnungen

Textauszeichnungen sind vom Typ „inline“ und lassen sich hinsichtlich ihrer Konkretheit unterscheiden:

1. Physische Auszeichnungen (HTML 4) [SELFHTML]

`<i>`	zeichnet einen Text als kursiv aus
`<b>`	zeichnet einen Text als fett aus
`<tt>`	zeichnet einen Text in Schreibmaschinenschrift aus
`<u>`	zeichnet einen Text als unterstrichen aus
`<strike>`	zeichnet einen Text als durchgestrichen aus
`...`	

2. Logische Auszeichnungen [SELFHTML]

`<em>`	zeichnet einen Text als betonten, wichtigen Text aus
`<strong>`	zeichnet einen Text als stark betont aus (Steigerung von `<em>`)
`<code>`	zeichnet einen Text als Quelltext aus
`<cite>`	zeichnet einen Text als Zitat aus
`<del>`	zeichnet einen Text als gelöscht aus
`...`	

Bemerkungen:

- In HTML 5 wird auf eine Reihe von Elementen verzichtet, die vordringlich das Layout und physische Auszeichnungen betreffen:
  `<basefont>, <big>, <center>, <font>, <frame>, <frameset>, <noframes>, <s>, <strike>, <tt>, <u>`
- Beispiele für physische Auszeichnungen in LaTeX sind die Schriftschnitte und -gewichte:
  `\itshape`, `\bfseries`, `\fontfamily{phv}\fontsize{8}{0}\selectfont`
- Beispiele für logische Auszeichnungen in LaTeX :
  `\em`, `\begin{quote} ... \end{quote}`

HTML

Hyperlinks [SELFHTML]

Zur Definition von Start und Ziel von Hyperlinks dient das `<a>`-Element *(Anchor)*. Als Inline-Element kann es keine Instanzen von Block-Elementen auszeichnen.

- Hyperlink-Start

- Hyperlink-Ziel

HTML

Hyperlinks [SELFHTML]

Zur Definition von Start und Ziel von Hyperlinks dient das `<a>`-Element *(Anchor)*. Als Inline-Element kann es keine Instanzen von Block-Elementen auszeichnen.

- Hyperlink-Start

`<a` **`href`**`="<`*URI*`>">`	Ziel ist durch <*URI*> definiert
`<a` **`href`**`="#<`*Identifier*`>">`	Ziel ist benannter Anker im Dokument
`<a` **`href`**`="<`*URI*`>#<`*Identifier*`>">`	Ziel ist Konkatenation von <*URI*> und Ankername

Optionale Attribute des Anchor-Elements:

`target`	spezifiziert den Frame, in dem das Zieldokument angezeigt wird; vordefinierte Targets sind `_self, _parent, _top`.
`title`	Definition von Mouse-Over-Text
`type`	MIME-Type des Zieldokuments

- Hyperlink-Ziel

HTML

Hyperlinks [SELFHTML]

Zur Definition von Start und Ziel von Hyperlinks dient das `<a>`-Element *(Anchor)*. Als Inline-Element kann es keine Instanzen von Block-Elementen auszeichnen.

- Hyperlink-Start

`<a` **`href`**`="<`*URI*`>">`	Ziel ist durch <*URI*> definiert
`<a` **`href`**`="#<`*Identifier*`>">`	Ziel ist benannter Anker im Dokument
`<a` **`href`**`="<`*URI*`>#<`*Identifier*`>">`	Ziel ist Konkatenation von <*URI*> und Ankername

Optionale Attribute des Anchor-Elements:

`target`	spezifiziert den Frame, in dem das Zieldokument angezeigt wird; vordefinierte Targets sind `_self, _parent, _top`.
`title`	Definition von Mouse-Over-Text
`type`	MIME-Type des Zieldokuments

- Hyperlink-Ziel

`<a` **`name`**`="<`*Identifier*`>">`	Zieldefinition im selben Dokument

Alternativ kann mittels des Universalattributs `id="<`*Identifier*`>"` fast jede Elementinstanz zum Ziel eines Hyperlinks gemacht werden.

Bemerkungen:

- Die Syntax von Hyperlinks ist unabhängig von dem angegebenen Ziel.
- URIs, die mit einem Dokumentanker #<*Identifier*> abschließen, werden auch als Fragment-Identifier bezeichnet, weil sie ein Dokument nicht als Ganzes, sondern abschnittsgenau adressieren.

HTML

Formulare [SELFHTML]

Alles, was zwischen dem einleitenden `<form>`-Tag und dem abschließenden `</form>`-Tag steht, gehört zum Formular.

- Attribute des `<form>`-Elements

- Kindelemente des `<form>`-Elements

HTML

Formulare [SELFHTML]

Alles, was zwischen dem einleitenden `<form>`-Tag und dem abschließenden `</form>`-Tag steht, gehört zum Formular.

- Attribute des `<form>`-Elements

`action`	definiert URI vom Server-Anwendungsprogramm oder `mailto:`
`method`	spezifiziert die `get` oder `post`-Methode des HTTP-Protokolls
`enctype`	Angabe eines MIME-Typs

- Kindelemente des `<form>`-Elements

`<input>`	Definition von Eingabefeldern
`<select>`	Definition von Auswahllisten
`<option>`	Definition von Optionen innerhalb von `<select>`

HTML

Formulare [SELFHTML]

Alles, was zwischen dem einleitenden `<form>`-Tag und dem abschließenden `</form>`-Tag steht, gehört zum Formular.

- Attribute des `<form>`-Elements

`action`	definiert URI vom Server-Anwendungsprogramm oder `mailto:`
`method`	spezifiziert die `get` oder `post`-Methode des HTTP-Protokolls
`enctype`	Angabe eines MIME-Typs

- Kindelemente des `<form>`-Elements

`<input>`	Definition von Eingabefeldern
`<select>`	Definition von Auswahllisten
`<option>`	Definition von Optionen innerhalb von `<select>`

Attribute des `<input>`-Elements

`type`	Typ des Eingabefeldes: `text, radio, submit, file, hidden`
`size`	definiert die Größe des Eingabefeldes
`value`	definiert einen Default-Wert

Bemerkungen:

- HTML 5 erweitert die Attribute des `<input>`-Elements. So ermöglicht `type` zusätzliche Datentypen mit den passenden Eingaben, `placeholder` eine adäquatere Gestaltung und `autofocus, pattern, required` eine leistungsfähigere Validierung. [W3C]

HTML

Formulare: Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
    <title>Test</title>
  </head>

  <body>
    <form name="WebIs" action="mailto:stein@upb.de" method="post" enctype="text/plain">
      <p>
        <input type="radio" name="x" value="1"> Radio-Text 1 <br>
        <input type="radio" name="x" value="2" checked="checked"> Radio-Text 2 <br>
      </p>
      <p><input type="submit" name="z" value="Button-Label"> <br></p>
    </form>









  </body>
</html>
```

HTML

Formulare: Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
    <title>Test</title>
  </head>

  <body>
    <form name="WebIs" action="mailto:stein@upb.de" method="post" enctype="text/plain">
      <p>
        <input type="radio" name="x" valu
        <input type="radio" name="x" valu
      </p>
      <p><input type="submit" name="z" va
    </form>

  </body>
</html>
```

HTML

Formulare: Beispiel (Fortsetzung)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
    <title>Test</title>
  </head>

  <body>
    <form name="WebIs" action="mailto:stein@upb.de" method="post" enctype="text/plain">
      <p>
        <input type="radio" name="x" value="1"> Radio-Text 1 <br>
        <input type="radio" name="x" value="2" checked="checked"> Radio-Text 2 <br>
      </p>
      <p><input type="submit" name="z" value="Button-Label"> <br></p>
    </form>

    <form action="http://wwwcs.upb.de/cgi-bin/ag-KlBue/stein/bs-samples/form-sample.cgi">
      <table border="0" cellpadding="0" cellspacing="4">
      <tr>
        <td align="right">Vorname:</td>
        <td><input name="vorname" type="text" size="30" maxlength="30" value="Stefan"></td>
      </tr><tr>
        <td align="right">Nachname:</td>
        <td><input name="nachname" type="text" size="30" maxlength="40"></td>
      </tr>
      </table>
      <p><input type="submit" name="z" value="Abschicken"> <br></p>
    </form>
  </body>
</html>
```

HTML

Formulare: Beispiel (Fortsetzung)

CGI-Script:

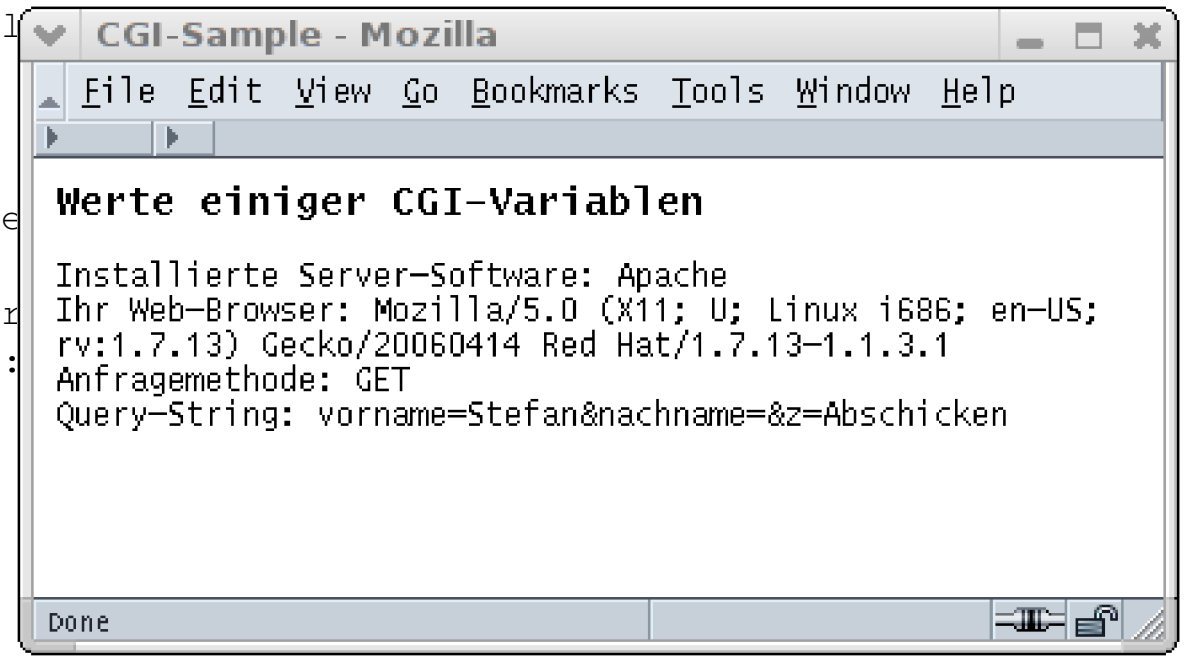
```
#!/bin/csh

echo "content-type: text/html"
echo ""
echo "<html>"
echo "<head>"
echo "<meta http-equiv="\""content-type"\" "content=" ...
echo "<title>CGI-Sample</title>"
echo "</head>"
echo ""
echo "<body>"
echo "<h3>Werte einiger CGI-Variablen</h3>"
echo ""
echo "Installierte Server-Software: " $SERVER_SOFTWARE "<br>"
echo "Ihr Web-Browser: " $HTTP_USER_AGENT "<br>"
echo "Anfragemethode: " $REQUEST_METHOD "<br>"
echo "Query-String: " $QUERY_STRING
echo "</body>"
echo "<html>"
```

HTML

Formulare: Beispiel (Fortsetzung)

CGI-Script:

```
#!/bin/csh

echo "content-type: text/html"
echo ""
echo "<html>"
echo "<head>"
echo "<meta http-equiv="\""content-type"\" "content=" ...
echo "<title>CGI-Sampl
echo "</head>"
echo ""
echo "<body>"
echo "<h3>Werte einige
echo ""
echo "Installierte Ser
echo "Ihr Web-Browser:
echo "Anfragemethode:
echo "Query-String: "
echo "</body>"
echo "<html>"
```



Cascading Stylesheets CSS

Historie

1996 CSS Level 1. Recommendation. [W3C]

2011 CSS Level 2. Recommendation. [W3C]

201x CSS Level 3 (für einige Module als Candidate Recommendation) [W3C]

Mit Cascading Stylesheets ist auch bei HTML-Dokumenten eine Trennung zwischen Inhalt und Darstellung möglich.

Ziele von CSS:

1. leistungsfähige Layout-Definition für HTML-Dokumente
2. Anpassung an verschiedene Ausgabegeräte/-medien
3. zentrales Layout-Management

Bemerkungen:

- Cascading Stylesheets sind ein Kompromiss in dem Sinne, als dass sie *ergänzend* zu den direkten HTML-Formatierungsangaben verwendbar sind. Hintergrund: Gewährleistung der Abwärtskomtabilität zu den existierenden (Milliarden von) Seiten.
- Das Wort „Cascading“ bezieht sich auf die kombinierte Anwendung mehrerer Stylesheets. Konflikte zwischen anwendbaren Layout-Vorgaben werden mit Rücksicht auf Ursprung, Gewichtung und Spezialisierungsgrad gelöst. [W3C] [SELFHTML]
- Die Entwicklung der Cascading Style Sheets geschieht in „Leveln“ (nicht Versionen), die aufeinander aufbauen. Dabei stellen die Features eines höheren Levels eine Übermenge der Features eines niedrigeren Levels dar. Das erlaubte Verhalten für ein Feature in einem höheren Level muss jedoch eingeschränkter bzw. weniger tolerant als in einem niedrigeren Level definiert sein. [W3C]
- Die CSS1 Specification ist veraltet. [W3C]
- Die CSS2.1 Spezifikation definiert CSS Level 2. [W3C]
- CSS-Home und Überblick über die Arbeiten des W3C am CSS-Standard. [W3C 1 2]

Cascading Stylesheets CSS

Einbindung von Stylesheet-Information [SELFHTML]

Die Einbindung bzw. Deklaration von Stylesheet-Information kann auf folgende, miteinander kombinierbare Arten geschehen:

1. Stylesheet-Deklaration in eigener CSS-Datei

2. Stylesheet-Deklaration im HTML-Dokument

3. Styleattribut-Deklaration im Start-Tag einer Elementinstanz

Cascading Stylesheets CSS

Einbindung von Stylesheet-Information [SELFHTML]

Die Einbindung bzw. Deklaration von Stylesheet-Information kann auf folgende, miteinander kombinierbare Arten geschehen:

1. Stylesheet-Deklaration in eigener CSS-Datei

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

   Das `<link>`-Element darf nur im Dokumenten-Head einer HTML-Datei verwendet werden.

2. Stylesheet-Deklaration im HTML-Dokument

   ```
   <style type="text/css">
      /* ... Hier werden die Formate definiert ... */
   </style>
   ```

   Das `<style>`-Element darf in dieser Form nur im Dokumenten-Head einer HTML-Datei verwendet werden.

3. Styleattribut-Deklaration im Start-Tag einer Elementinstanz

Cascading Stylesheets CSS

Einbindung von Stylesheet-Information [SELFHTML]

Die Einbindung bzw. Deklaration von Stylesheet-Information kann auf folgende, miteinander kombinierbare Arten geschehen:

1. Stylesheet-Deklaration in eigener CSS-Datei

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

   Das `<link>`-Element darf nur im Dokumenten-Head einer HTML-Datei verwendet werden.

2. Stylesheet-Deklaration im HTML-Dokument

   ```
   <style type="text/css">
      /* ... Hier werden die Formate definiert ... */
   </style>
   ```

   Das `<style>`-Element darf in dieser Form nur im Dokumenten-Head einer HTML-Datei verwendet werden.

3. Styleattribut-Deklaration im Start-Tag einer Elementinstanz

   ```
   <h3 style="color: red; font: arial">Neues Kapitel</h3>
   ```

   Die Syntax des Attributwertes entspricht dem Deklarationsteil einer CSS-Regel.

Bemerkungen:

- Der Geltungsbereich von Stylesheet-Deklarationen, die aus einer CSS-Datei eingebunden werden, ist global für das HTML-Dokument. Die CSS-Datei muss die Namenserweiterung `.css` haben.
- Der Geltungsbereich von Stylesheet-Deklarationen, die im Dokumenten-Head gemacht werden, ist global: die Deklarationen beziehen sich auf das gesamte HTML-Dokument.
- Der Geltungsbereich von Styleattribut-Deklarationen ist die Elementinstanz selbst einschließlich ihrer Kindelemente.
- Im Konfliktfall haben lokale Deklarationen Vorrang vor globalen.
- Stylesheet-Deklarationen innerhalb eines HTML-Dokuments widersprechen dem Paradigma der Trennung von Inhalt und Darstellung.

Bemerkungen (Fortsetzung) :

- Über das `media`-Attribut im `<link>`-Tag können Medientypen direkt angegeben werden: `screen`, `print`, `aural`, `braille`, `handheld`, `tv`, `tty`, `all`, etc. Je nach Endgerät werden passende Stylesheets ausgewählt.
- CSS-Dateien lassen sich kombinieren:

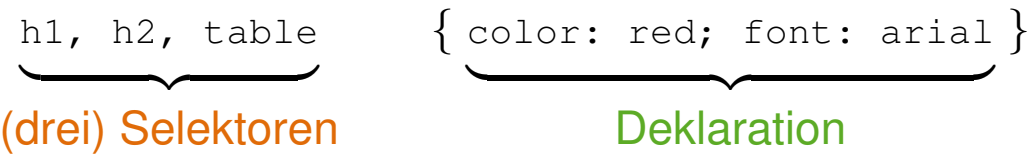
```
<link rel="stylesheet" href="base.css">
<link rel="stylesheet" href="print.css" media="print">
<link rel="stylesheet" href="screen-1-1.css" media="screen">
<link rel="stylesheet" href="screen-1-2.css" media="screen">
<link rel="alternate stylesheet" title="Style 2"
   href="screen-2.css" media="screen">
```

  - `base.css` gilt für alle Medientypen.
  - `print.css` gilt zusätzlich für den Medientyp `print`. Die enthaltenen CSS-Regeln werden z.B. bei der Erzeugung der Druckvorschau berücksichtigt.
  - Für den Medientyp `screen` gelten zusätzlich `screen-1-1.css` und `screen-1-2.css` oder alternativ `screen-2.css`.
- Alternative Stylesheets für den Medientyp `screen` können meist im Browsermenü ausgewählt werden. Als Menüeintrag wird der Text des `title`-Attributs verwendet.

Cascading Stylesheets CSS

Stylesheet-Regeln [SELFHTML]

Ein Stylesheet ist eine Sammlung von Layout-Regeln. Eine Layout-Regel ist wie folgt aufgebaut:



- Mittels des Selektors werden passende Elementinstanzen ausgewählt. Mehrere Selektoren können – durch Kommata getrennt – in einer Liste angegeben sein.
- Der Deklarationsteil enthält – durch Semikola getrennt – die Layout-Vorgaben.

Cascading Stylesheets CSS

Selektoren [W3C]

1. Elementtypselektoren [W3C]
2. Attributselektoren [W3C]
3. Klassenselektoren [W3C]
4. ID-Selektoren [W3C]
5. Pseudo-Klassenselektoren [W3C]
6. Pseudo-Elementselektoren [W3C]
7. Kombinierte Selektoren [W3C]

Cascading Stylesheets CSS

Selektoren [W3C]

1. Elementtypselektoren [W3C]

   ```
   h1 {color: red}                    <h1> ... </h1>
   ```

2. Attributselektoren [W3C]

3. Klassenselektoren [W3C]

4. ID-Selektoren [W3C]

5. Pseudo-Klassenselektoren [W3C]

6. Pseudo-Elementselektoren [W3C]

7. Kombinierte Selektoren [W3C]

Cascading Stylesheets CSS

Selektoren [W3C]

1. Elementtypselektoren [W3C]

```
h1 {color: red}                    <h1> ... </h1>
```

2. Attributselektoren [W3C]

```
h1[hreflang=fr] {color: red}   <h1 hreflang="fr"> ... </h1>
```

3. Klassenselektoren [W3C]

4. ID-Selektoren [W3C]

5. Pseudo-Klassenselektoren [W3C]

6. Pseudo-Elementselektoren [W3C]

7. Kombinierte Selektoren [W3C]

Cascading Stylesheets CSS

Selektoren [W3C]

1. Elementtypselektoren [W3C]

   ```
   h1 {color: red}                    <h1> ... </h1>
   ```

2. Attributselektoren [W3C]

   ```
   h1[hreflang=fr] {color: red}   <h1 hreflang="fr"> ... </h1>
   ```

3. Klassenselektoren [W3C]

   ```
   .Classname {color: red}            <div class="Classname"> ... </div>
   h1.Classname {color: red}          <h1 class="Classname"> ... </h1>
   ```

4. ID-Selektoren [W3C]

5. Pseudo-Klassenselektoren [W3C]

6. Pseudo-Elementselektoren [W3C]

7. Kombinierte Selektoren [W3C]

Cascading Stylesheets CSS

Selektoren [W3C]

1. Elementtypselektoren [W3C]

   ```
   h1 {color: red}                    <h1> ... </h1>
   ```

2. Attributselektoren [W3C]

   ```
   h1[hreflang=fr] {color: red}   <h1 hreflang="fr"> ... </h1>
   ```

3. Klassenselektoren [W3C]

   ```
   .Classname {color: red}          <div class="Classname"> ... </div>
   h1.Classname {color: red}        <h1 class="Classname"> ... </h1>
   ```

4. ID-Selektoren [W3C]

   ```
   #Identifier {color: red}         <h1 id="Identifier"> ... </h1>
   ```

5. Pseudo-Klassenselektoren [W3C]

6. Pseudo-Elementselektoren [W3C]

7. Kombinierte Selektoren [W3C]

Cascading Stylesheets CSS

Selektoren [W3C]

1. Elementtypselektoren [W3C]

```
h1 {color: red}                    <h1> ... </h1>
```

2. Attributselektoren [W3C]

```
h1[hreflang=fr] {color: red}   <h1 hreflang="fr"> ... </h1>
```

3. Klassenselektoren [W3C]

```
.Classname {color: red}        <div class="Classname"> ... </div>
h1.Classname {color: red}      <h1 class="Classname"> ... </h1>
```

4. ID-Selektoren [W3C]

```
#Identifier {color: red}       <h1 id="Identifier"> ... </h1>
```

5. Pseudo-Klassenselektoren [W3C]

```
a:visited {color: red}         <a> ... </a>
```

6. Pseudo-Elementselektoren [W3C]

7. Kombinierte Selektoren [W3C]

Cascading Stylesheets CSS

Selektoren [W3C]

1. Elementtypselektoren [W3C]

```
h1 {color: red}                    <h1> ... </h1>
```

2. Attributselektoren [W3C]

```
h1[hreflang=fr] {color: red}   <h1 hreflang="fr"> ... </h1>
```

3. Klassenselektoren [W3C]

```
.Classname {color: red}        <div class="Classname"> ... </div>
h1.Classname {color: red}      <h1 class="Classname"> ... </h1>
```

4. ID-Selektoren [W3C]

```
#Identifier {color: red}       <h1 id="Identifier"> ... </h1>
```

5. Pseudo-Klassenselektoren [W3C]

```
a:visited {color: red}         <a> ... </a>
```

6. Pseudo-Elementselektoren [W3C]

```
p::first-line {color: red}     <p> ... </p>
```

7. Kombinierte Selektoren [W3C]

Cascading Stylesheets CSS

Selektoren [W3C]

1. Elementtypselektoren [W3C]

```
h1 {color: red}                    <h1> ... </h1>
```

2. Attributselektoren [W3C]

```
h1[hreflang=fr] {color: red}       <h1 hreflang="fr"> ... </h1>
```

3. Klassenselektoren [W3C]

```
.Classname {color: red}            <div class="Classname"> ... </div>
h1.Classname {color: red}          <h1 class="Classname"> ... </h1>
```

4. ID-Selektoren [W3C]

```
#Identifier {color: red}           <h1 id="Identifier"> ... </h1>
```

5. Pseudo-Klassenselektoren [W3C]

```
a:visited {color: red}             <a> ... </a>
```

6. Pseudo-Elementselektoren [W3C]

```
p::first-line {color: red}         <p> ... </p>
```

7. Kombinierte Selektoren [W3C]

```
h1 em {color: red}                 <h1>This is <em>very</em> ... </h1>
```

Cascading Stylesheets CSS

Selektoren (Fortsetzung)

Mit Hilfe von Klassenselektoren kann die Einführung neuer Elementtypen nachempfunden werden.

Besonders geeignet sind die funktionslosen HTML-Elementtypen:

1. Block-Elementtyp `<div>`
2. Inline-Elementtyp `<span>`

Cascading Stylesheets CSS

Selektoren (Fortsetzung)

Mit Hilfe von Klassenselektoren kann die Einführung neuer Elementtypen nachempfunden werden.

Besonders geeignet sind die funktionslosen HTML-Elementtypen:

1. Block-Elementtyp `<div>`
2. Inline-Elementtyp `<span>`

Beispiel:

```
<head>
  <title>Example</title>
  <style type="text/css">
    .myheading {font-family: sans-serif; color: blue}
  </style>
</head>
<body>
  <div class="myheading">Ein eigener Titelstil</div>
  ...
```

Bemerkungen:

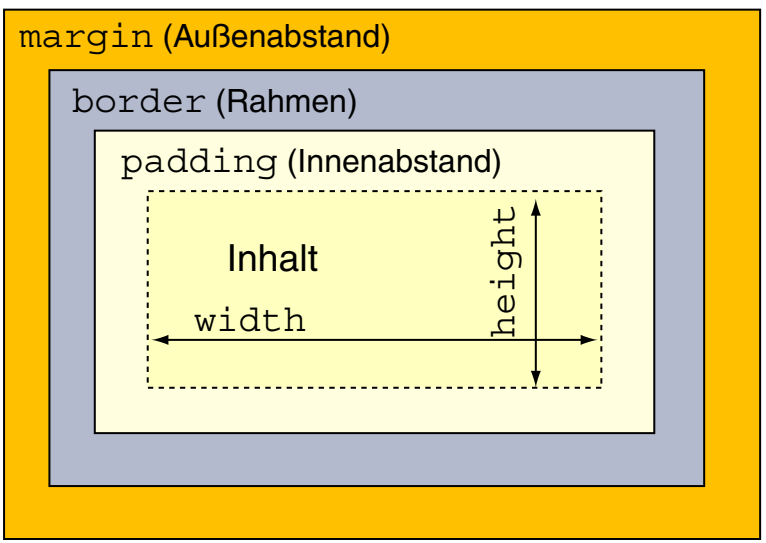
- HTML verwendet eine fixe Dokumentstruktur und somit sind alle Elementtypen vorgegeben; diese HTML-Elementtypen besitzen elementtypspezifische Vorgaben für ihre Darstellung.
- Die Schaffung von Elementinstanzen, die zu einer gemeinsamen Stylesheet-Klasse gehören und darüberhinaus ohne weitere Funktion (= intendierte Semantik) sind, geschieht in zwei Schritten:
  1. Definition einer neuen Stylesheet-Klasse mittels `.`*Classname* `{...}`.
  2. Verwendung der neuen Stylesheet-Klasse mittels `class`="*Classname*" in Elementinstanzen des Typs `<div>` oder `<span>`.
- Das W3C warnt vor der übertriebenen Nutzung dieser Möglichkeit, weil die intendierte Semantik selbstdefinierter Klassen für Außenstehende oft nicht erkennbar ist. [W3C]

Cascading Stylesheets CSS

Deklarationen: allgemein

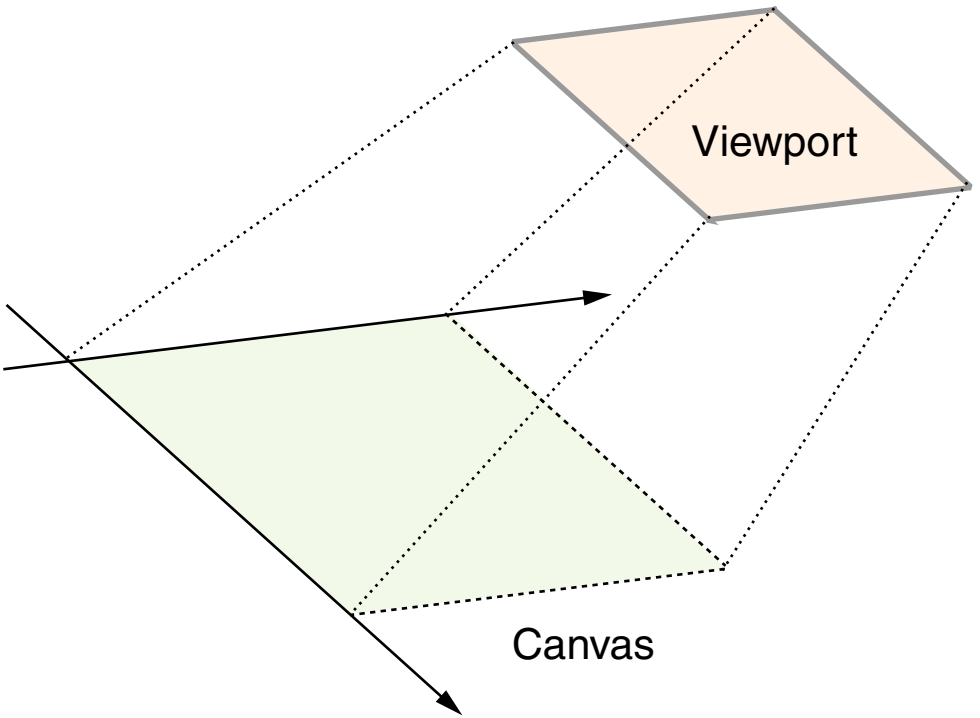
CSS-Deklarationen können sich auf nahezu alle Aspekte der Gestaltung beziehen:

- Schrift [SELFHTML]
  `font-family`, `font-style`, `font-weight`, `font-size`, `text-decoration`, `color`, etc.
- Layout von Bildern
- Listen- und Tabellendarstellung
- Maßeinheiten, Farben [SELFHTML]
- Abstände, Box-Modell [SELFHTML]



Cascading Stylesheets CSS

Deklarationen: Viewport-Modell



Über einen Viewport wird der Zugriff auf ein Dokument ermöglicht. Das Layout erfolgt auf der Zeichenfläche, Positionen können jedoch fest an den Viewport gebunden werden.

Cascading Stylesheets CSS

Deklarationen: Layout-Constraints

- Property `display` [W3C]

`block`	Element erzeugt eine Block-Box.
`inline`	Element erzeugt eine oder mehrere Inline-Boxen.
`inline-block`	Element erzeugt eine nicht teilbare Inline-Box, der Inhalt wird im Block-Kontext formatiert.

- Property `position` [W3C]

`static`	Box wird im vorliegenden Kontext (Block/Inline) gesetzt.
`relative`	Box wird mit Offset relativ zur normalen Position gesetzt.
`absolute`	Box wird an absoluter Position im enthaltenen Block gesetzt. (i.d.R. absolute Position auf Zeichenfläche für Kindelemente des `<body>`-Elementes)
`fixed`	Box wird an absoluter Position im enthaltenen Block gesetzt, aber in der Viewport-Ebene. (Position fest im Viewport)

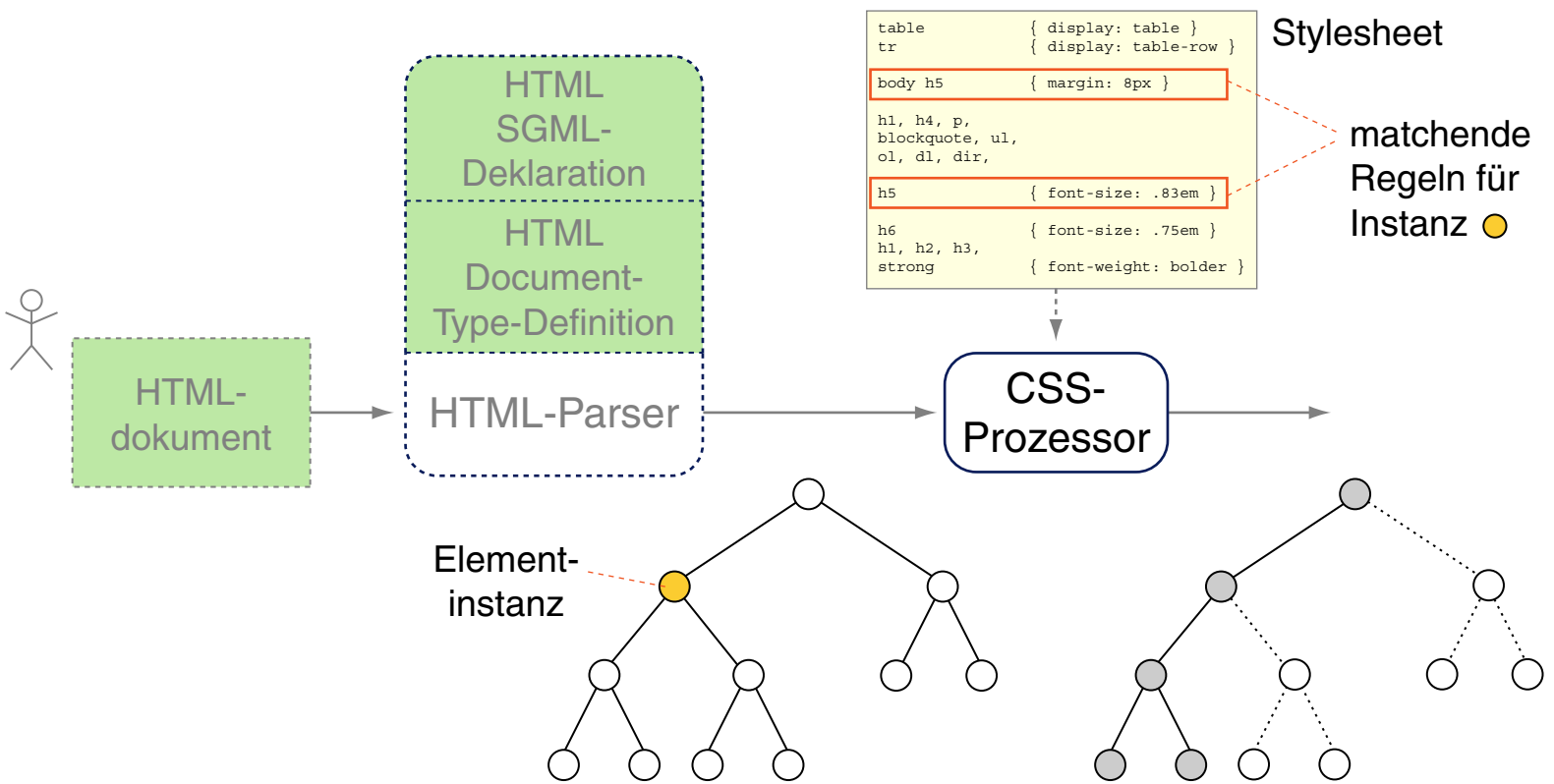
- Property `top, right, bottom, left` [W3C]
  Angabe der Position als Abstand vom jeweiligen Rand der enthaltenden Box (absolute/fixed) oder von der vorgesehenen Position der Box (relative).

`length`	absolute Abstandsangabe
`percentage`	relative Abstandsangabe

Cascading Stylesheets CSS

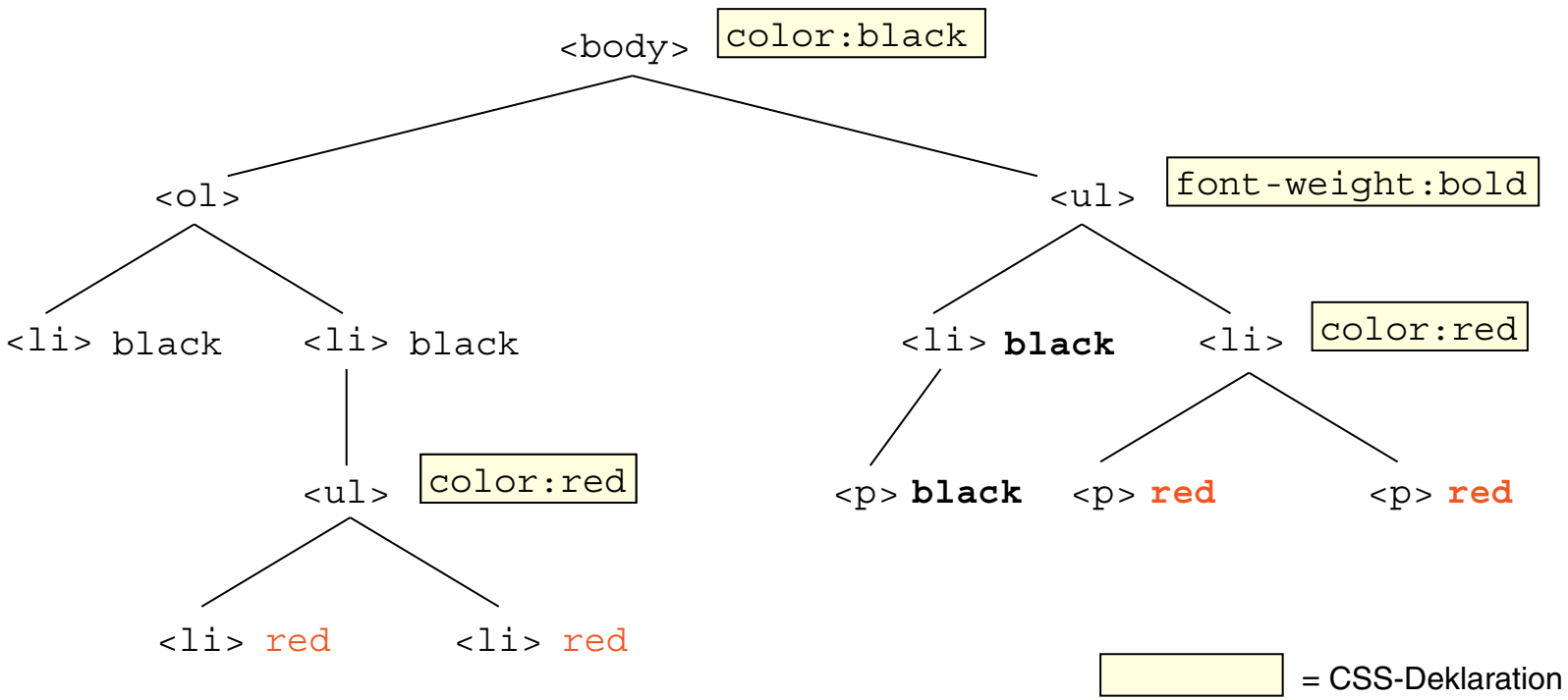
Verarbeitungsstrategie [XSL-Verarbeitung]

Bei der Darstellung einer Elementinstanz werden in den zugehörigen Stylesheets anhand der Selektoren die matchenden Regeln identifiziert. Ihre Anwendung geschieht in der Reihenfolge von den weniger zu den mehr spezifischen.



Cascading Stylesheets CSS

Verarbeitungsstrategie (Fortsetzung)



- Regeln, die für einen bestimmten Elementtyp deklariert sind, werden an eingebettete Elementinstanzen vererbt.
- Vererbte Werte und Defaults werden durch lokale Vorgaben überschrieben:
  `color:black→red`, `font-weight:normal→bold`

Cascading Stylesheets CSS

Verarbeitungsstrategie (Fortsetzung) [W3C] [SELFHTML]

Bei der Darstellung von HTML-Dokumenten sind drei Arten von Stylesheets beteiligt:

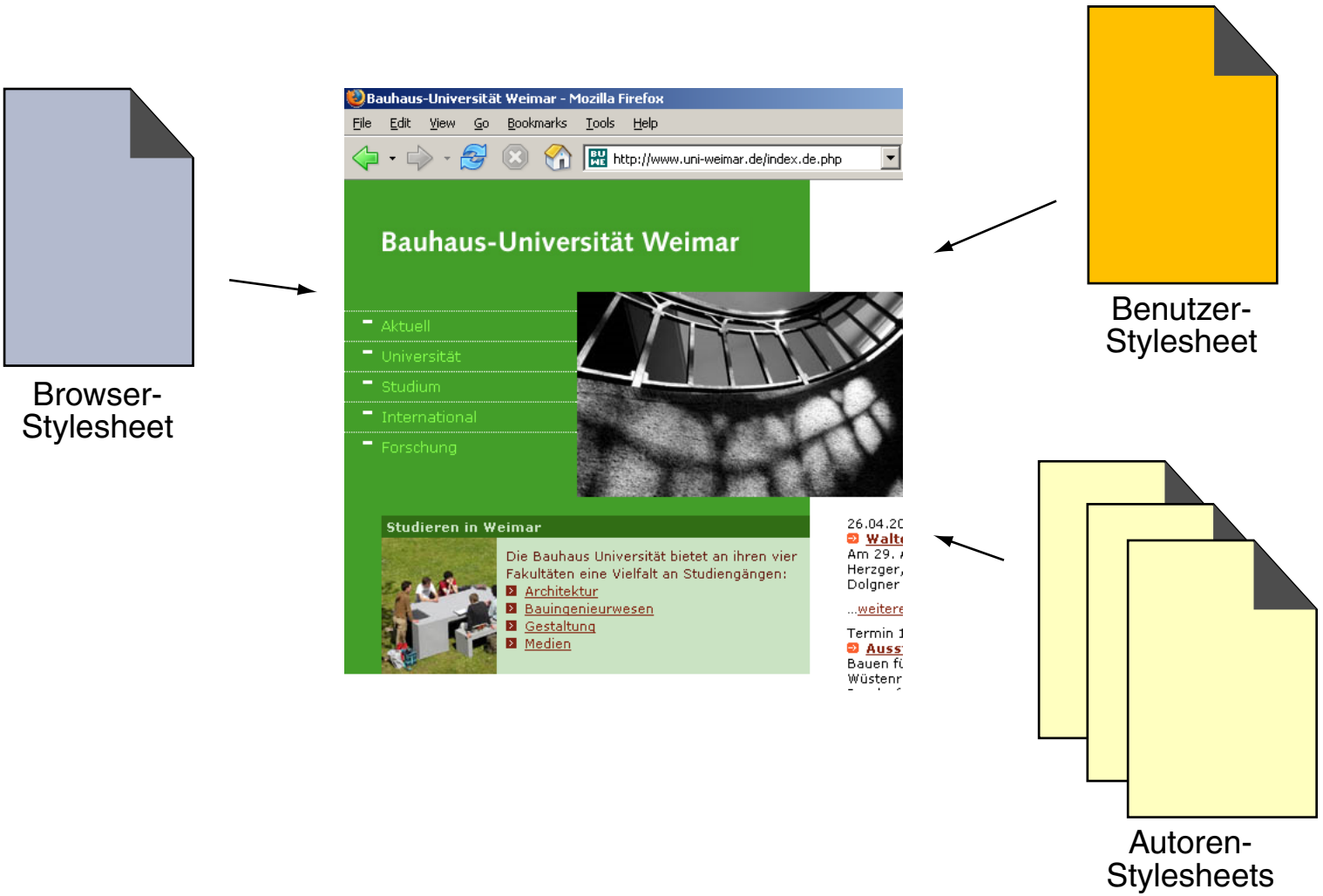
1. Browser-Stylesheet
   Definiert das Standard-Layout für die Elementinstanzen; dieses Stylesheet ist Browser-spezifisch und wird vom Browser-Hersteller entwickelt. Ein entsprechender W3C-Vorschlag befindet hier [W3C].

2. Benutzer-Stylesheet
   Definiert die Präferenzen eines Benutzers. Die Spezifikation des Benutzer-Stylesheets geschieht über einen Browser-Dialog.

3. Autoren-Stylesheet(s)
   Die „eigentlichen“ (sichtbaren) Stylesheets, die ein Autor eines HTML-Dokuments zur Realisierung seiner Layout-Ziele entwickelt hat.

Cascading Stylesheets CSS

Verarbeitungsstrategie (Fortsetzung)



Bemerkungen:

- Regeln können durch Angabe von `!important` stärker gewichtet werden:

```
p {
    font-style: italic !important
    color: red;
  }
```

  Im Beispiel ist das Setzen der Property `font-style` stärker gewichtet, das Setzen der Property `color` nicht. Eine Gewichtung ist nur für Autoren- und Benutzer-Stylesheets spezifizierbar.

- Konflikte zwischen anwendbaren Layout-Vorgaben werden zunächst mit Rücksicht auf Ursprung und Gewichtung gelöst [W3C] [SELFHTML] :

  1. `!important`-Regeln aus Benutzer-Stylesheet
  2. `!important`-Regeln aus Autoren-Stylesheets
  3. normale Regeln aus Autoren-Stylesheets
  4. normale Regeln aus Benutzer-Stylesheet
  5. Regeln aus Browser-Stylesheet

  Bestehen danach immer noch Konflikte, so werden diese mit Rücksicht auf

  6. den Spezialisierungsgrad (speziellere Regeln vor allgemeineren) und dann
  7. der Reihenfolge (spätere Regeln vor früheren) gelöst.

HTML, CSS

Quellen zum Nachlernen und Nachschlagen im Web

- WHATWG. *HTML, Living Standard.*
  www.whatwg.org
- D. Raggett, A. Le Hors, I. Jacobs. *HTML 4.01 Specification.*
  www.w3.org/TR/html401
- S. Münz. *SELFHTML.*
  www.selfhtml.org, de.selfhtml.org
- W3 Schools. *HTML.*
  www.w3schools.com/html
- S. Pemberton et. al. *XHTML 1.0, Second Edition.*
  www.w3.org/TR/xhtml1

HTML, CSS

Quellen zum Nachlernen und Nachschlagen im Web (Fortsetzung)

- W3C. *Cascading Style Sheets Home Page.*
  www.w3.org/Style/CSS
- B. Bos et. al. *Cascading Style Sheets, Level 2.*
  www.w3.org/TR/REC-CSS2
- W3 Schools. *CSS.*
  www.w3schools.com/css

HTML, CSS

Quellen zum Nachlernen und Nachschlagen im Web (Fortsetzung)

- W3C. *Markup Validation Service.*
  validator.w3.org
- D. Raggett. *HTML Tidy.* (standardisieren und säubern von HTML-Code)
  w3c.github.io/tidy-html5, www.w3.org/People/Raggett/tidy
- V. Flanders. *Web Pages That Suck.* (Web-Design)
  www.webpagesthatsuck.com

Kapitel WT:III (Fortsetzung)

XML-Grundlagen

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
    <title>Person</title>
  </head>

  <body>
    <h3>Alan Turing</h3>
    <p>
      23. Juni 1912* <br>
      Mathematiker, Informatiker
    </p>
  </body>
</html>
```

XML-Grundlagen

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
    <title>Person</title>
  </head>

  <body>
    <h3>Alan Turing</h3>
    <p>
      23. Juni 1912* <br>
      Mathematiker, Informatiker
    </p>
  </body>
</html>
```

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE person SYSTEM "person.dtd">

<person>
  <name>
    <vorname>Alan</vorname>
    <nachname>Turing</nachname>
  </name>

  <geburtstag>23. Juni 1912</geburtstag>

  <beruf>Mathematiker</beruf>
  <beruf>Informatiker</beruf>
</person>
```

XML-Grundlagen

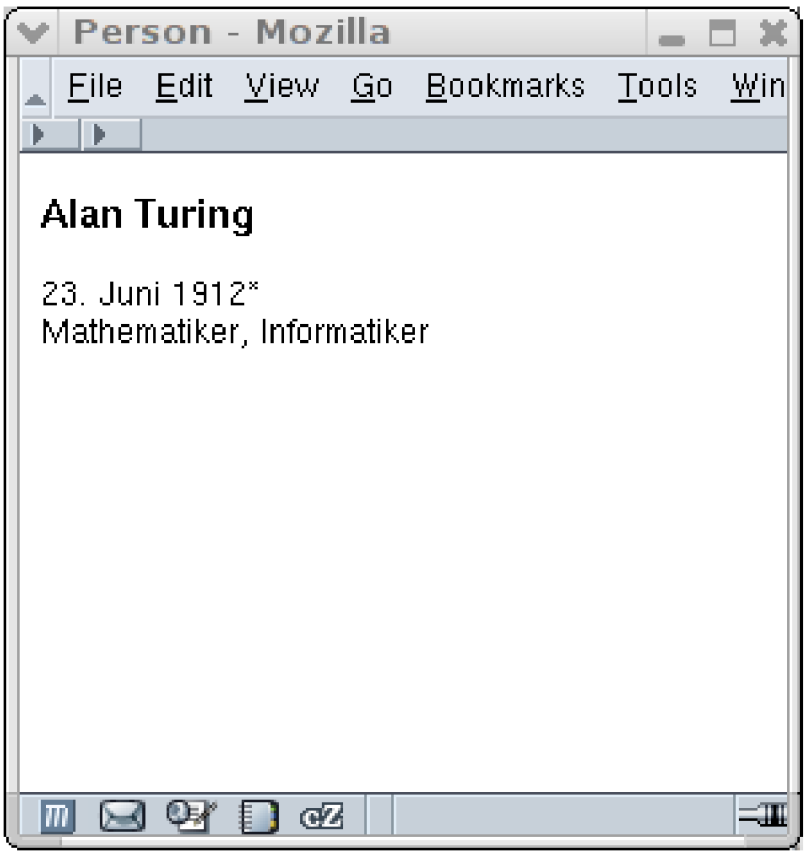
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
    <title>Person</title>
  </head>

  <body>
    <h3>Alan Turing</h3>
    <p>
      23. Juni 1912* <br>
      Mathematiker, Informatiker
    </p>
  </body>
</html>
```

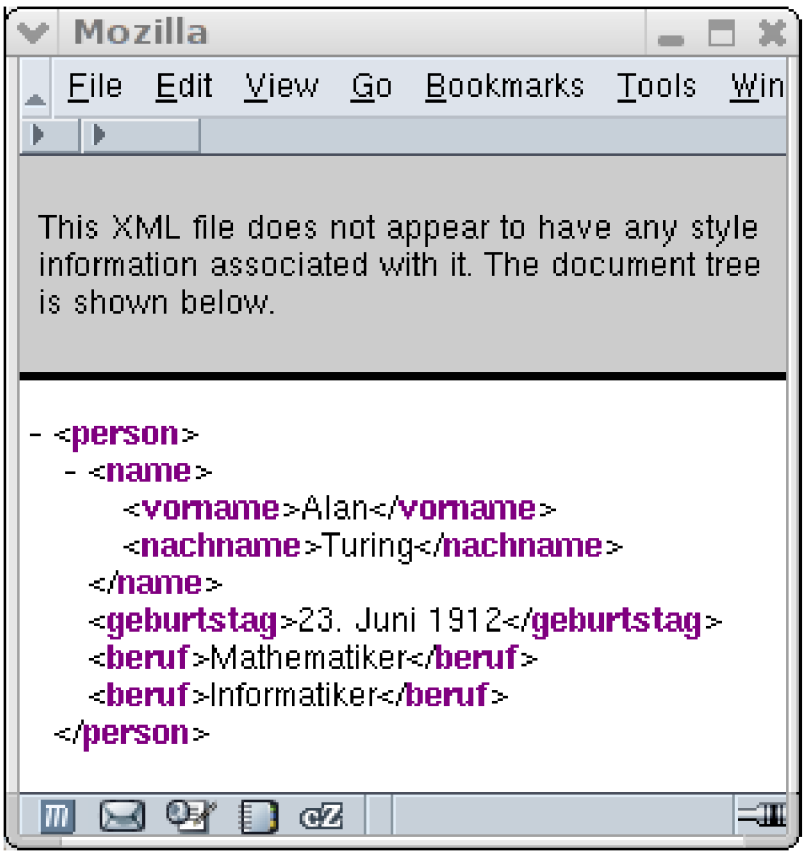
```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE person SYSTEM "person.dtd">

<person>
  <name>
    <vorname>Alan</vorname>
    <nachname>Turing</nachname>
  </name>

  <geburtstag>23. Juni 1912</geburtstag>

  <beruf>Mathematiker</beruf>
  <beruf>Informatiker</beruf>
</person>
```

Keine operationale Semantik:

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE person SYSTEM "person.dtd">

<person>
  <address>
    <zip>Alan</zip>
    <city>Turing</city>
  </address>

  <weight>23. Juni 1912</weight>

  <name>Mathematiker</name>
  <name>Informatiker</name>
</person>
```

XML-Grundlagen





In XML können beliebige Elementtypen deklariert werden. Mittels einer DTD *(Document Type Definition)* lassen sich strukturelle Constraints (= Inhaltsmodelle) für die Verwendung von Elementinstanzen vorschreiben.

XML-Grundlagen

Historie

1998 XML 1.0. Von Anfang an ein riesiger Erfolg. [W3C]

2001 XML-Schema 1.0. Schemasprache, die DTDs ablösen soll. [W3C]

2003 MathML 2.0. Markup-Sprache für mathematische Notation. [W3C]

2003 SVG 1.1. Beschreibungssprache für zweidimensionale Vektorgrafiken. [W3C]

2004 RDF. Formale Beschreibung von Metainformation. [W3C]

2004 XML 1.1. Erweiterung auf neue Schriftfamilien wie z.B. Mongolisch. [W3C]

2005 Atom Syndication Format. Markup-Sprache für News-Feeds. [RFC 4287]

2007 WSDL 2.0. Beschreibungssprache für Web-Services. [W3C]

2007 XSLT 2.0. Transformation von XML-Dokumenten. [W3C]

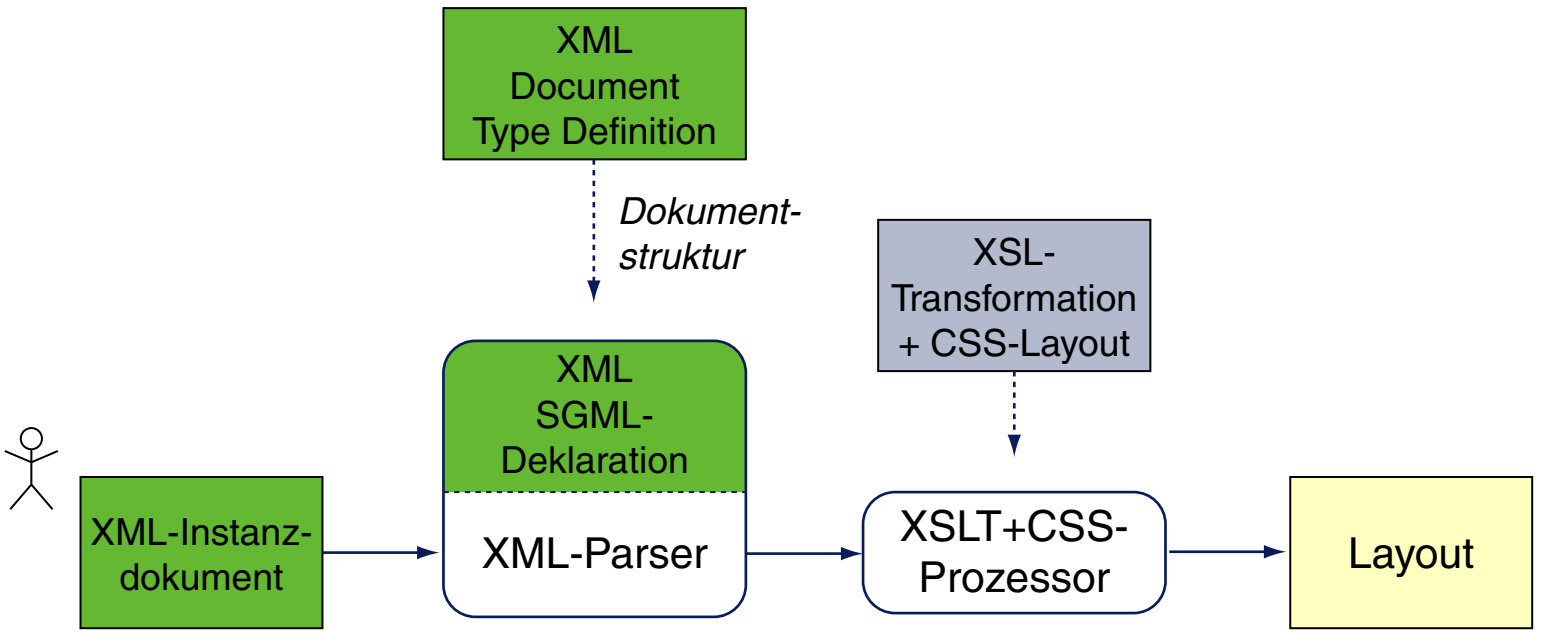
2008 SMIL 3.0. Zeitliche Integration multimedialer Datenobjekte. [W3C]

Bemerkungen:

- Viele Entwickler forderten eine erweiterbare Markup-Sprache, die weniger kompliziert als SGML ist. Tatsächlich ist die Spezifikation von XML deutlich einfacher und kürzer als die von SGML.
- XML-Dokumente werden nicht nur in Web-Anwendungen genutzt. Aufgrund seiner Flexibilität kann XML den Publishing-Anforderungen von Büchern, Zeitschriften, Katalogen, Flugblättern etc. gerecht werden.

XML-Grundlagen

XML Dokumentenverarbeitung



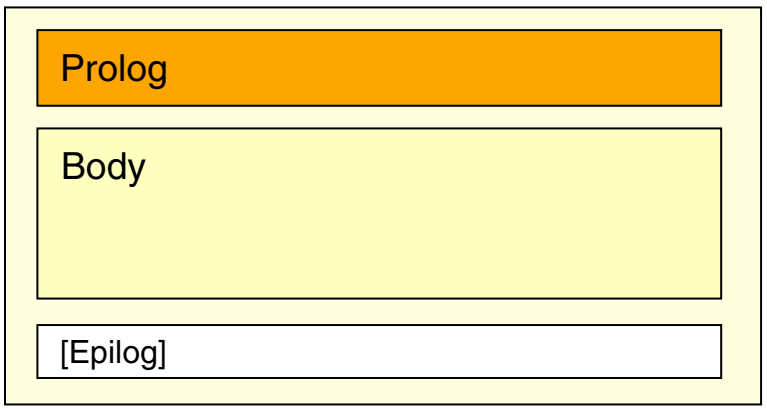
Vergleiche hierzu

- die SGML Dokumentenverarbeitung
- und die HTML Dokumentenverarbeitung.

XML-Grundlagen

XML-Dokument

XML-Dokument




```
<?xml version="1.0"?>
<!DOCTYPE person ...>

<person>
  <name>
    <vorname>Alan</vorname>
    <nachname>...
  </name>
</person>
```

- Zum Prolog zählt alles vor dem Start des XML-Wurzelelementes; der Prolog enthält Meta-Informationen über das Dokument.
- Der Body besteht aus ineinander geschachtelten XML-Elementen.
- Der Epilog enthält Kommentare und Verarbeitungsanweisungen für das Dokument; er ist optional.
- Vergleiche hierzu die HTML-Dokumentstruktur.

XML-Grundlagen

Dokumenten-Prolog

1. XML-Deklaration.

   ```
   <?xml version="1.0" encoding="UTF-8" standalone="no"?>
   ```

   Jeder XML-Prolog startet mit einer XML-Deklaration.

XML-Grundlagen

Dokumenten-Prolog

1. XML-Deklaration.

   ```
   <?xml version="1.0" encoding="UTF-8" standalone="no"?>
   ```

   Jeder XML-Prolog startet mit einer XML-Deklaration.

2. XML-Dokumenttyp-Deklaration.

   ```
   <!DOCTYPE Wurzelelementname SYSTEM "URI">
   ```

   Anstelle oder kombiniert mit der Referenz auf eine Datei lassen sich DTD-Befehle auch im Dokument einbinden. Man spricht von der externen bzw. der internen DTD-Teilmenge *(external subset, internal subset)*; zusammen bilden sie die vollständige DTD.

   ```
   <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
   <!DOCTYPE poem [
     <!ELEMENT poem (#PCDATA)>
     <!ENTITY author "William Shakespeare">
   ]>
   <poem>
     Dieses Gedicht stammt von &author;
   </poem>
   ```

XML-Grundlagen

Dokumenten-Prolog

1. XML-Deklaration.

   ```
   <?xml version="1.0" encoding="UTF-8" standalone="no"?>
   ```

   Jeder XML-Prolog startet mit einer XML-Deklaration.

2. XML-Dokumenttyp-Deklaration.

   ```
   <!DOCTYPE Wurzelelementname SYSTEM "URI">
   ```

   Anstelle oder kombiniert mit der Referenz auf eine Datei lassen sich DTD-Befehle auch im Dokument einbinden. Man spricht von der externen bzw. der internen DTD-Teilmenge *(external subset, internal subset)*; zusammen bilden sie die vollständige DTD.

   ```
   <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
   <!DOCTYPE poem [
     <!ELEMENT poem (#PCDATA)>
     <!ENTITY author "William Shakespeare">
   ]>
   <poem>
     Dieses Gedicht stammt von &author;
   </poem>
   ```

3. Verarbeitungsanweisung für XML-Stylesheets.

   ```
   <?xml-stylesheet type="text/css" href="person.css"?>
   ```

Bemerkungen:

- `standalone="no"` bedeutet, dass der Rückgriff auf externe DTD erforderlich ist.
- Die Dokumenttyp-*Deklaration* enthält eine Referenz auf (= deklariert) eine DTD *(Document Type Definition)*, die wiederum Deklarationen für Elemente und Attribute enthält.
- Die Dateiendung einer DTD-Datei ist `.dtd`.
- Die Dokumenttyp-Deklaration kann entfallen.
- Bei einer weltweit bekannten DTD kann anstelle des Schlüsselwortes `SYSTEM` das Schlüsselwort `PUBLIC` zusammen mit dem Public-Identifier dieser DTD verwendet werden; der Public-Identifier wird durch einen lokalen Katalog-Server auf eine URL abgebildet. Sicherheitshalber ist zusätzlich noch eine URI anzugeben. In der Praxis werden Public-Identifier kaum verwendet.

XML-Grundlagen

Dokumenten-Body

Allgemeine Form einer XML-Elementinstanz:

`<elementName Attributliste>` *eigentlicher Inhalt* `</elementName>`

Tags müssen balanciert sein – Ausnahme sind *Empty-Element-Tags*:

`<elementName/>`, äquivalente Schreibweise: `<elementName></elementName>`

XML-Grundlagen

Dokumenten-Body

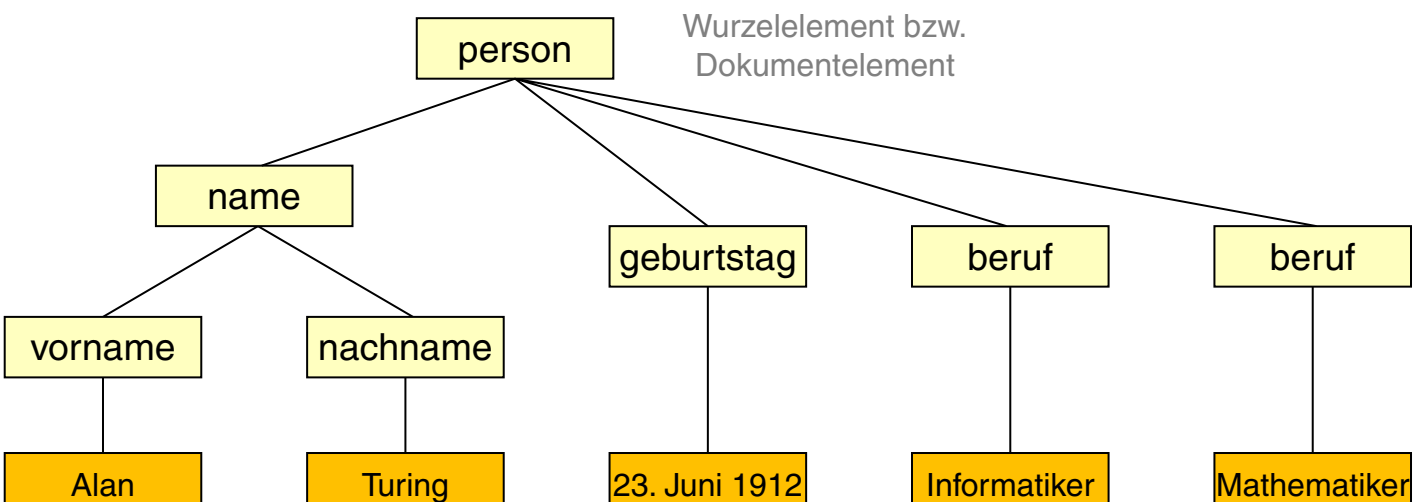
Allgemeine Form einer XML-Elementinstanz:

`<elementName Attributliste>` *eigentlicher Inhalt* `</elementName>`

Tags müssen balanciert sein – Ausnahme sind *Empty-Element-Tags*:

`<elementName/>`, äquivalente Schreibweise: `<elementName></elementName>`

Die Elementstruktur des Bodies entspricht einem Baum:



XML-Grundlagen

Attribute

Verwendung von Attributen wie in SGML [SGML] :

```
<person geboren="1912-06-23" gestorben="1954-06-07">
  Alan Turing
</person>
```

Frage des Stils: Elementmodellierung (a) oder Attributmodellierung (b)

(a)
```
<person>
  <name>
    <vorname>Alan</vorname>
    <nachname>Turing</nachname>
  </name>
  <beruf>Mathematiker</beruf>
  <beruf>Informatiker</beruf>
</person>
```

(b)
```
<person>
  <name vorname="Alan" nachname="Turing"/>
  <beruf wert="Mathematiker"/>
  <beruf wert="Informatiker"/>
</person>
```

Bemerkungen:

- Notiert man Empty-Element-Tags anstatt `<elementName ...>` als `<elementName ...></elementName>`, so darf kein Whitespace (Leerzeichen, Tabulatorzeichen, Zeilenvorschub) zwischen dem Start-Tag und dem Ende-Tag stehen.
- Zum Modellierungsstil: mit Hilfe von Attributen sollten nur Meta-Daten über eine Elementinstanz spezifiziert werden. Deshalb ist im vorigen Beispiel die Elementmodellierung (a) vorzuziehen.
- Eine eindeutige Trennung zwischen Objektdaten und Meta-Daten ist nicht immer möglich.

XML-Grundlagen

Weitere Regeln zur Syntax

- XML-Namen dürfen aus beliebigen alphanumerischen, ideographischen sowie den drei Interpunktionszeichen „_“ , „–“ und „.“ bestehen.
- Entity-Referenzen wie in SGML: `&`*Entity-Name*`;` [SGML]
- Kommentare: `<!-- Dies ist ein Kommentar -->`

XML-Grundlagen

Weitere Regeln zur Syntax

- XML-Namen dürfen aus beliebigen alphanumerischen, ideographischen sowie den drei Interpunktionszeichen „_“ , „–“ und „.“ bestehen.
- Entity-Referenzen wie in SGML: `&`*Entity-Name*`;` [SGML]
- Kommentare: `<!-- Dies ist ein Kommentar -->`
- Die CDATA-Deklaration ermöglicht die literale Verwendung aller Zeichen:

```
<![CDATA[
  <svg xmls="http://www.w3.org./2000/svg"
    width="12cm" height="10cm">
  <ellipse rx="110" ry="130" />
  ...
]]>
```

- Verarbeitungsanweisungen werden mit `<?` und `?>` eingeschlossen:

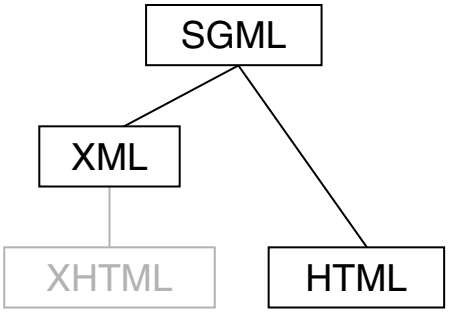
```
<?php
  mysql_connect("database.unc.edu", "clerk", "password");
  ...
?>
```

Bemerkungen:

- Ideographische Zeichen, auch Bildzeichen genannt, sind Zeichen, die eine unmittelbare Interpretation besitzen. Sie können sprachunabhängig als auch sprachspezifisch sein. Beispiele sind mathematische Zeichen, chinesische Schriftzeichen oder Logogramme.
- Kommentare und Verarbeitungsanweisungen sind Markup, aber keine Elementinstanzen. Sie dürfen überall im Dokument – jedoch nicht *in* einem Tag stehen.
- Das Schlüsselwort `#CDATA` bezeichnet den Datentyp *Character Data*. Abschnitte diesen Datentyps werden durch den Parser nicht analysiert. Innerhalb eines CDATA-Abschnitts wird nur die Zeichenkette „`]]>`“ als Markup interpretiert; sie markiert das CDATA-Ende-Tag. [SELFHTML]
- Das Schlüsselwort `#PCDATA` bezeichnet den Datentyp *Parsed Character Data*. Abschnitte diesen Datentyps werden durch den Parser analysiert. Innerhalb eines PCDATA-Abschnitts müssen deshalb Zeichen, die Bestandteil der HTML-Markup-Syntax sind (<, >, etc.) maskiert werden, wenn sie nicht als Markup interpretiert werden sollen. Es sind alle Arten von Zeichen, Entity-Referenzen, CDATA-Abschnitte, Kommentare und Verarbeitungsanweisungen zugelassen, jedoch keine Elementinstanzen. Bei PCDATA handelt es sich üblicherweise um Text, der zwischen dem Anfang- und dem Ende-Tag einer Elementinstanz notiert wird. [SELFHTML]

XML-Grundlagen

Zusammenhang XHTML, HTML



XHTML:

- MIME-Type `text/html` oder `application/xml`
- Dateisuffix `.xhtml`
- XML-Deklaration obligatorisch
- Namensraumangabe obligatorisch
- Dateigrundgerüst enthält immer Namespace, Head und Body
- Case-sensitiv. Insbesondere werden alle Sprachelemente kleingeschrieben.
- balancierte Tags oder Closing-Delimiter
- keine Attribute ohne Zuweisung
- `id`-Tag identifiziert Hyperlink-Ziel

HTML:

- MIME-Type `text/html`
- Dateisuffix `.html` oder `.htm`
- historisch gewachsenes Inhaltsmodell: Beschränkung hinsichtlich der Verschachtelung bestimmter Elemente

XML-Grundlagen

Wohlgeformtheit und Validität

XML-Dokumente müssen wohlgeformt sein:

1. balancierte und geschachtelte (unverschränkte) Tags
2. genau ein Wurzelelement
3. Attributnamen eindeutig pro Element, Wertzuweisungen müssen in Anführungszeichen stehen
4. keine Kommentare und Verarbeitungsanweisungen in Tags

(es gibt noch mehr)

XML-Grundlagen

Wohlgeformtheit und Validität

XML-Dokumente müssen wohlgeformt sein:

1. balancierte und geschachtelte (unverschränkte) Tags
2. genau ein Wurzelelement
3. Attributnamen eindeutig pro Element, Wertzuweisungen müssen in Anführungszeichen stehen
4. keine Kommentare und Verarbeitungsanweisungen in Tags

(es gibt noch mehr)

XML-Dokumente können valide (gültig) sein. Das heißt,

1. das Dokument ist wohlgeformt,
2. das Dokument enthält eine DTD oder eine Referenz darauf, und
3. der gesamte Dokumenteninhalt ist konform mit der DTD.

XML-Grundlagen

XML Document Type Definition

XML-Dokument:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE person SYSTEM "person.dtd">

<person>
  <name>
    <vorname>Alan</vorname>
    <nachname>Turing</nachname>
  </name>
  <geburtstag>23. Juni 1912</geburtstag>
  <beruf>Mathematiker</beruf>
  <beruf>Informatiker</beruf>
</person>
```

Zugehörige XML Document Type Definition:

```
<!ELEMENT person (name, geburtstag, beruf+)>
<!ELEMENT name (vorname, nachname)>
<!ELEMENT vorname (#PCDATA)>
<!ELEMENT nachname (#PCDATA)>
<!ELEMENT geburtstag (#PCDATA)>
<!ELEMENT beruf (#PCDATA)>
```

XML-Grundlagen

XML Document Type Definition

XML-Dokument:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE person SYSTEM "person.dtd">

<person>
  <name>
    <vorname>Alan</vorname>
    <nachname>Turing</nachname>
  </name>
  <geburtstag>23. Juni 1912</geburtstag>
  <beruf>Mathematiker</beruf>
  <beruf>Informatiker</beruf>
</person>
```

Zugehörige XML Document Type Definition:

```
<!ELEMENT person (name, geburtstag, beruf+)>
<!ELEMENT name (vorname, nachname)>
<!ELEMENT vorname (#PCDATA)>
<!ELEMENT nachname (#PCDATA)>
<!ELEMENT geburtstag (#PCDATA)>
<!ELEMENT beruf (#PCDATA)>
```

XML-Grundlagen

XML Document Type Definition

XML-Dokument:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE person SYSTEM "person.dtd">

<person>
  <name>
    <vorname>Alan</vorname>
    <nachname>Turing</nachname>
  </name>
  <geburtstag>23. Juni 1912</geburtstag>
  <beruf>Mathematiker</beruf>
  <beruf>Informatiker</beruf>
</person>
```

Zugehörige XML Document Type Definition:

```
<!ELEMENT person (name, geburtstag, beruf+)>
<!ELEMENT name (vorname, nachname)>
<!ELEMENT vorname (#PCDATA)>
<!ELEMENT nachname (#PCDATA)>
<!ELEMENT geburtstag (#PCDATA)>
<!ELEMENT beruf (#PCDATA)>
```

XML-Grundlagen

XML Document Type Definition

XML-Dokument:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE person SYSTEM "person.dtd">
<person>
  <name>
    <vorname>Alan</vorname>
    <nachname>Turing</nachname>
  </name>
  <geburtstag>23. Juni 1912</geburtstag>
  <beruf>Mathematiker</beruf>
  <beruf>Informatiker</beruf>
</person>
```

Zugehörige XML Document Type Definition:

```
<!ELEMENT person (name, geburtstag, beruf+)>
<!ELEMENT name (vorname, nachname)>
<!ELEMENT vorname (#PCDATA)>
<!ELEMENT nachname (#PCDATA)>
<!ELEMENT geburtstag (#PCDATA)>
<!ELEMENT beruf (#PCDATA)>
```

XML-Grundlagen

XML Document Type Definition (Fortsetzung) [SGML]

Die DTD definiert:

1. Art und Aufbau von Elementtypen für eine Klasse von Dokumenten
   = die Inhaltsmodelle der Elementtypen
2. die in Elementinstanzen verwendbaren Attribute und ihre Datentypen
3. verschiedene Arten von Textkonstanten, sogenannte *Entities*

Closed-World-Semantik: Alles, was nicht erlaubt ist, ist verboten.

XML-Grundlagen

XML Document Type Definition (Fortsetzung) [SGML]

Die DTD definiert:

1. Art und Aufbau von Elementtypen für eine Klasse von Dokumenten
   = die Inhaltsmodelle der Elementtypen
2. die in Elementinstanzen verwendbaren Attribute und ihre Datentypen
3. verschiedene Arten von Textkonstanten, sogenannte *Entities*

Closed-World-Semantik: Alles, was nicht erlaubt ist, ist verboten.

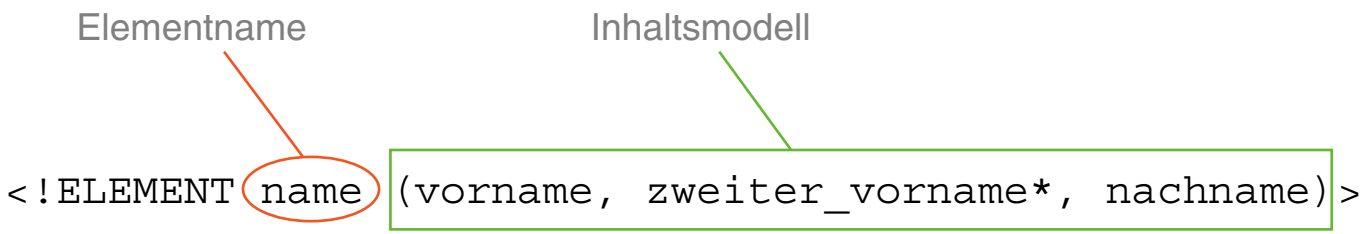
Allgemeine Syntax:

1. `<!ELEMENT` *Elementname Inhaltsmodell* `>`
2. `<!ATTLIST` *Elementname* `{` *Attributname Attributtyp Default-Wert* `}*` `>`
3. `<!ENTITY` *Entity-Name Zeichenkette* `>`

XML-Grundlagen

XML Document Type Definition (Fortsetzung)

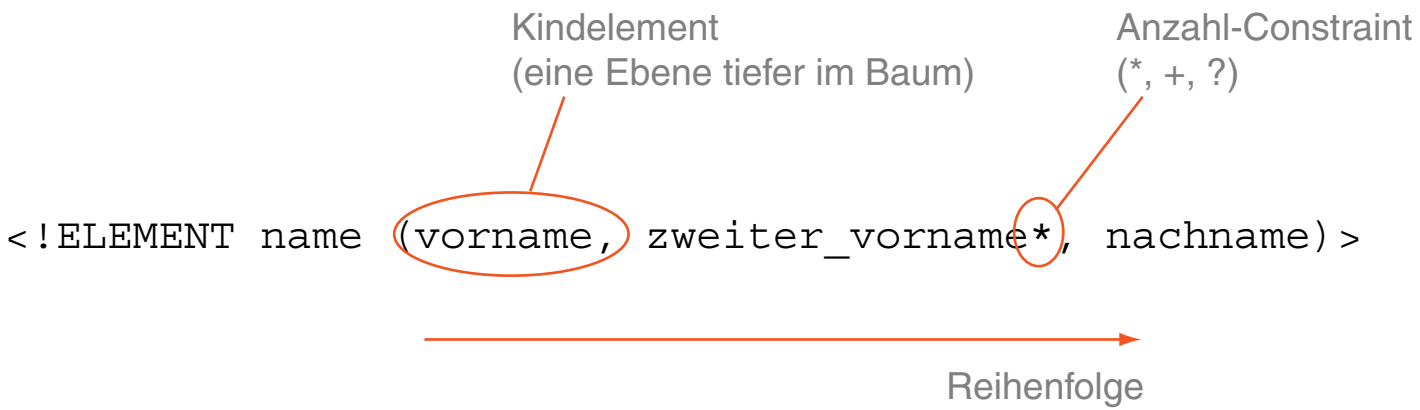
Beispiel für Elementdeklaration:



XML-Grundlagen

XML Document Type Definition (Fortsetzung)

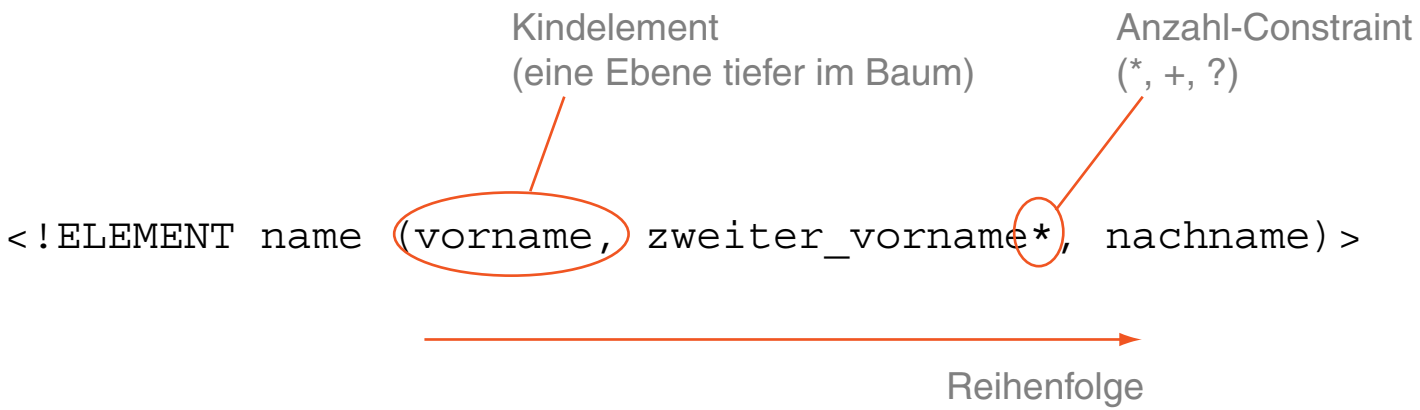
Beispiel für Elementdeklaration:



XML-Grundlagen

XML Document Type Definition (Fortsetzung)

Beispiel für Elementdeklaration:



gültige Elementinstanz:

```
<name>
  <vorname>Alan</vorname>
  <nachname>Turing</nachname>
</name>
```

ungültige Elementinstanzen:

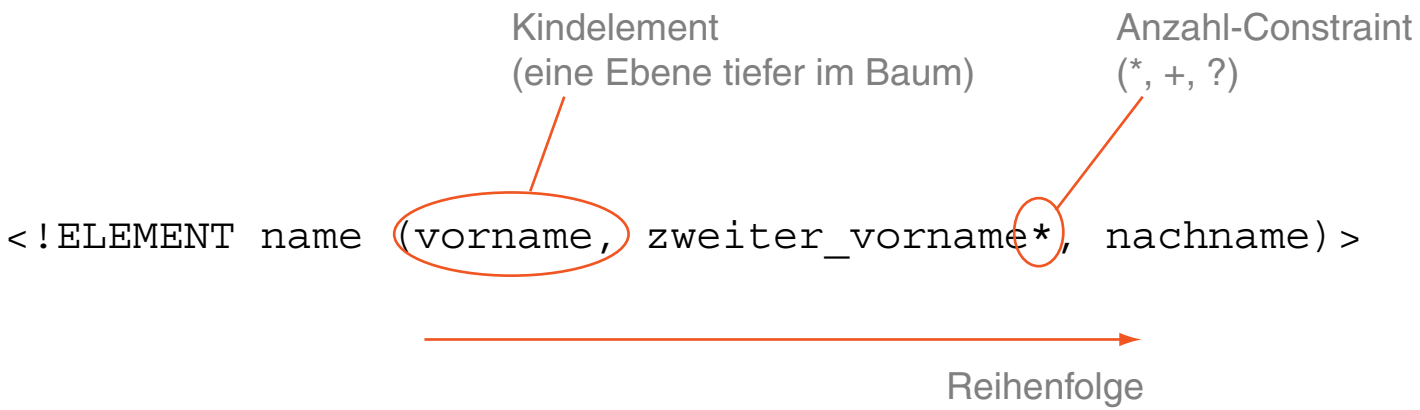
```
<name>
  <nachname>Turing</nachname>
  <vorname>Alan</vorname>
</name>
```

```
<name>
  <nachname>Turing</nachname>
</name>
```

XML-Grundlagen

XML Document Type Definition (Fortsetzung)

Beispiel für Elementdeklaration:



gültige Elementinstanz:

```
<name>
  <vorname>Alan</vorname>
  <nachname>Turing</nachname>
</name>
```

ungültige Elementinstanzen:

```
<name>
  <nachname>Turing</nachname>
  <vorname>Alan</vorname>
</name>
```

```
<name>
  <nachname>Turing</nachname>
</name>
```

XML-Grundlagen

XML Document Type Definition (Fortsetzung)

Deklaration von Alternativen:

```
<!ELEMENT ziffer (null | eins | zwei | drei | vier | fünf)>
```

Klammerung und Kombination der syntaktischen Einheiten:

```
<!ELEMENT kreis (zentrum, (radius | durchmesser))>
```

Wichtige Inhaltsmodelle für Elementtypen:

Inhaltsmodell	**Typischer Aufbau**
einfacher Inhalt	<!ELEMENT *Elementname* (#PCDATA)>
explizite Kindelemente	<!ELEMENT *Elementname* (*Elementname*, . . . , *Elementname*)>
gemischter Inhalt	<!ELEMENT *Elementname* (#PCDATA|*Elementname*)*>
beliebiger Inhalt	<!ELEMENT *Elementname* ANY>
leerer Inhalt	<!ELEMENT *Elementname* EMPTY>

XML-Grundlagen

XML Document Type Definition (Fortsetzung)

Deklaration von Alternativen:

```
<!ELEMENT ziffer (null | eins | zwei | drei | vier | fünf)>
```

Klammerung und Kombination der syntaktischen Einheiten:

```
<!ELEMENT kreis (zentrum, (radius | durchmesser))>
```

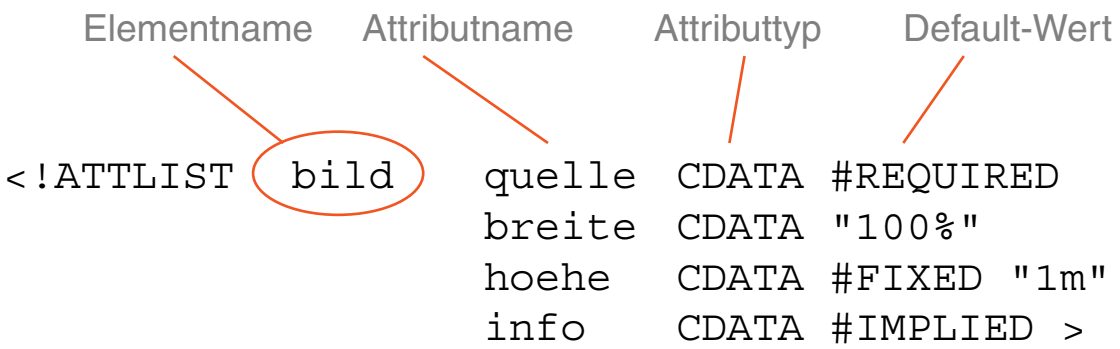
Wichtige Inhaltsmodelle für Elementtypen:

Inhaltsmodell	**Typischer Aufbau**
einfacher Inhalt	`<!ELEMENT` *Elementname* `(#PCDATA)>`
explizite Kindelemente	`<!ELEMENT` *Elementname* `(`*Elementname*`, . . . ,` *Elementname*`)>`
gemischter Inhalt	`<!ELEMENT` *Elementname* `(#PCDATA|`*Elementname*`)*>`
beliebiger Inhalt	`<!ELEMENT` *Elementname* `ANY>`
leerer Inhalt	`<!ELEMENT` *Elementname* `EMPTY>`

XML-Grundlagen

XML Document Type Definition (Fortsetzung)

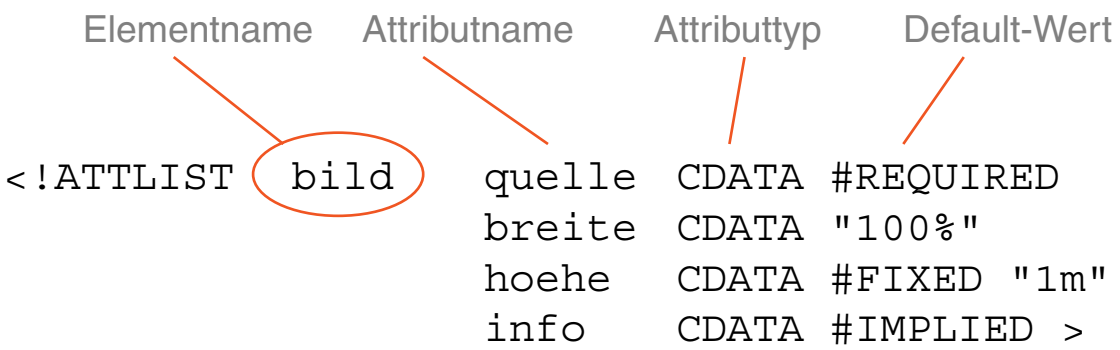
Beispiel für Attributdeklaration:



XML-Grundlagen

XML Document Type Definition (Fortsetzung)

Beispiel für Attributdeklaration:



Wichtige Attributtypen:

`CDATA` (Zeichenkette), `ID` (eindeutiger Name), `IDREF` (Verweis auf ein ID-Attribut), `NMTOKEN` (Symbol), `ENTITY`

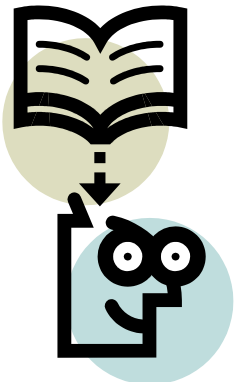
Default-Wert für Attribut	**Semantik**
`#IMPLIED`	das Attribut ist optional
`#REQUIRED`	das Attribut ist obligatorisch
`#FIXED` *Wert*	der Attributwert ist fest und kann angegeben sein
Wert	Default, falls kein Attributwert im Dokument angegeben ist

Bemerkungen:

- Attributwerte dürfen keine Elemente sein oder enthalten.
- In XML gibt es 10 Attributtypen.

XML-Grundlagen

Quiz [www.w3schools.com]



XML-Grundlagen

Internationalisierung

XML-Grundlagen

Internationalisierung (Fortsetzung)

Unterscheidung folgender Konzepte:

1. abstraktes Zeichen *(Abstract character, Character)* [unicode]
   Element zur Konstruktion oder Repräsentation textueller Daten.
2. Zeichendarstellung *(Glyph image, Glyph, Character shape)* [unicode 1, 2]
3. Zeichenvorrat, Zeichensatz *(Character repertoire)* [unicode]
   Menge von abstrakten Zeichen (1) zur Repräsentation von Text.

XML-Grundlagen

Internationalisierung (Fortsetzung)

Unterscheidung folgender Konzepte:

1. abstraktes Zeichen *(Abstract character, Character)* [unicode]
   Element zur Konstruktion oder Repräsentation textueller Daten.
2. Zeichendarstellung *(Glyph image, Glyph, Character shape)* [unicode 1, 2]
3. Zeichenvorrat, Zeichensatz *(Character repertoire)* [unicode]
   Menge von abstrakten Zeichen (1) zur Repräsentation von Text.
4. Code-Raum *(Codespace)* [unicode]
   Menge von Zahlen, die abstrakten Zeichen zugeordnet werden können. Zahlen des Code-Raums heißen Zeichen-Codes *(Code points, Character codes, Character numbers)*.
5. Code-Tabelle, codierter Zeichensatz *(Coded character set, Charset)* [unicode]
   Abbildung eines Zeichenvorrats bzw. Zeichensatzes (3) auf Code-Points (4).
6. Codierungsformat *(Encoding form, Encoding scheme, Encoding)* [unicode 1, 2]
   Format der Byte-Repräsentation eines Code-Points (4).

Bemerkungen:

- Die Namen der meisten Zeichen, die irgendwo auf der Welt benutzt werden, findet man unter www.unicode.org/charts/charindex.html.
- Unterschied zwischen abstrakten Zeichen und Zeichendarstellung:
  “The difference between identifying a code point and rendering it on screen or paper is crucial to understanding. The character identified by a code point is an abstract entity, such as «LATIN CHARACTER CAPITAL A» or «BENGALI DIGIT 5». The mark made on screen or paper – called a glyph – is a visual representation of the character.
  The Unicode Standard does not define glyph images. The standard defines how characters are interpreted, not how glyphs are rendered. The software or hardware-rendering engine of a computer is responsible for the appearance of the characters on the screen. The Unicode Standard does not specify the size, shape, nor style of on-screen characters.” [unicode]

  “In Unicode, the letter A is a platonic ideal. It’s just floating in heaven.”
  [www.joelonsoftware.com]
- Ein Beispiel für einen Zeichenvorrat (3) ist Latein 1 *(Latin 1)*: es ist die Menge der Zeichen, die in ISO/IEC 8859-1 aufgeführt sind und die zum Schreiben der Sprachen Westeuropas benötigt werden.

XML-Grundlagen

Internationalisierung (Fortsetzung)

Ein Zeichenvorrat (3) für eine bestimmte Sprache heißt Alphabet oder Schrift.

Beispiele: lateinische Schrift, chinesische Schrift.

Anspruch von Unicode:

> *"The Unicode Standard provides the capacity to encode all of the characters used for the written languages of the world. To keep character coding simple and efficient, the Unicode Standard assigns each character a unique numeric value and name."* [unicode]

Sprache	(4) Code-Raum	(5) **Code-Tabelle** bzw. **Charset**	(6) **Codierungsformat** bzw. **Encoding**	Code-Einheit	Code-Länge
Englisch	0-7F	US-ASCII	*kanonisch*	1 Byte	1 Byte
Westeuropa	0-FF	ISO 8859-1	*kanonisch*	1 Byte	1 Byte
Chinesisch	0-FFFF	Big 5	*kanonisch*	2 Byte	2 Byte
alle	0-10FFFF	Unicode 3.0	UTF-8 UTF-16	1 Byte 2 Byte	1-3 Byte 2-4 Byte

XML-Grundlagen

Internationalisierung (Fortsetzung)

Ein Zeichenvorrat (3) für eine bestimmte Sprache heißt Alphabet oder Schrift.

Beispiele: lateinische Schrift, chinesische Schrift.

Anspruch von Unicode:

> *"The Unicode Standard provides the capacity to encode all of the characters used for the written languages of the world. To keep character coding simple and efficient, the Unicode Standard assigns each character a unique numeric value and name."* [unicode]

Sprache	(4) Code-Raum	(5) **Code-Tabelle** bzw. **Charset**	(6) **Codierungsformat** bzw. **Encoding**	Code-Einheit	Code-Länge
Englisch	0-7F	US-ASCII	*kanonisch*	1 Byte	1 Byte
Westeuropa	0-FF	ISO 8859-1	*kanonisch*	1 Byte	1 Byte
Chinesisch	0-FFFF	Big 5	*kanonisch*	2 Byte	2 Byte
alle	0-10FFFF	Unicode 3.0	UTF-8 UTF-16	1 Byte 2 Byte	1-3 Byte 2-4 Byte

Bemerkungen:

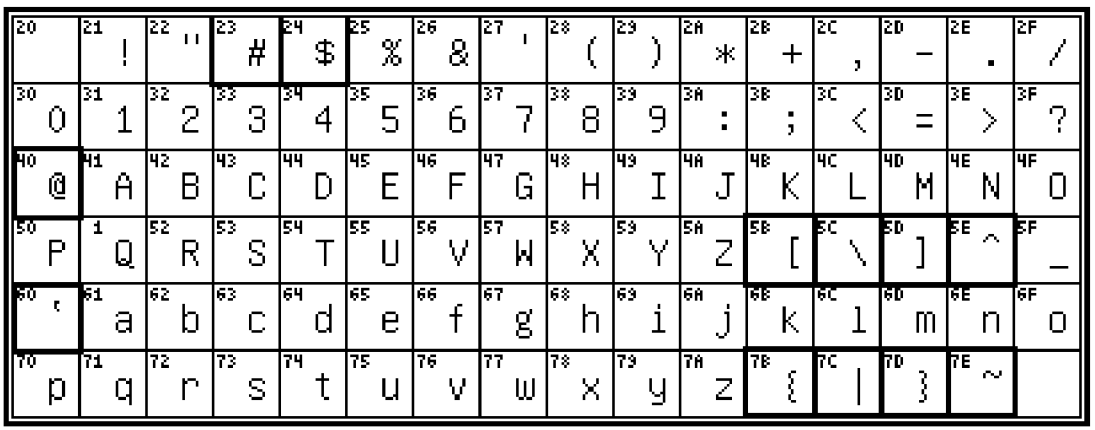
- UTF-8 und UTF-16 verwenden variable Code-Längen.
- Viele Code-Tabellen liegen in nur *einem* Codierungsformat / Encoding (6) vor, das in der Tabelle hier als „kanonisch“ bezeichnet ist. In der Praxis wird oft – aber fälschlicherweise – der Name der Code-Tabelle / Charset (5) für das Codierungsformat / Encoding (6) verwendet.
- Um die Verwirrung komplett zu machen:
  Tatsächlich spezifiziert man in der Meta-Information von HTML-Dokumenten das Attribut charset und in der Meta-Information von XML-Dokumenten das Attribut encoding. Als Werte hierfür können die Namen der Code-Tabelle (`ISO 8859-1`, `US-ASCII`, etc.) oder des Codierungsformats (`UTF-8`, `UTF-16`, etc.) auftreten.
- UTF steht für Unicode Transformation Format. [unicode]
- Damit ein XML-Parser ein Dokument lesen kann, muss er die verwendete Code-Tabelle / Charset (5) und dessen Codierungsformat / Encoding (6) kennen.

XML-Grundlagen

Internationalisierung (Fortsetzung)

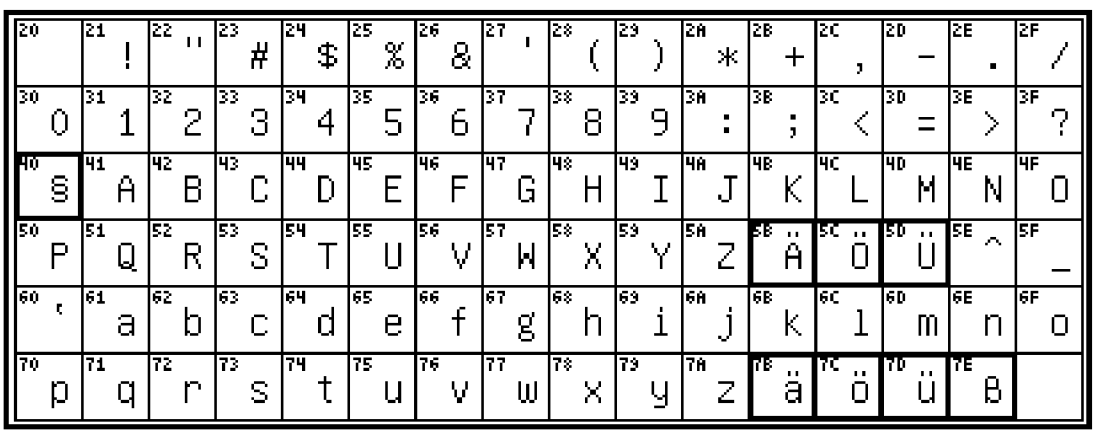
Beispiel ASCII-Code-Tabellen: druckbare Zeichen erhalten die Zeichen-Codes von 32 bis 126, Steuerzeichen von 0 bis 31.

US-ASCII.



20	21 !	22 "	23 #	24 $	25 %	26 &	27 '	28 (	29)	2A *	2B +	2C ,	2D -	2E .	2F /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7	38 8	39 9	3A :	3B ;	3C <	3D =	3E >	3F ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G	48 H	49 I	4A J	4B K	4C L	4D M	4E N	4F O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W	58 X	59 Y	5A Z	5B [	5C \	5D]	5E ^	5F _
60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g	68 h	69 i	6A j	6B k	6C l	6D m	6E n	6F o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w	78 x	79 y	7A z	7B {	7C |	7D }	7E ~	

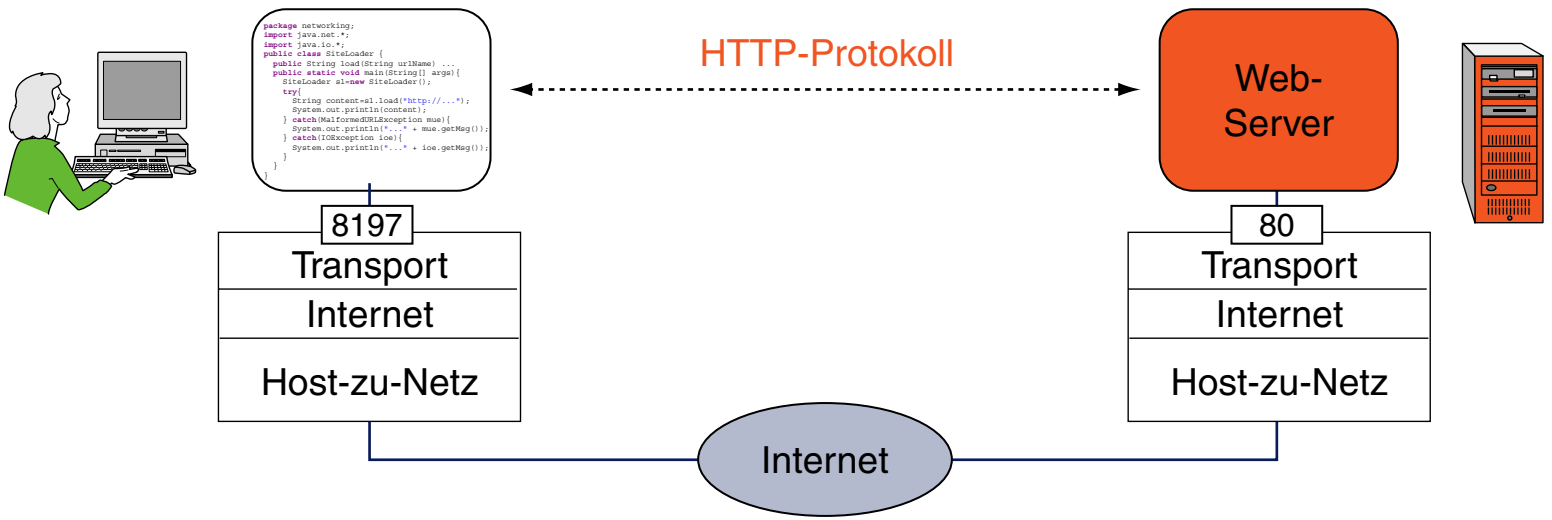
GER-ASCII.



20	21 !	22 "	23 #	24 $	25 %	26 &	27 '	28 (	29)	2A *	2B +	2C ,	2D -	2E .	2F /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7	38 8	39 9	3A :	3B ;	3C <	3D =	3E >	3F ?
40 §	41 A	42 B	43 C	44 D	45 E	46 F	47 G	48 H	49 I	4A J	4B K	4C L	4D M	4E N	4F O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W	58 X	59 Y	5A Z	5B Ä	5C Ö	5D Ü	5E ^	5F _
60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g	68 h	69 i	6A j	6B k	6C l	6D m	6E n	6F o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w	78 x	79 y	7A z	7B ä	7C ö	7D ü	7E ß	

XML-Grundlagen

Internationalisierung: HTTP-Kommunikation mit Java [Response-Message]



XML-Grundlagen

Internationalisierung: HTTP-Kommunikation mit Java (Fortsetzung) [SiteLoader]

```
public String load(String urlName) throws MalformedURLException, IOException{
     URL url=new URL(urlName);
     HttpURLConnection con=(HttpURLConnection)url.openConnection();

     String contentType=con.getContentType();
     System.out.println("Content-Type: "+contentType);
     String charsetEncodingName=parseCharsetName(contentType);
     System.out.println("Charset encoding: "+charsetEncodingName);
     InputStream in=con.getInputStream();
     BufferedReader br;
     if(charsetEncodingName!=null)
          br=new BufferedReader(new InputStreamReader(in, charsetEncodingName));
     else
          br=new BufferedReader(new InputStreamReader(in));

     String curLine;
     StringBuilder content=new StringBuilder();
     while((curLine=br.readLine())!=null){
          content.append(curLine);
          content.append("\n");
     }
     br.close();
     in.close();
     con.disconnect();
     return content.toString();
}
```

XML-Grundlagen

Internationalisierung: HTTP-Kommunikation mit Java (Fortsetzung) [SiteLoader]

```
public String load(String urlName) throws MalformedURLException, IOException{
     URL url=new URL(urlName);
     HttpURLConnection con=(HttpURLConnection)url.openConnection();

     String contentType=con.getContentType();
     System.out.println("Content-Type: "+contentType);
     String charsetEncodingName=parseCharsetName(contentType);
     System.out.println("Charset encoding: "+charsetEncodingName);
     InputStream in=con.getInputStream();
     BufferedReader br;
     if(charsetEncodingName!=null)
          br=new BufferedReader(new InputStreamReader(in, charsetEncodingName));
     else
          br=new BufferedReader(new InputStreamReader(in));

     String curLine;
     StringBuilder content=new StringBuilder();
     while((curLine=br.readLine())!=null){
          content.append(curLine);
          content.append("\n");
     }
     br.close();
     in.close();
     con.disconnect();
     return content.toString();
}
```

XML-Grundlagen

Internationalisierung: HTTP-Kommunikation mit Java (Fortsetzung) [Response-Message]

```
public String parseCharsetName(String contentType){
     // Extracts the charset-value from the Content-Type header.
     // Example. Content-Type: "text/html; charset=UTF-8"
     // If no charset-value is specified, return ISO-8859-1 as default.

     String charset = "ISO-8859-1";
     for (String param : contentType.replace(" ","").split(";")) {
         if (param.toLowerCase().startsWith("charset=")) {
              charset = param.split("=", 2)[1];
              break;
         }
     }
     return charset;
}
```

XML-Grundlagen

Internationalisierung: HTTP-Kommunikation mit Java (Fortsetzung) [SiteLoader]

```
package documentlanguages.webcrawler;
import java.net.*;
import java.io.*;

public class SiteLoader2 {

     public String load(String urlName) ...
     public String parseCharsetName(String contentType) ...

     public static void main(String[] args){
          SiteLoader2 sl=new SiteLoader2();
          try{
               String content=sl.load("http://www.microsoft.de");
               System.out.println(content);
          }
          catch(Exception e){
               System.out.println("Exception: " + e.getMessage());
          }
     }
}
```

XML-Grundlagen

Internationalisierung: HTTP-Kommunikation mit Java (Fortsetzung) [Response-Message]

```
Content-Type: text/html; charset=utf-8
Charset encoding: utf-8


<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >

<html dir="ltr" lang="de">
<head>
<META http-equiv="Content-Type" content="text/html; charset=utf-8" >
<!--TOOLBAR_EXEMPT-->
<meta http-equiv="PICS-Label" content="(PICS-1.1 &quot;http://www.rsac.org/...
<meta name="KEYWORDS" content="Produkte; Headlines; Downloads; News; ...
<meta name="DESCRIPTION" content="Microsoft Deutschland Homepage....
<meta name="MS.LOCALE" content="de-DE" >
<meta name="CATEGORY" content="home page" >
<meta name='WT.sp' content='_germany_'>
<meta name='DCSext.wt_target' content='Generic'>
<title>Microsoft Deutschland GmbH</title>
<base href="http://g.msn.com/mh_mshp_de-DE/98765" >
<style type="text/css" media="all">
@import "http://i.microsoft.com/h/en-us/r/hp.css";
</style>

<body>
...
```

XML-Grundlagen

Namensräume

Namensräume *(Namespaces)* in XML haben folgende Aufgaben:

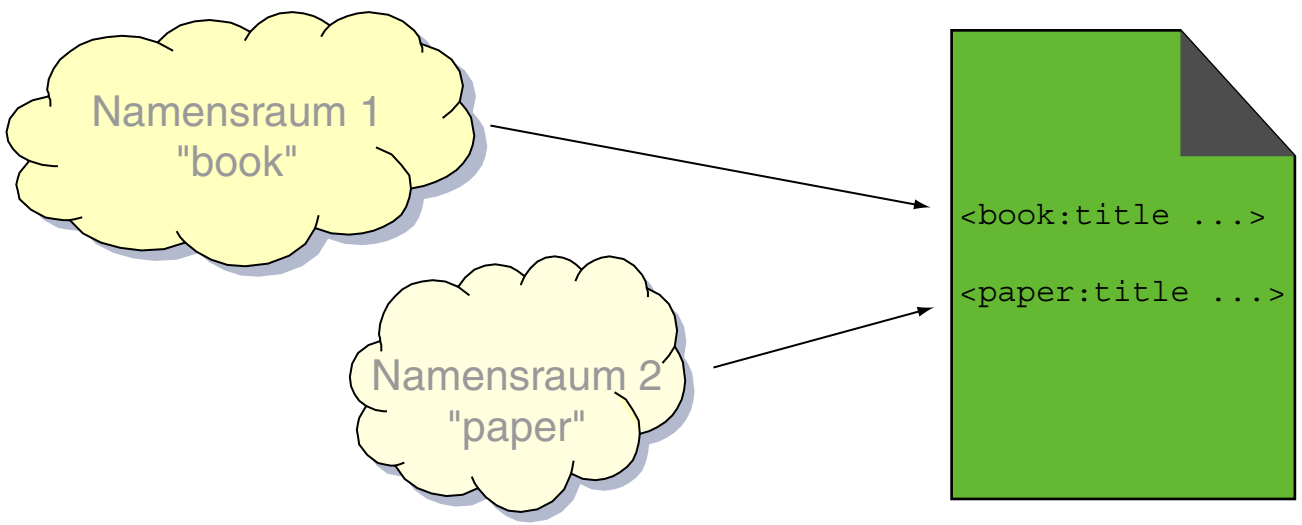
- Unterscheidung von Elementen und Attributen, die den gleichen Namen haben, aber in verschiedenen Zusammenhängen verwendet werden.
  Stichwort: Vermeidung von Namenskonflikten
- Logische Gruppierung aller Elemente und Attribute einer Anwendung.

XML-Grundlagen

Namensräume

Namensräume *(Namespaces)* in XML haben folgende Aufgaben:

- Unterscheidung von Elementen und Attributen, die den gleichen Namen haben, aber in verschiedenen Zusammenhängen verwendet werden.
  Stichwort: Vermeidung von Namenskonflikten
- Logische Gruppierung aller Elemente und Attribute einer Anwendung.



Namensräume sind Bezeichner – sie definieren keine Umgebung *(Scope)*.

XML-Grundlagen

Namensräume (Fortsetzung)

Namensräume *(Namespaces)* in XML haben folgende Aufgaben:

- Unterscheidung von Elementen und Attributen, die den gleichen Namen haben, aber in verschiedenen Zusammenhängen verwendet werden.
  Stichwort: Vermeidung von Namenskonflikten
- Logische Gruppierung aller Elemente und Attribute einer Anwendung.

Verwendung von Namensräumen in zwei Schritten:

1. Deklaration einer Präfix-Bindung an eine URI mittels `xmlns`.

   ```
   xmlns:book="http://www.books.com"
   ```

2. Qualifizierung

book:`title`	qualifizierender Name
book:	Präfix
`title`	lokaler Teil

Jede URI ist als Namensraum verwendbar.

XML-Grundlagen

Namensräume (Fortsetzung)

Gültigkeit der Präfix-Bindung:

- innerhalb des Elements (einschließlich), in dem die Bindung deklariert ist.

```
<book:Buch xmlns:book="http://www.books.com">
  <book:Titel>Heuristics</book:Titel>
  <Autor>Judea Pearl<Autor>
</book:Buch>
```

- Überschreiben der Deklaration innerhalb der Elementhierarchie möglich.

XML-Grundlagen

Namensräume (Fortsetzung)

Gültigkeit der Präfix-Bindung:

- innerhalb des Elements (einschließlich), in dem die Bindung deklariert ist.

```
<book:Buch xmlns:book="http://www.books.com">
  <book:Titel>Heuristics</book:Titel>
  <Autor>Judea Pearl<Autor>
</book:Buch>
```

- Überschreiben der Deklaration innerhalb der Elementhierarchie möglich.

Default-Namensraum:

- Bindung des leeren Präfix an eine URI.

```
<Buch xmlns="http://www.books.com">
  <Titel>Heuristics</Titel>
  <Autor>Judea Pearl</Autor>
</Buch>
```

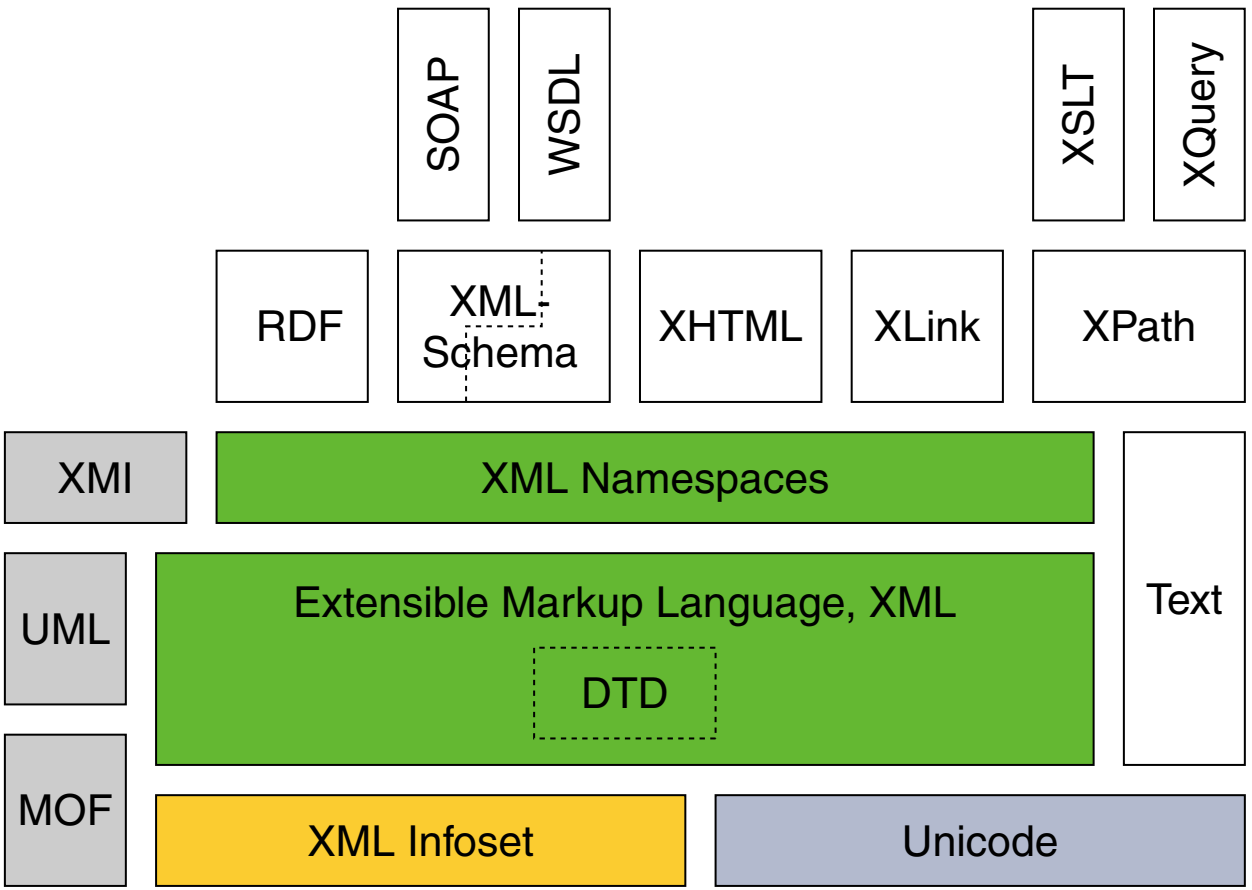
Bemerkungen:

- Man bezeichnet Elementnamen, Attributnamen etc. die zu einem Namensraum gehören, als „qualifiziert“. Diese Qualifizierung kann explizit über ein Präfix, oder implizit über die Deklaration eines Default-Namensraums geschehen.
- Nicht qualifizierte Namen gehören zu dem anonymen bzw. universellen Namensraum. Im ersten Beispiel befindet sich `<Autor>` im anonymen Namensraum.
- Die Konzepte Default-Namensraum (= implizite Qualifizierung ohne Präfix) und anonymer Namensraum (= keine Qualifizierung) sind sorgfältig zu unterscheiden.
- *Attribute* ohne Präfix liegen nicht im Default-Namensraum. D.h., auch wenn sich ein Element in einem bestimmten (Default-)Namensraum befindet, so befinden sich seine *Attribute* ohne Präfix im anonymen Namensraum.
- Eine explizit durch Präfixzuordnung vorgenommene Namensraumdeklaration besitzt Präferenz gegenüber dem Default-Namensraum.
- Durch Überschreibung des Default-Namensraums mit der Zeichenkette leeren Inhalts – formal: der Zuweisung der leeren URI – kann eine bestehende Namensraumdefinition aufgehoben werden. Als Resultat entsteht eine Situation identisch zu einem Dokument ohne festgelegte Namensräume; d.h., die Elemente befinden sich im anonymen Namensraum.
- Eine Elementinstanz kann mehrere Namensraumdeklarationen aufnehmen. In der Praxis hat es sich aus Übersichtlichkeitsgründen durchgesetzt, alle in einem XML-Dokument verwendeten Namensräume zu Beginn des Dokuments im Wurzelelement zu deklarieren.

XML-Grundlagen

XML Information Set [W3C]

Einordnung [Jeckle 2004] :



XML-Grundlagen

XML Information Set (Fortsetzung)

Das XML Information Set (Infoset) definiert das XML-Dokumenten unterliegende Datenmodell. Es dient zur Beantwortung der Frage:

> Welche Informationen sind in einem XML-Dokument codiert?

Ein Parser, der ein XML-Dokument analysiert, orientiert sich an der Definition des XML Information Set und stellt die entsprechenden aus dem XML-Dokument ableitbaren Informationen in einer Datenstruktur bereit.

XML-Grundlagen

XML Information Set (Fortsetzung)

Das XML Information Set (Infoset) definiert das XML-Dokumenten unterliegende Datenmodell. Es dient zur Beantwortung der Frage:

> Welche Informationen sind in einem XML-Dokument codiert?

Ein Parser, der ein XML-Dokument analysiert, orientiert sich an der Definition des XML Information Set und stellt die entsprechenden aus dem XML-Dokument ableitbaren Informationen in einer Datenstruktur bereit. Beispiele:

- wie ein Element heißt
- zu welchem Namensraum ein Element gehört
- Reihenfolge der Elementinstanzen
- Code-Tabelle

Beispiele für nicht ableitbare Information:

- Größe des Leerraums zwischen Attributen
- Reihenfolge der Attribute eines Elementtyps

Bemerkungen:

- Das XML Information Set ist keine Sprache wie andere W3C-Spezifikationen, sondern ein Datenmodell. Die XML-Syntax ist eine Serialisierung dieses Datenmodells.
- Das W3C hat mittlerweile drei Datenmodelle für XML-Dokumente definiert: XML Information Set, XPath, Document Object Model (DOM). Das XPath-Datenmodell basiert auf einer Baumstruktur, die bei der Abfrage eines XML-Dokuments durchlaufen wird und ist dem XML Information Set ähnlich; DOM ist der Vorläufer beider Datenmodelle. DOM und das XPath-Datenmodell können als Interpretationen des XML Information Sets betrachtet werden. [MSDN]

XML-Grundlagen

XML Information Set (Fortsetzung)

Das XML Information Set eines XML-Dokuments wird als Baum repräsentiert.

Die Elemente des Baums heißen Informationseinheiten *(Information Items)* und sind von einem der folgenden Typen:

1. Document Information Item
2. Element Information Item
3. Attribute Information Item
4. Processing Instruction Information Item
5. Unexpanded Entity Reference Information Item
6. Character Information Item
7. Comment Information Item
8. Document Type Declaration Information Item
9. Unparsed Entity Information Item
10. Notation Information Item
11. Namespace Information Item

XML-Grundlagen

XML Information Set: Beispiel

```
<?xml version="1.0"
 encoding="ISO-8859-1"
 standalone="yes">

<person>
  <name geburtstag="23-06-1912">
    <vorname>Alan</vorname>
    <nachname>Turing</nachname>
  </name>
</person>
```

XML-Grundlagen

XML Information Set: Beispiel (Fortsetzung)

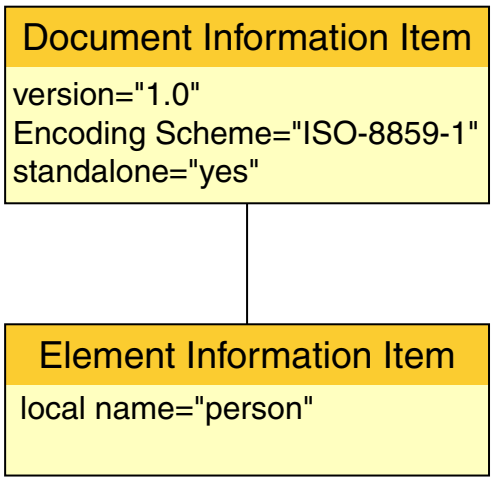
Document Information Item

version="1.0"
Encoding Scheme="ISO-8859-1"
standalone="yes"

```
<?xml version="1.0"
 encoding="ISO-8859-1"
 standalone="yes">

<person>
  <name geburtstag="23-06-1912">
    <vorname>Alan</vorname>
    <nachname>Turing</nachname>
  </name>
</person>
```

XML-Grundlagen

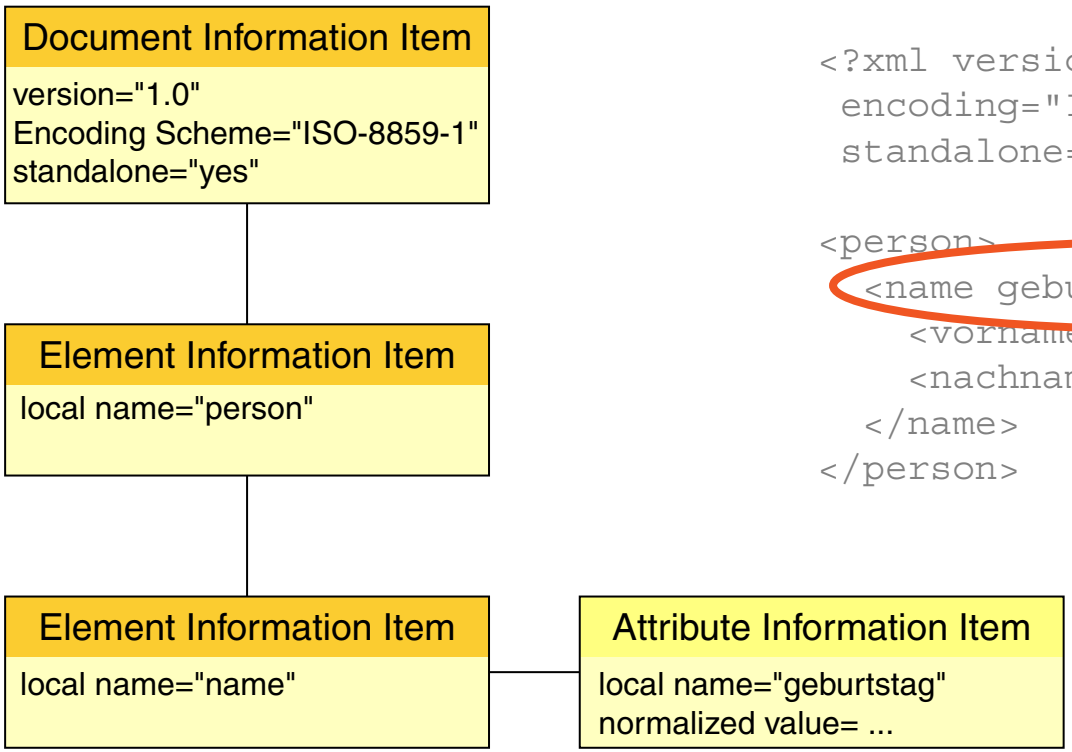
XML Information Set: Beispiel (Fortsetzung)




```
<?xml version="1.0"
 encoding="ISO-8859-1"
 standalone="yes">

<person>
  <name geburtstag="23-06-1912">
    <vorname>Alan</vorname>
    <nachname>Turing</nachname>
  </name>
</person>
```

XML-Grundlagen

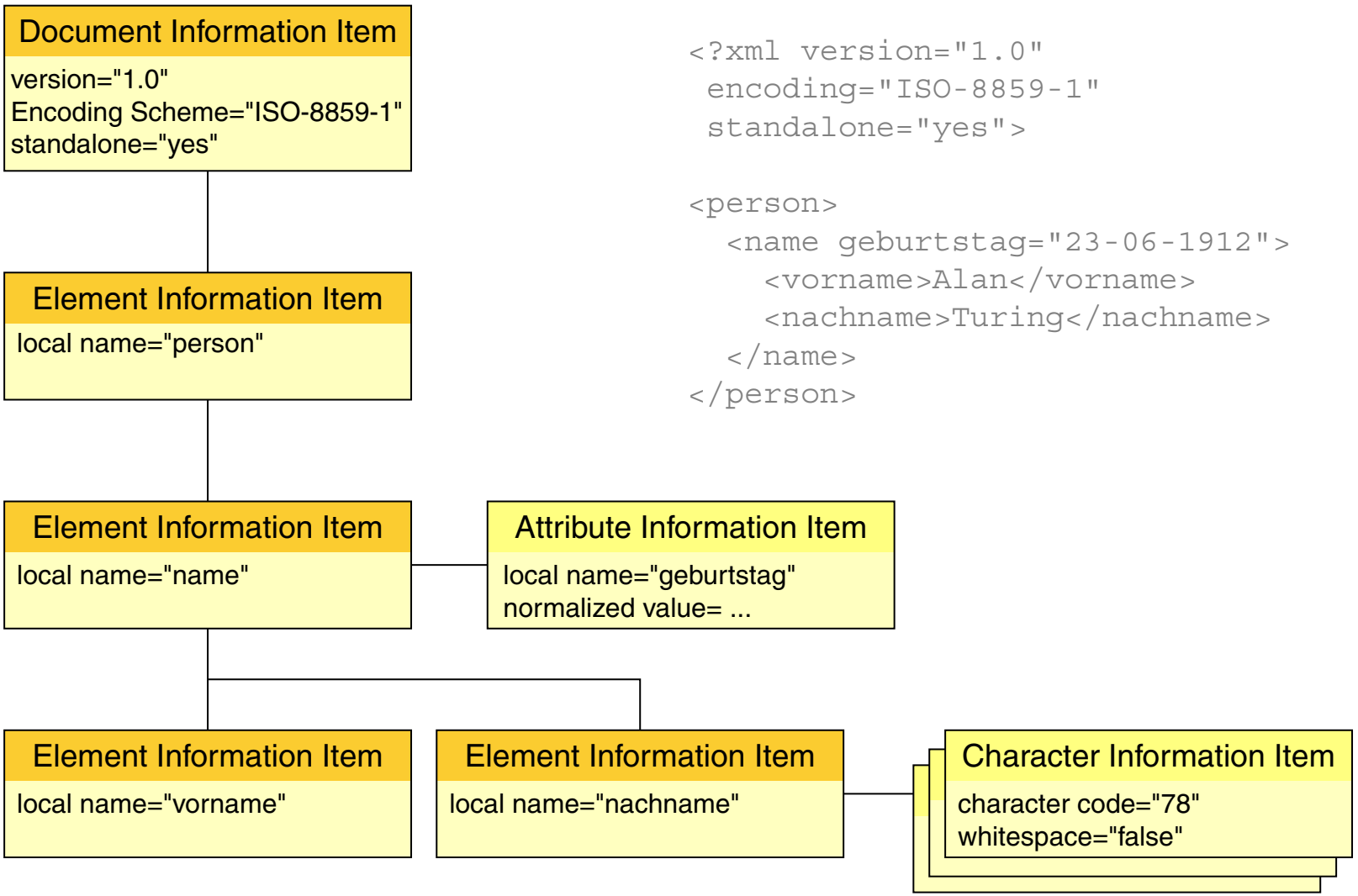
XML Information Set: Beispiel (Fortsetzung)




```
<?xml version="1.0"
 encoding="ISO-8859-1"
 standalone="yes">

<person>
  <name geburtstag="23-06-1912">
    <vorname>Alan</vorname>
    <nachname>Turing</nachname>
  </name>
</person>
```

XML-Grundlagen

XML Information Set: Beispiel (Fortsetzung)




```
<?xml version="1.0"
 encoding="ISO-8859-1"
 standalone="yes">

<person>
  <name geburtstag="23-06-1912">
    <vorname>Alan</vorname>
    <nachname>Turing</nachname>
  </name>
</person>
```

XML-Grundlagen

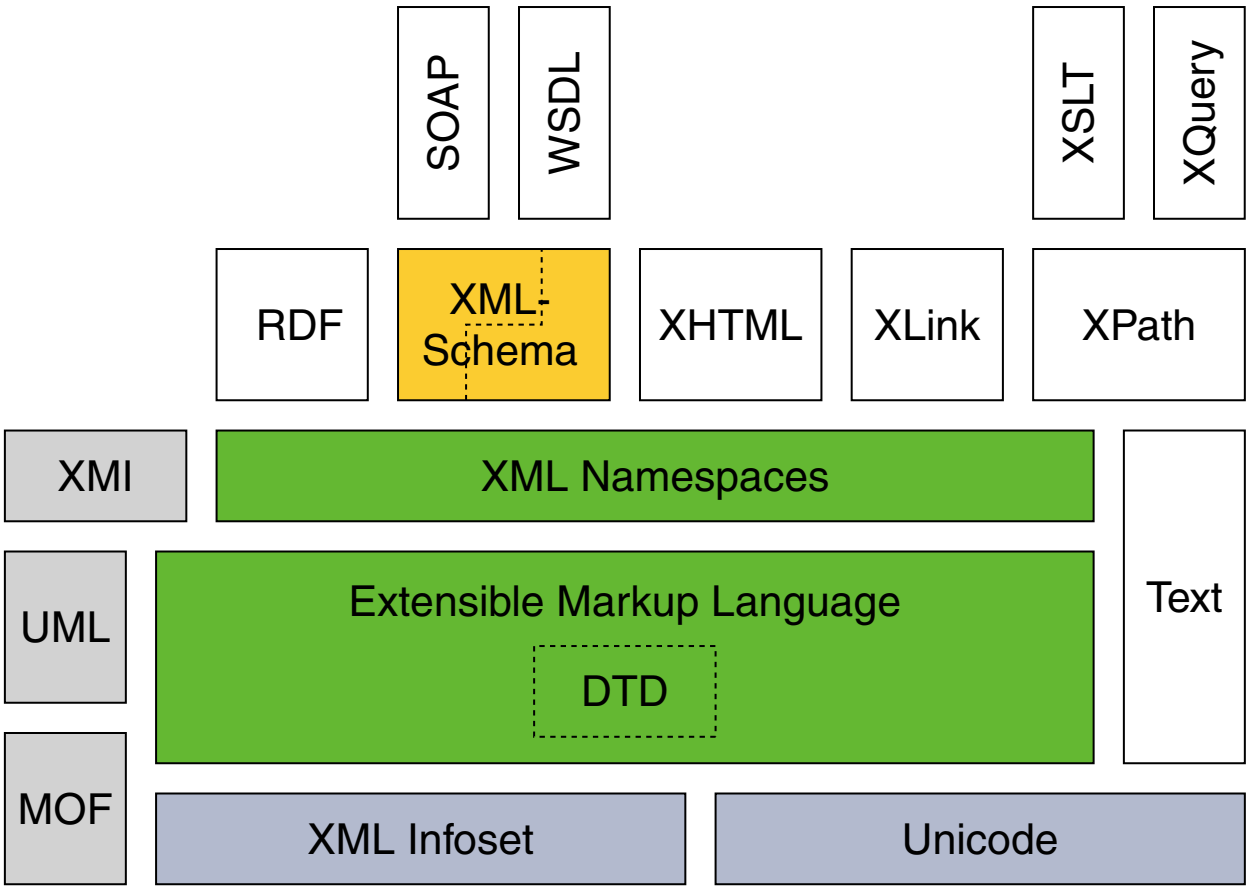
Quellen zum Nachlernen und Nachschlagen im Web

- T. Bray et al. *XML 1.1.*
  www.w3.org/TR/xml11
- T. Bray et al. *Namespaces in XML 1.1.*
  www.w3.org/TR/xml-names11
- J. Cowan, R. Tobin. *XML Information Set, Second Edition.*
  www.w3.org/TR/xml-infoset (deutsche Übersetzung)
- W3 Schools. *XML Tutorial.*
  www.w3schools.com/xml
- M. Jeckle. *Web-Seiten von Mario Jeckle mit XML Vorlesungen und Links.*
  www.jeckle.de
- Unicode.*Glossary.*
  www.unicode.org/glossary
- Joel on Software. *Unicode Essay.*
  www.joelonsoftware.com/articles/Unicode.htm

Kapitel WT:III (Fortsetzung)

XML-Schema

Einordnung [Jeckle 2004]



XML-Schema

Historie

Es gab/gibt eine Reihe von Erweiterungen und Vorschlägen hinsichtlich neuer Schemasprachen. Die größte praktische Bedeutung hat der W3C-Standard XML Schema Definition Language, XSD – kurz: XML-Schema.

- XML-Schema bildet zusammen mit XML 1.0 und den Namensräumen die Basis aller weiteren W3C-XML-Sprachstandards.
- Mittelfristig werden die Grammatiken neu entwickelter XML-Sprachen nicht mehr in der DTD-Syntax formuliert.

1999 XML-Schema 1.1 Requirements, Note. [W3C]

2004 XML-Schema Second Edition: Primer, Structures, Datatypes. Recommendation. [W3C 0, 1, 2]

2012 XML Schema Definition Language, XSD: Structures, Datatypes. Recommendation. [W3C 1, 2]

Bemerkungen:

- XSD ist eine Weiterentwicklung von XML: unpraktikable oder fehlerträchtige Sprachbestandteile von SGML werden nicht mehr durch den Grammatikmechanismus unterstützt.
- Der XSD-Sprachvorschlag gliedert sich in zwei Teile:
  XSD-Part 1 „Structures“ www.w3.org/TR/xmlschema11-1. Beschreibung von Inhaltsmodellen für Elemente, Attributstrukturen und wiederverwendbare Strukturen. Bildet die Konzepte von DTDs nach.
  XSD-Part 2 „Datatypes“ www.w3.org/TR/xmlschema11-2. Beschreibung von Datentypen und Constraints zur Konsistenzerhaltung. Es handelt sich um ein eigenständiges Typsystem, das auch in anderen W3C-Arbeitsgruppen Verwendung findet.

  Der XML-Schema Part 0 „Primer“ www.w3.org/TR/xmlschema-0 gibt eine gute Einführung in die Konzepte und Ziele zu XML-Schema.

XML-Schema

DTD versus XML-Schema

```
<!ELEMENT person (vorname, nachname, beruf+)>
<!ATTLIST person geburtsjahr NMTOKEN #IMPLIED>
<!ELEMENT vorname (#PCDATA)>
<!ELEMENT nachname (#PCDATA)>
<!ELEMENT beruf (#PCDATA)>
```

XML-Schema

DTD versus XML-Schema

```
<!ELEMENT person (vorname, nachname, beruf+)>
<!ATTLIST person geburtsjahr NMTOKEN #IMPLIED>
<!ELEMENT vorname (#PCDATA)>
<!ELEMENT nachname (#PCDATA)>
<!ELEMENT beruf (#PCDATA)>
```

```
<?xml version="1.0"?>
<xsd:schema ...>
  <xsd:element name="person">
    <xsd:complexType>
      <xsd:all>
        <xsd:element name="vorname" type="xsd:string"/>
        <xsd:element name="nachname" type="xsd:string"/>
        <xsd:element name="beruf" type="xsd:string"
           minOccurs="0" maxOccurs="3"/>
      </xsd:all>
      <xsd:attribute name="geburtsjahr" type="xsd:gYear" use="optional"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

XML-Schema

DTD versus XML-Schema

```
<!ELEMENT person (vorname, nachname, beruf+)>
<!ATTLIST person geburtsjahr NMTOKEN #IMPLIED>
<!ELEMENT vorname (#PCDATA)>
<!ELEMENT nachname (#PCDATA)>
<!ELEMENT beruf (#PCDATA)>
```

```
<?xml version="1.0"?>
<xsd:schema ...>
  <xsd:element name="person">
    <xsd:complexType>
      <xsd:all>
        <xsd:element name="vorname" type="xsd:string"/>
        <xsd:element name="nachname" type="xsd:string"/>
        <xsd:element name="beruf" type="xsd:string"
           minOccurs="0" maxOccurs="3"/>
      </xsd:all>
      <xsd:attribute name="geburtsjahr" type="xsd:gYear" use="optional"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

XML-Schema

DTD versus XML-Schema

```
<!ELEMENT person (vorname, nachname, beruf+)>
<!ATTLIST person geburtsjahr NMTOKEN #IMPLIED>
<!ELEMENT vorname (#PCDATA)>
<!ELEMENT nachname (#PCDATA)>
<!ELEMENT beruf (#PCDATA)>
```

```
<?xml version="1.0"?>
<xsd:schema ...>
  <xsd:element name="person">
    <xsd:complexType>
      <xsd:all>
        <xsd:element name="vorname" type="xsd:string"/>
        <xsd:element name="nachname" type="xsd:string"/>
        <xsd:element name="beruf" type="xsd:string"
           minOccurs="0" maxOccurs="3"/>
      </xsd:all>
      <xsd:attribute name="geburtsjahr" type="xsd:gYear" use="optional"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

XML-Schema

DTD versus XML-Schema

```
<!ELEMENT person (vorname, nachname, beruf+)>
<!ATTLIST person geburtsjahr NMTOKEN #IMPLIED>
<!ELEMENT vorname (#PCDATA)>
<!ELEMENT nachname (#PCDATA)>
<!ELEMENT beruf (#PCDATA)>
```

```
<?xml version="1.0"?>
<xsd:schema ...>
  <xsd:element name="person">
    <xsd:complexType>
      <xsd:all>
        <xsd:element name="vorname" type="xsd:string"/>
        <xsd:element name="nachname" type="xsd:string"/>
        <xsd:element name="beruf" type="xsd:string"
           minOccurs="0" maxOccurs="3"/>
      </xsd:all>
      <xsd:attribute name="geburtsjahr" type="xsd:gYear" use="optional"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Bemerkungen:

- Ein XML-Schema beinhaltet [W3C] :
  1. *Definitionen* von einfachen und komplexen *Datentypen*.
  2. *Deklarationen* von *Elementtypen*, die in Instanzdokumenten erlaubt sind; ein Elementtyp ist von einem bestimmten Datentyp.

- Beachte, dass Instanzen des `<xsd:element>`-Elements sowohl mit Inhalt zwischen öffnendem und schließendem Tag:

  ```
  <xsd:element name="person">
    <xsd:complexType> ... </xsd:complexType>
  </xsd:element>
  ```

  als auch leer Verwendung finden:

  ```
  <xsd:element name="vorname" type="xsd:string"/>
  ```

  Die Syntax im ersten Fall folgt dem Entwurfsmuster zur Deklaration von Elementtypen einschließlich der Definition eines neuen (hier: anonymen, komplexen) Datentyps. Die Syntax im zweiten Fall folgt dem Entwurfsmuster zur Deklaration von Elementtypen unter Rückgriff auf einen existierenden Datentyp über das `type`-Attribut.

- Gewöhnungsbedürftig bei XML-Schema ist, dass die (kontextfreie) Grammatik zur Beschreibung einer Dokumentenstruktur nicht mehr in der Form von Regeln, sondern „objektorientiert“, mittels XSD-Elementinstanzen und deren Schachtelung geschieht.

- Gewöhnungsbedürftig bei XML-Schema ist auch, dass die Syntax der Metasprache (hier: Grammatik im XML-Schemadokument) gleich der Syntax der Objektsprache (hier: Elemente im XML-Instanzdokument) ist.

XML-Schema

DTD versus XML-Schema (Fortsetzung)

Grenzen von DTDs:

- DTDs sind schwer zu lesen bzw. zu verstehen. Das betrifft vor allem die Konzepte zur Strukturierung.
- DTDs sind nachträglich nicht erweiterbar. Die Definition einer DTD muss vollständig sein und sämtliche Regeln für ihre Anwendung definieren.
- DTDs unterstützen nur wenige Datentypen, das Typsystem ist nicht erweiterbar.
- DTDs erschweren die Wiederverwendbarkeit: Elementstrukturen können nur innerhalb der definierenden DTD wiederverwendet werden, Attribute sind immer an das umgebende Element gebunden.
- DTDs unterstützen keine Vererbung.
- Die DTD-Syntax ist nicht XML-konform.
- DTDs unterstützen keine Namensräume.

XML-Schema

DTD versus XML-Schema (Fortsetzung)

Durch die Datentypen in XML-Schema sind Aufgaben einfacher geworden:

- Definition von Constraints für zugelassenen Inhalt
- Überprüfung der Korrektheit von Daten
- Verarbeitung von Daten aus Datenbanken
- Spezialisierung und Anpassung von Datentypen
- Definition komplexer Datentypen
- Konvertierung zwischen Daten verschiedenen Typs

Weitere Errungenschaften [w3schools] :

- XML-Schemata sind modular verwendbar
- XML-Schemata sind in XML-Syntax geschrieben
- XML-Schemata verwenden das XML-Namensraumkonzept

XML-Schema

DTD versus XML-Schema (Fortsetzung)

Unterscheidung von Dokumenten hinsichtlich ihres Aufbaus:

1. Erzählende *(narrative)* Dokumente.
   Dokumente, die aus Abschnitten und Unterabschnitten bestehen; die gesamte Struktur ist weitgehend linear: Bücher, Artikel, etc.
2. Datensatzartige Dokumente.
   Stark typisierte Dokumente; zielen auf Datenaustausch ab.

DTDs sind gut für erzählende Dokumente geeignet,

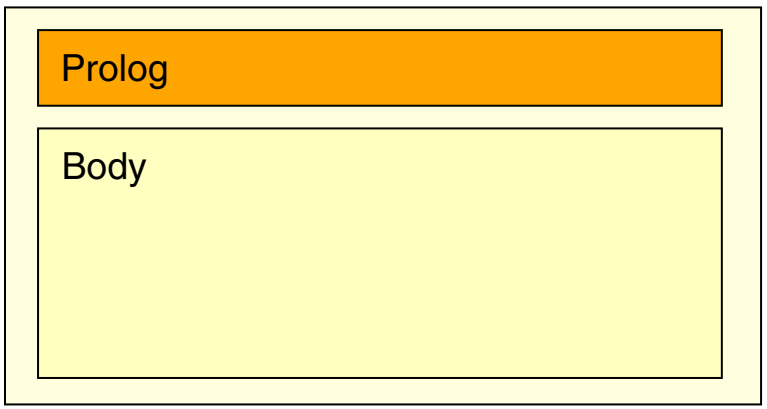
XML-Schema eignet sich gut für datensatzartige Dokumente.

XML-Schema

Aufbau eines XML-Schemas

XML-Schemata sind XML-Dokumente:

XML-Schema




```
<?xml version="1.0"?>


<xsd:schema xmlns:xsd=...>
  <xsd:element name=...>
    ...
  </xsd:element>
  ...
</xsd:schema>
```

- Wurzelelement jedes XML-Schemas ist das Element `<xsd:schema>`.
- Direkte (= nicht tiefer verschachtelt liegende) Kindelemente von `<xsd:schema>` sind *globale* Elemente.
- Die Attribute von `<xsd:schema>` definieren Eigenschaften, die für alle Elemente und Attribute des Schemas gelten.
- Vergleiche hierzu die Grundlagen zum XML-Dokument.

Bemerkungen:

- Globale Elemente können als Wurzelelement eines Instanzdokuments verwendet werden.
- Die Dateiendung einer XML-Schemadatei ist `.xsd`.

XML-Schema

Aufbau eines XML-Schemas (Fortsetzung) [Beispiel]

```
<?xml version="1.0"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.buw.de/webtec"
    elementFormDefault="qualified">

  <xsd:element name="person">
    <xsd:complexType>
      ...
  </xsd:element>

</xsd:schema>
```

XML-Schema

Aufbau eines XML-Schemas (Fortsetzung) [Beispiel]

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.buw.de/webtec"
    elementFormDefault="qualified">
  <xsd:element name="person">
    <xsd:complexType>
      ...
  </xsd:element>
</xsd:schema>
```

- Jedes XML-Schema ist eine eigenständige Datei; die Einbettung in ein Instanzdokument, also die Bildung von Internal Subsets, ist nicht möglich.
- Zwei Arten von Namensräumen, die deklariert werden können:
  1. Einen oder mehrere Namensräume, aus denen die *im Schema verwendeten* Elemente und Datentypen stammen.
  2. Einen Zielnamensraum *(Target Namespace)*, dem die Elemente eines schemagültigen Instanzdokumentes angehören müssen.

Bemerkungen:

- Die Syntax und Semantik des XSD-Vokabulars ist durch durch die normativen Referenzen www.w3.org/TR/xmlschema-0, www.w3.org/TR/xmlschema11-1 und www.w3.org/TR/xmlschema11-2 des W3C standardisiert.
- Das XSD-Vokabular umfasst Namen für Elemente und Attribute, die zur Erstellung von XML-Schemadokumenten zur Verfügung stehen. Der zugehörige Namensraum heißt www.w3.org/2001/XMLSchema. Übliches Präfix ist `xsd:` oder `xs:`, kann aber beliebig gewählt werden.
- Alle im Schema *global* notierten Elemente und Attribute sind dem Zielnamensraum zugeordnet. Bei ihrer Instanziierung in einem schemagültigen Instanzdokument müssen sie aus dem entsprechenden Namensraum stammen, also entsprechend qualifiziert sein. Auch bei Referenzen innerhalb des Schemas muss diese Namensraumzugehörigkeit berücksichtigt werden.
- Das `targetNamespace`-Attribut, also die Deklaration eines Zielnamensraums, ist optional. Legt das XML-Schema *keinen* Zielnamensraum fest, so befinden sich alle im Schema neu eingeführten Namen im Default-Namensraum.

Bemerkungen (Fortsetzung):

- Durch Angabe der Attribute `elementFormDefault` und `attributeFormDefault` lässt sich die Zuordnung der *lokal definierten* Elemente und Attribute zum Zielnamensraum steuern. Standdardmäßig sind diese Attribute auf `unqualified` gesetzt und ein deklarierter Zielnamensraum bezieht sich nur auf die globalen Elemente und Attribute. Wird der Wert der beiden Attribute auf `qualified` gesetzt, so sind auch die lokalen Elemente und Attribute als `qualified` deklariert und müssen im Instanzdokument entsprechend qualifiziert verwendet werden.
- Um die Wartbarkeit und Lesbarkeit großer Schemata zu verbessern, kann ein Schema auf mehrere Schemadateien (mit dem gleichen Zielnamensraum) aufgeteilt werden. Die einzelnen Dateien werden dann mittels des Elements `include` aus dem XSD-Vokabular in einem Master-Dokument zusammengefasst.
- Mit dem Element `import` aus dem XSD-Vokabular lassen sich – unter Angabe eines neuen Zielnamensraums – Elemente aus „fremden“ Schemata importieren.

XML-Schema

Verknüpfung von XML-Schema und Instanzdokument [Beispiel]

```
<?xml version="1.0"?>

<person
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.buw.de/webtec person.xsd"
    xmlns="http://www.buw.de/webtec">

  <vorname>Alan</vorname>
  <nachname>Turing</nachname>
  <beruf>Mathematiker</beruf>
  <beruf>Informatiker</beruf>
</person>
```

XML-Schema

Verknüpfung von XML-Schema und Instanzdokument [Beispiel]

```
<?xml version="1.0"?>

<person
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.buw.de/webtec person.xsd"
    xmlns="http://www.buw.de/webtec">

  <vorname>Alan</vorname>
  <nachname>Turing</nachname>
  <beruf>Mathematiker</beruf>
  <beruf>Informatiker</beruf>
</person>
```

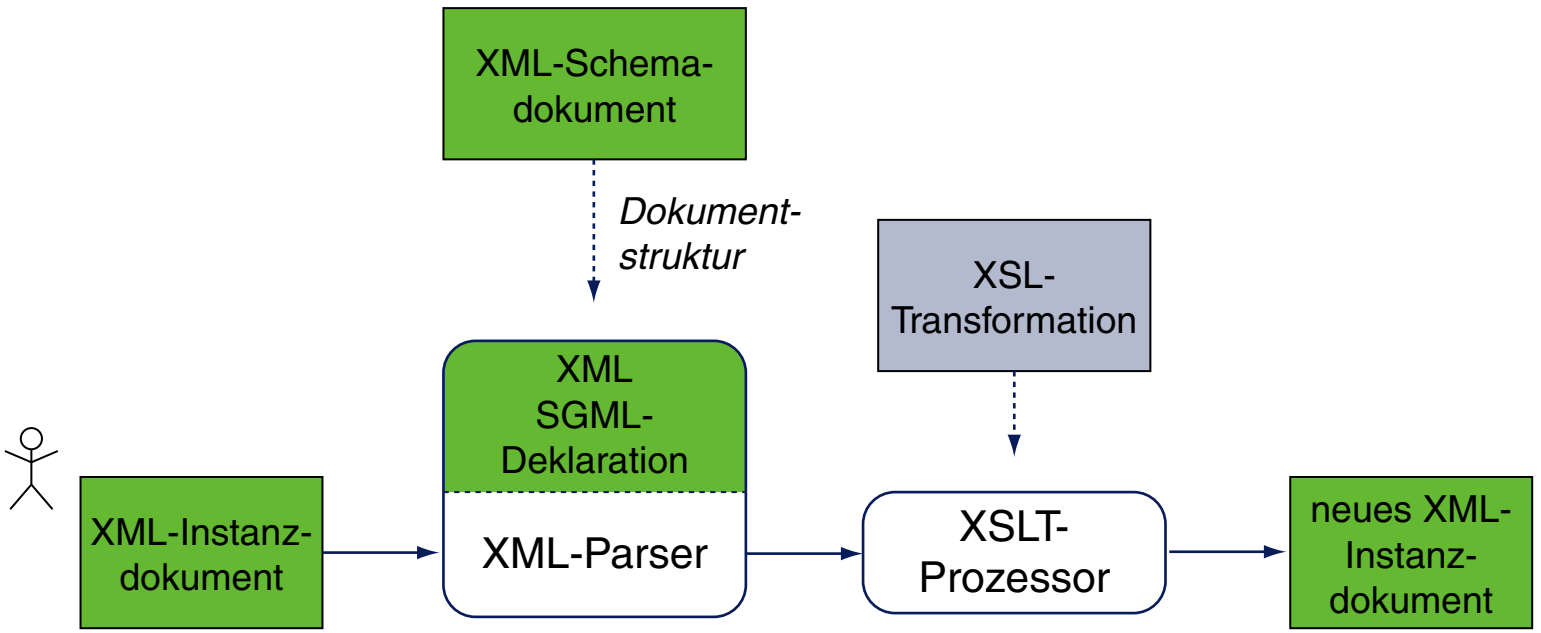
- Die Attribute `schemaLocation` und `noNamespaceSchemaLocation` dienen zur Referenz auf das Schema. Genau eines muss angegeben sein – abhängig davon, ob das Schema einen Zielnamensraum festlegt oder nicht.
- Der erste Parameter von `schemaLocation` ist eine URI für den Zielnamensraum; der zweite Parameter ist eine relative oder absolute URL, die angibt, wo das Schema liegt.
- `noNamespaceSchemaLocation` hat nur den URL-Parameter für das Schema.

Bemerkungen:

- Elemente und Attribute aus dem standardisierten XSD-Vokabular, die in Instanzdokumenten verwendet werden, gehören zum Namensraum www.w3.org/2001/XMLSchema-instance. Übliches Präfix ist `xsi:`, kann aber beliebig gewählt werden.
- Im Beispiel muss das Präfix `xsi:` mit dem Namensraum für Instanzdokumente eingeführt werden, um das Attribut `schemaLocation` zu qualifizieren – das Attribut also vollständig zu benennen und so dem Parser bekannt zu machen.
- Der erste Parameter des Attributes `schemaLocation` im XML-Instanzdokument muss mit dem Wert des Attributes `targetNamespace` im XML-Schema übereinstimmen. Legt das XML-Schema *keinen* Zielnamensraum fest, so wird eine Referenz auf dieses Schema mittels `noNamespaceSchemaLocation` spezifiziert.
- Im Beispiel wird der Zielnamensraum `http://www.buw.de/webtec` an den leeren Präfix gebunden (`xmlns="http://www.buw.de/webtec"`) und dadurch zum Default-Namensraum im XML-Instanzdokument.

XML-Schema

Verknüpfung von XML-Schema und Instanzdokument (Fortsetzung)



Vergleiche hierzu

- die SGML Dokumentenverarbeitung,
- die HTML Dokumentenverarbeitung,
- und die XML Dokumentenverarbeitung mit DTDs.

XML-Schema

Verknüpfung von XML-Schema und Instanzdokument (Fortsetzung)

Definition 1 (gültig hinsichtlich Schema, Instanzdokument)
Ein XML-Dokument heißt gültig hinsichtlich eines Schemas *(schema valid)*, wenn es über ein Schema verfügt, und konform zu diesem aufgebaut ist.

Das zu validierende Dokument wird als Instanzdokument bezeichnet.

Bemerkungen:

- Der angegebene Gültigkeitsbegriff lässt die Konformität hinsichtlich einer eventuell existierenden DTD außer Acht. So sind Dokumente denkbar, die zwar hinsichtlich einer gegebenen DTD als valid eingestuft werden, jedoch ein zugehöriges Schema verletzen – und umgekehrt.
- Aufgrund der Realisierung der Schemasprache als XML-Sprache ist jedes Schema auch ein XML-Dokument. Dadurch wird es möglich, alle erlaubten Schemata selbst durch ein Schema – ein sogenanntes *Metaschema* – zu beschreiben und zu validieren. [W3C] Somit können Schemata mit denselben Werkzeugen analysiert, verarbeitet und geprüft werden, die auch für Instanzdokumente Verwendung finden.
- Für ein Metaschema könnte wiederum ein Schema angegeben werden. Um eine unendliche Reihung zur Validierung notwendiger Schemata zu vermeiden, hat man bei der XML-Standardisierung darauf geachtet, dass die Sprache zur Beschreibung von XML-Schemata ausdrucksstark genug ist, um als Metasprache zur Beschreibung aller erlaubten Schemata zu fungieren.
- W3C-Service zur Schemavalidierung: www.w3.org/2001/03/webdata/xsv
- Altova-Werkzeuge für XML: http://www.altova.com/download-trial.html
- Java Xerces Library: `[steinwebis]$ java dom.Writer -v -s` *Instanzdokument*

XML-Schema

Wie man ein XML-Schema entwickelt

Voraussetzungen, um ein neues XML-Schema zu entwickeln:

- Verständnis für den Zusammenhang zwischen
  - Inhaltsmodell: charakterisiert die Struktur von Elementinstanzen
  - Elementtyp: implementiert ein Inhaltsmodell, ist von einem Datentyp
  - Datentyp eines Elementtyps: deklariert erlaubte Daten
- Sicherer Umgang mit (Ziel-, Default-) Namensräumen

XML-Schema

Wie man ein XML-Schema entwickelt

Voraussetzungen, um ein neues XML-Schema zu entwickeln:

- Verständnis für den Zusammenhang zwischen
  - Inhaltsmodell: charakterisiert die Struktur von Elementinstanzen
  - Elementtyp: implementiert ein Inhaltsmodell, ist von einem Datentyp
  - Datentyp eines Elementtyps: deklariert erlaubte Daten
- Sicherer Umgang mit (Ziel-, Default-) Namensräumen

Aufgaben bei der Entwicklung eines XML-Schemas:

1. *Definition* neuer Datentypen sowie die *Deklaration* der benötigten Elementtypen.
2. Anwendung eines Entwurfsmusters: anonyme versus benannte Datentypen
3. Optimierung durch Anwendung leistungsfähiger Konzepte (Vererbung, Ableitung, etc.) bei der Definition neuer Datentypen.

XML-Schema

Wie man ein XML-Schema entwickelt (Fortsetzung) [Beispiel]

Zwei Entwurfsmuster zur Deklaration von Elementtypen:

1. `<xsd:element name="`*Elementname*`">` *Typedefinition* `</xsd:element>`
   Deklaration des Elementtyps *Elementname* einschließlich der Definition eines Datentyps.

2. `<xsd:element name="`*Elementname*`" type="`*Typename*`"/>`
   Deklaration des Elementtyps *Elementname* unter Verwendung eines gegebenen Datentyps.

XML-Schema

Wie man ein XML-Schema entwickelt (Fortsetzung) [Beispiel]

Zwei Entwurfsmuster zur Deklaration von Elementtypen:

1. `<xsd:element name="`*Elementname*`">` *Typedefinition* `</xsd:element>`
   Deklaration des Elementtyps *Elementname* einschließlich der Definition eines Datentyps.

2. `<xsd:element name="`*Elementname*`" type="`*Typename*`"/>`
   Deklaration des Elementtyps *Elementname* unter Verwendung eines gegebenen Datentyps.

Datentypdefinitionen:

- `<xsd:complexType>` *Typedefinition* `</xsd:complexType>`
  Definition eines anonymen, komplexen Datentyps für Instanzen von *Elementname*.
  (innerhalb eines `<xsd:element name="`*Elementname*`">`-Elementes)

XML-Schema

Wie man ein XML-Schema entwickelt (Fortsetzung) [Beispiel]

Zwei Entwurfsmuster zur Deklaration von Elementtypen:

1. `<xsd:element name="`*Elementname*`">` *Typedefinition* `</xsd:element>`
   Deklaration des Elementtyps *Elementname* einschließlich der Definition eines Datentyps.

2. `<xsd:element name="`*Elementname*`" type="`*Typename*`"/>`
   Deklaration des Elementtyps *Elementname* unter Verwendung eines gegebenen Datentyps.

Datentypdefinitionen:

- `<xsd:complexType>` *Typedefinition* `</xsd:complexType>`
  Definition eines anonymen, komplexen Datentyps für Instanzen von *Elementname*.
  (innerhalb eines `<xsd:element name="`*Elementname*`">`-Elementes)

- `<xsd:complexType name="`*Typename*`">` *Typedefinition* `</xsd:complexType>`
  Definition eines benannten, komplexen Datentyps mit dem Namen *Typename*.
  (global, auf oberster Ebene unter `<xsd:schema>`)

- `<xsd:simpleType name="`*Typename*`">` *Typedefinition* `</xsd:simpleType>`
  Definition eines benannten, einfachen Datentyps mit dem Namen *Typename*.
  (global, auf oberster Ebene unter `<xsd:schema>`)

Bemerkungen:

- W3C-Definition von Inhaltsmodell:
  “A content model is a simple grammar governing the allowed types of the child elements and the order in which they are allowed to appear.” [W3C 1, 2]
- Entwurfsmuster 1 zur Deklaration von Elementtypen kommt zur Anwendung, wenn kein passender Datentyp vorhanden ist und dieser *(inline, ad-hoc)* definiert werden soll.
- Entwurfsmuster 2 zur Deklaration von Elementtypen kommt zur Anwendung, wenn ein passender Datentyp vordefiniert *(built-in)* ist oder an anderer Stelle definiert wurde.
- Zwischen der Definition von Datentypen und der Deklaration von Elementtypen ist sorgfältig zu unterscheiden. Das Erstgenannte deklariert einen Wertebereich für einen neu geschaffenen (= definierten) Datentyp; das Zweitgenannte deklariert die erlaubten Elementinstanzen in Instanzdokumenten. [W3C]
- Die Definition benannter Datentypen ist sinnvoll, wenn diese mehrfach (mittels des `type`-Attributs) Verwendung finden sollen.

XML-Schema

XML-Schema Teil 2: Datentypen [W3C]

Einteilung der Datentypen:

- Primitive Datentypen lassen sich nicht auf Basis anderer Datentypen ableiten. Gegensatz: abgeleitete Datentypen
- Vordefinierte Datentypen *(Built-in Datatypes)* [W3C] sind Teil der Schema Definition Language XSD. Gegensatz: anwenderspezifische Datentypen

XML-Schema

XML-Schema Teil 2: Datentypen [W3C]

Einteilung der Datentypen:

- Primitive Datentypen lassen sich nicht auf Basis anderer Datentypen ableiten. Gegensatz: abgeleitete Datentypen
- Vordefinierte Datentypen *(Built-in Datatypes)* [W3C] sind Teil der Schema Definition Language XSD. Gegensatz: anwenderspezifische Datentypen

Schemaelemente zur Definition anwenderspezifischer Datentypen:

1. `<xsd:complexType>`
   Definiert einen neuen komplexen Datentyp für Elemente. Kann weitere Elemente (= Kindelemente) deklarieren und Attribute haben.
2. `<xsd:simpleType>`
   Definiert einen neuen einfachen Datentyp für Elemente und Attribute, der ausschließlich aus Text besteht und weder Attribute noch Kindelemente deklarieren darf.

XML-Schema

XML-Schema Teil 2: Datentypen (Fortsetzung)

Wichtige Schemaelemente, um Constraints für Kindelemente in komplexen Datentypen zu spezifizieren:

Element	Semantik
`<xsd:all>`	jedes Kindelement kann höchstens ein Mal auftreten
`<xsd:choice>`	erlaubt das Auftreten eines der Kindelemente gemäß Attributen
`<xsd:sequence>`	Kindelemente müssen in angegebener Reihenfolge auftreten

Weitere Konzepte zur Definition neuer Datentypen:

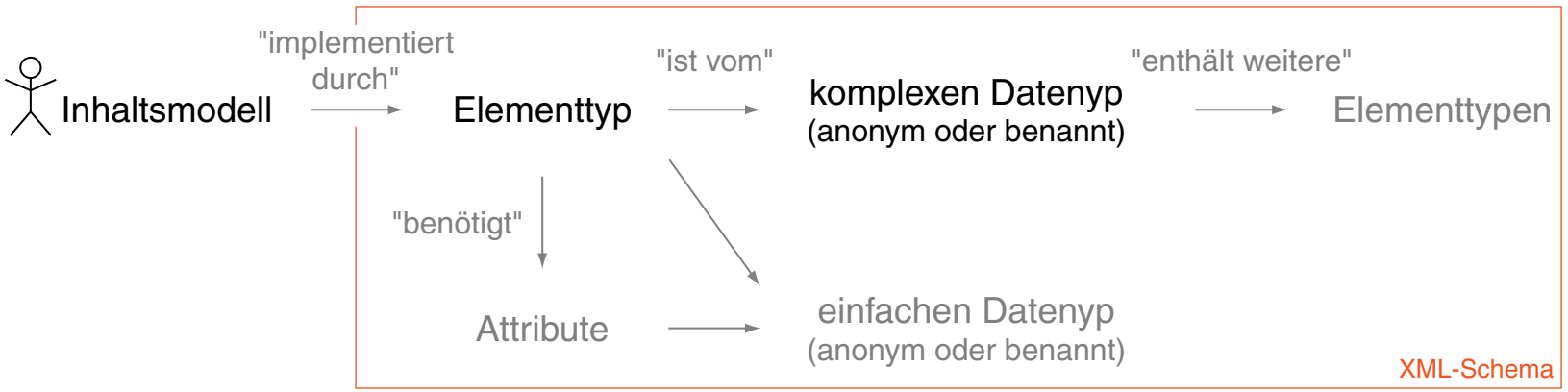
- Vererbung
- Ableiten durch Einschränkung
- Ableiten durch Erweiterung
- Formulierung von Wertebereichs-Constraints z.B. durch reguläre Ausdrücke

Bemerkungen:

- Ein Beispiel für einen primitiven Datentyp ist `xsd:float`. Dagegen ist `xsd:integer` kein primitiver Datentyp, weil er durch Spezialisierung von `xsd:decimal` abgeleitet ist.
- Alle vier Datentypkombinationen sind möglich: primitive wie auch abgeleitete Datentypen können vordefiniert *(built-in)* oder anwenderspezifisch sein.
- Die Definition anwenderspezifischer Datentypen kann *anonym* oder *benannt* geschehen. Eine Benennung erfolgt durch das `name`-Attribut im `<xsd:complexType>` bzw. `<xsd:simpleType>`-Element.
  Anonyme Datentypen beziehen sich nur auf die Deklaration der Elementtypen, in der sie eingeführt wurden. Benannte Datentypen sind für alle tieferliegenden Ebenen sichtbar und lassen sich mittels des `type`-Attributes zur Datentypdeklaration verwenden.
- Bei der Erzeugung anwenderspezifischer Datentypen können noch die Schemaelemente `<xsd:simpleContent>` und `<xsd:complexContent>` zum Einsatz kommen.
  Innerhalb einer `<xsd:complexType>`-Definition – und nur dort kommen sie zum Einsatz – dienen sie zur Deklaration des Inhalts im Zusammenhang mit `restriction` (falls der Inhalt auf bestimmte Datentypen eingeschränkt sein soll) oder mit `extension` (falls der Inhalt hinsichtlich bestimmter Datentypen erweitert werden soll).

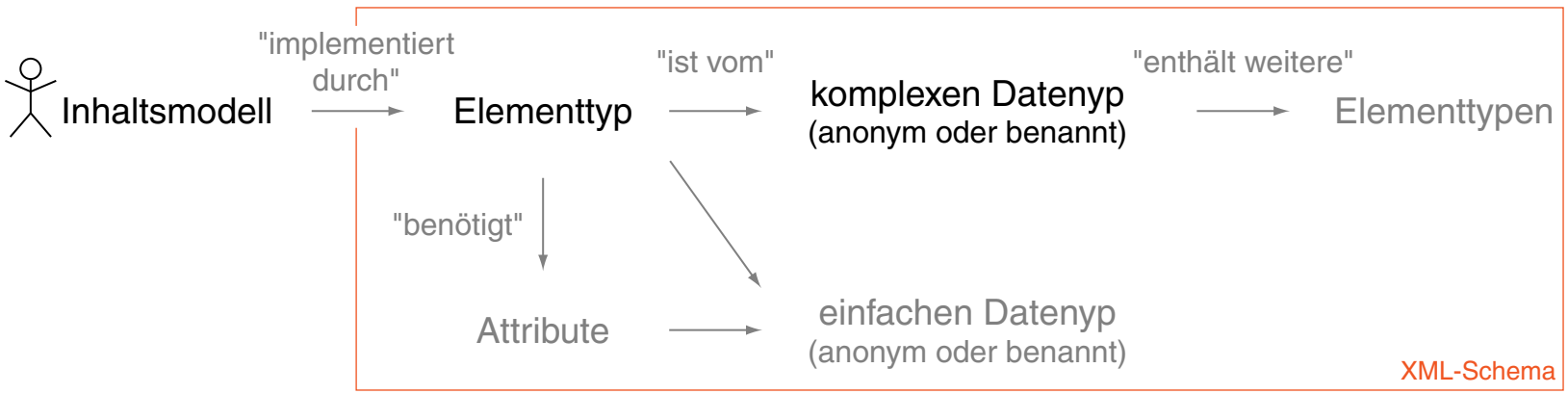
XML-Schema

XML-Schema Teil 1: Inhaltsmodelle



XML-Schema

XML-Schema Teil 1: Inhaltsmodelle



Wichtige Inhaltsmodelle [DTD] :

(a) einfacher Inhalt: unstrukturierter, typisierter Text

(b) explizite Kindelemente: feste Schachtelungsstruktur vorgeschrieben

(c) gemischter Inhalt: Kombination von Elementen mit unstrukturiertem Text

(d) beliebiger Inhalt: keine Constraints für Schachtelungsstrukturen

(e) leerer Inhalt

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(a) einfacher Inhalt [W3C] :

```
<xsd:element
  name="Elementname"
  type="Typename"/>
```

Vergleichbare DTD-Definition, ohne Typ-Deklaration:

```
<!ELEMENT Elementname (#PCDATA)>
```

Beispiel einer Elementtypdeklaration:

```
<xsd:element
  name="vorname"
  type="xsd:string"
  minOccurs="1"
  maxOccurs="3"/>
```

Beispiel einer Elementinstanz:

```
<vorname>Tim</vorname>
```

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(a) einfacher Inhalt [W3C] :

```
<xsd:element
  name="Elementname"
  type="Typename"/>
```

Vergleichbare DTD-Definition, ohne Typ-Deklaration:

```
<!ELEMENT Elementname (#PCDATA)>
```

Beispiel einer Elementtypdeklaration:

```
<xsd:element
  name="vorname"
  type="xsd:string"
  minOccurs="1"
  maxOccurs="3"/>
```

Beispiel einer Elementinstanz:

```
<vorname>Tim</vorname>
```

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(a) einfacher Inhalt [W3C] :

```
<xsd:element
  name="Elementname"
  type="Typename"/>
```

Vergleichbare DTD-Definition, ohne Typ-Deklaration:

```
<!ELEMENT Elementname (#PCDATA)>
```

Beispiel einer Elementtypdeklaration:

```
<xsd:element
  name="vorname"
  type="xsd:string"
  minOccurs="1"
  maxOccurs="3"/>
```

Beispiel einer Elementinstanz:

```
<vorname>Tim</vorname>
```

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

Wichtige Attribute in der Deklaration eines Elementtyps:

Attribut	Semantik
`default`	Zeichenkette mit Default-Wert, der konform zum gewählten Datentyp ist
`minOccurs`	Mindestanzahl zulässiger Instanzen dieses Elementtyps, Default ist 1
`maxOccurs`	Höchstanzahl zulässiger Instanzen dieses Elementtyps, Default ist 1
`name`	unqualifizierter Name des Elementtyps
`ref`	Verweis auf eine globale Deklaration des Elementtyps
`type`	Deklaration des Datentyps für den Elementtyp

Bemerkungen:

- Für einfache Inhaltsmodelle mit unstrukturiertem Text ist in DTDs nur der Datentyp `#PCDATA` vorgesehen. XML-Schema Part 2 definiert 44 primitive Datentypen.
- Mit der Spezifikation `maxOccurs="unbounded"` wird eine beliebige Anzahl an Instanzen zugelassen.

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(b) explizite Kindelemente [W3C] :

```
<xsd:element name="person">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="vorname" type="xsd:string" maxOccurs="3"/>
      <xsd:element name="nachname" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Vergleichbare DTD-Definition, ohne Anzahl-Constraint:

```
<!ELEMENT person (vorname+, nachname)>
```

Beispiel einer Elementinstanz:

```
<person>
  <vorname>Tim</vorname>
  <nachname>Berners-Lee</nachname>
</person>
```

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(b) explizite Kindelemente [W3C] :

```
<xsd:element name="person">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="vorname" type="xsd:string" maxOccurs="3"/>
      <xsd:element name="nachname" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Vergleichbare DTD-Definition, ohne Anzahl-Constraint:

```
<!ELEMENT person (vorname+, nachname)>
```

Beispiel einer Elementinstanz:

```
<person>
  <vorname>Tim</vorname>
  <nachname>Berners-Lee</nachname>
</person>
```

Bemerkungen:

- Das Beispiel folgt Entwurfsmuster 1: Deklaration eines Elementtyps einschließlich der Definition eines neuen (anonymen, komplexen) Datentyps.
- Das Inhaltsmodell mit explizit angegebenen Kindelementen stellt das für die Modellierungspraxis wichtigste Inhaltsmodell dar.
- Alternativ zu `<xsd:sequence>` zur Angabe der Reihenfolge von Kindelementen kann die Angabe `<xsd:choice>` für eine exklusive Auswahl oder `<xsd:all>` für eine Liste von Kindelementen in beliebiger Reihenfolge (jedes höchstens einmal) erfolgen.
- Die angegebene Deklaration des Elementtyps ist eine Abkürzung der folgenden Deklaration, bei der ein komplexes Inhaltsmodell (`<xsd:complexContent>`) mittels `<xsd:restriction>` eines allgemeineren Typs `<xsd:anyType>` notiert ist:

```
<xsd:element name="person">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
        <xsd:sequence>
          <xsd:element name="vorname" type="xsd:string" maxOccurs="3"/>
          <xsd:element name="nachname" type="xsd:string"/>
        </xsd:sequence>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
```

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(c) gemischter Inhalt [W3C] :

```
<xsd:element name="Brief">
  <xsd:complexType mixed="true">
    <xsd:sequence>
      <xsd:element name="Anrede" type="xsd:string"/>
      <xsd:element name="PS" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Vergleichbare DTD-Definition:

```
<!ELEMENT Brief (#PCDATA | Anrede | PS)*>
```

Beispiel einer Elementinstanz:

```
<Brief>
<Anrede>Sehr geehrter Herr Berners-Lee</Anrede> Die W3C-Empfehlung für
XSD bringt nicht nur Verbesserungen gegenüber DTDs. <PS>Viele Grüße</PS>
</Brief>
```

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(c) gemischter Inhalt [W3C] :

```
<xsd:element name="Brief">
  <xsd:complexType mixed="true">
    <xsd:sequence>
      <xsd:element name="Anrede" type="xsd:string"/>
      <xsd:element name="PS" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Vergleichbare DTD-Definition:

```
<!ELEMENT Brief (#PCDATA | Anrede | PS)*>
```

Beispiel einer Elementinstanz:

```
<Brief>
<Anrede>Sehr geehrter Herr Berners-Lee</Anrede> Die W3C-Empfehlung für
XSD bringt nicht nur Verbesserungen gegenüber DTDs. <PS>Viele Grüße</PS>
</Brief>
```

Bemerkungen:

- Das Beispiel folgt Entwurfsmuster 1: Deklaration eines Elementtyps einschließlich der Definition eines neuen (anonymen, komplexen) Datentyps.
- Ein gemischtes Inhaltsmodell *(Mixed Content Model)* liegt vor, wenn die Instanz eines Elementtyps sowohl einfachen Inhalt (unstrukturierten Text) als auch Kindelemente aufnehmen kann. Vergleiche die W3C-Definition von Mixed Content:
  "An element type has mixed content when elements of that type may contain character data, optionally interspersed with child elements. In this case, the types of the child elements MAY be constrained, but not their order or their number of occurrences." [W3C]
- Während gemischte Inhalte aus Auszeichnungssymbolen und freiem Text durch DTD-validierende Parser nur rudimentär geprüft werden können, ermöglicht XSD eine vollständige Validierung gemischter Inhalte. Die Strukturvalidierung der DTD erlaubt zwar die Spezifikation von innerhalb unstrukturierter Textpassagen auftretenden Elementen, nicht jedoch die Überwachung deren Auftrittshäufigkeit oder Reihenfolge. [Jeckle 2004]

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(d) beliebiger Inhalt [W3C] :

```
<xsd:element
  name="Elementname"
  type="xsd:anyType">
</xsd:element>
```

bzw.

```
<xsd:element
  name="Elementname">
</xsd:element>
```

oder

```
<xsd:element name="Elementname"/>
```

Vergleichbare DTD-Definition:

```
<!ELEMENT Elementname ANY>
```

Bemerkungen:

- Wird das `type`-Attribut nicht spezifiziert oder – alternativ – explizit der Wert `xsd:anyType` zugewiesen, so können Instanzen dieses Elementtyps beliebige, XML-wohlgeformte Inhalte aufnehmen.

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(e) leerer Inhalt [W3C] :

```
<xsd:element
  name="Elementname">
  <xsd:complexType/>
</xsd:element>
```

Vergleichbare DTD-Definition:

```
<!ELEMENT Elementname EMPTY>
```

Beispiel einer Elementtypdeklaration:

```
<xsd:element name="internationalPrice">
  <xsd:complexType>
    <xsd:attribute name="currency" type="xsd:string"/>
    <xsd:attribute name="value" type="xsd:decimal"/>
  </xsd:complexType>
</xsd:element>
```

Beispiel einer Elementinstanz:

```
<internationalPrice currency="EUR" value="423.46"/>
```

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(e) leerer Inhalt [W3C] :

```
<xsd:element
  name="Elementname">
  <xsd:complexType/>
</xsd:element>
```

Vergleichbare DTD-Definition:

```
<!ELEMENT Elementname EMPTY>
```

Beispiel einer Elementtypdeklaration:

```
<xsd:element name="internationalPrice">
  <xsd:complexType>
    <xsd:attribute name="currency" type="xsd:string"/>
    <xsd:attribute name="value" type="xsd:decimal"/>
  </xsd:complexType>
</xsd:element>
```

Beispiel einer Elementinstanz:

```
<internationalPrice currency="EUR" value="423.46"/>
```

XML-Schema

XML-Schema Teil 1: Inhaltsmodelle (Fortsetzung)

(e) leerer Inhalt [W3C] :

```
<xsd:element
  name="Elementname">
  <xsd:complexType/>
</xsd:element>
```

Vergleichbare DTD-Definition:

```
<!ELEMENT Elementname EMPTY>
```

Beispiel einer Elementtypdeklaration:

```
<xsd:element name="internationalPrice">
  <xsd:complexType>
    <xsd:attribute name="currency" type="xsd:string"/>
    <xsd:attribute name="value" type="xsd:decimal"/>
  </xsd:complexType>
</xsd:element>
```

Beispiel einer Elementinstanz:

```
<internationalPrice currency="EUR" value="423.46"/>
```

Bemerkungen:

- Leere Inhaltsmodelle vermitteln ihre Information auschließlich über Attribute oder über ihre Position innerhalb anderer Elemente.
- Ein XML-Schema-validierender Parser verhält sich hierbei identisch zu einem DTD-validierenden Parser für das Inhaltsmodell `EMPTY`. D.h., es werden nur die beiden folgenden Darstellungen zur Angabe eines leeren Elements akzeptiert: `<elementName/>` `<elementName></elementName>`
- Die angegebene Deklaration des Elementtyps ist eine Abkürzung der folgenden Deklaration, bei der ein komplexes Inhaltsmodell (`<xsd:complexContent>`) mittels `<xsd:restriction>` eines allgemeineren Typs `<xsd:anyType>` notiert ist:

```
<xsd:element name="internationalPrice">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
        <xsd:attribute name="currency" type="xsd:string"/>
        <xsd:attribute name="value" type="xsd:decimal"/>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
```

XML-Schema

XML-Schema Teil 1: Attribute

Beispiel:

```
<xsd:attribute name="myDecimal" type="xsd:decimal"/>
```

Innerhalb des `<xsd:attribute>`-Elements lassen sich (durch Attribute) spezifische Eigenschaften festlegen, unter anderem:

Attribut	Semantik
`default`	Zeichenkette mit Default-Wert, der konform zum gewählten Datentyp ist
`fixed`	unveränderliche Wertbelegung
`name`	unqualifizierter Name des Attributes
`ref`	Verweis auf eine globale Attributdefinition
`type`	Deklaration des Datentyps für das Attribut

Vergleiche hierzu die DTD-Attribut-Deklaration.

XML-Schema

XML-Schema Teil 1: Attribute

Beispiel 1:

```
<xsd:element name="book">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="title" type="xsd:string"/>
      <xsd:element name="author" type="xsd:string"/>
    </xsd:sequence>
    <xsd:attribute name="isbn" type="isbnType" use="required"/>
  </xsd:complexType>
</xsd:element>
```

XML-Schema

XML-Schema Teil 1: Attribute

Beispiel 1:

```
<xsd:element name="book">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="title" type="xsd:string"/>
      <xsd:element name="author" type="xsd:string"/>
    </xsd:sequence>
    <xsd:attribute name="isbn" type="isbnType" use="required"/>
  </xsd:complexType>
</xsd:element>
```

Beispiel 2:

```
<xsd:element name="description">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute name="source" type="xsd:NMTOKEN"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```

Bemerkungen:

- Attribute dürfen nur von einfachem Typ sein.
- Beispiel 1 illustriert, dass Attribute immer am Ende einer Deklaration angegeben werden, also vor `</xsd:extension>` oder vor `</xsd:restriction>` bzw. `</xsd:complexType>`.
- Beispiel 2 illustriert, dass auch bei der Deklaration eines Elementtyps ohne Kindelemente eine Attributdeklaration immer dazu führt, dass mittels `</xsd:complexType>` ein komplexer Datentyp definiert werden muss.
- In der folgenden, nicht abgekürzten Deklaration für das Beispiel 1 zeigt sich die kanonische Herangehensweise bei der Deklaration von Elementtypen:

```
<xsd:element name="book">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
        <xsd:sequence>
          <xsd:element name="title" type="xsd:string"/>
          <xsd:element name="author" type="xsd:string"/>
        </xsd:sequence>
        <xsd:attribute name="isbn" type="isbnType" use="required"/>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
```

XML-Schema

Schemaentwurf [Vlist 2001]

Instanzdokument:

Being a Dog Is a Full-Time Job

von Charles M. Schulz

Darsteller 1:

Name:	Snoopy
befreundet mit:	Peppermint Patty
seit:	1990-10-04
Eigenschaften:	extroverted beagle

Darsteller 2:

Name:	Peppermint Patty
seit:	1996-08-22
Eigenschaften:	bold, brash and tomboyish

XML-Schema

Schemaentwurf [Vlist 2001]

Instanzdokument:

Being a Dog Is a Full-Time Job

von Charles M. Schulz

Darsteller 1:

Name:	Snoopy
befreundet mit:	Peppermint Patty
seit:	1990-10-04
Eigenschaften:	extroverted beagle

Darsteller 2:

Name:	Peppermint Patty
seit:	1996-08-22
Eigenschaften:	bold, brash and tomboyish

Mögliche DTD:

```
<!ELEMENT book (title, author, character+)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT character (name, since?, qualification*)>
...
```

XML-Schema

Schemaentwurf

XML-Source des Instanzdokuments:

```
<?xml version="1.0" encoding="UTF-8"?>
<book isbn="0836217462">

  <title>
  <author>

  <character>
    <name>
    <friend-of>
    <since>
    <qualification>
  </character>

  <character>
    <name>
    <since>
    <qualification>
  </character>

</book>
```

XML-Schema

Schemaentwurf

XML-Source des Instanzdokuments:

```
<?xml version="1.0" encoding="UTF-8"?>
<book isbn="0836217462">

  <title>Being a Dog Is a Full-Time Job</title>
  <author>Charles M. Schulz</author>

  <character>
    <name>Snoopy</name>
    <friend-of>Peppermint Patty</friend-of>
    <since>1950-10-04</since>
    <qualification>extroverted beagle</qualification>
  </character>

  <character>
    <name>Peppermint Patty</name>
    <since>1966-08-22</since>
    <qualification>bold, brash and tomboyish</qualification>
  </character>

</book>
```

XML-Schema

Entwurfsmuster 1a: „Russian Doll Design“

Deklarationen von Elementtypen isoliert betrachtet:

```
<xsd:element name="book">
  <xsd:complexType>
    <xsd:sequence>
      ...
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

XML-Schema

Entwurfsmuster 1a: „Russian Doll Design“

Deklarationen von Elementtypen isoliert betrachtet:

```
<xsd:element name="book">
  <xsd:complexType>
    <xsd:sequence>
      ...
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="title" type="xsd:string"/>
<xsd:element name="author" type="xsd:string"/>

<xsd:element name="character" minOccurs="0" maxOccurs="unbounded">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      ...
      <xsd:element name="qualification" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

XML-Schema

Entwurfsmuster 1a: „Russian Doll Design“ (Fortsetzung)

Ineinander geschachtelte Deklarationen von Elementtypen:

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="book">
    <xsd:complexType>
      <xsd:sequence>









      </xsd:sequence>

    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

XML-Schema

Entwurfsmuster 1a: „Russian Doll Design“ (Fortsetzung)

Ineinander geschachtelte Deklarationen von Elementtypen:

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="book">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="title" type="xsd:string"/>
        <xsd:element name="author" type="xsd:string"/>
      </xsd:sequence>
      <xsd:attribute name="isbn" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

XML-Schema

Entwurfsmuster 1a: „Russian Doll Design“ (Fortsetzung)

Ineinander geschachtelte Deklarationen von Elementtypen:

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="book">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="title" type="xsd:string"/>
        <xsd:element name="author" type="xsd:string"/>
        <xsd:element name="character" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="name" type="xsd:string"/>
              ...
              <xsd:element name="qualification" type="xsd:string"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
      <xsd:attribute name="isbn" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Bemerkungen:

- Bei dem Entwurfsmuster „Russian Doll Design“ ist das entstehende XML-Schema eng an das Instanzdokument angelehnt; die Elementtypen werden entsprechend ihrer Verwendung im Instanzdokument deklariert.
- Nachteil: Es können tiefgeschachtelte Schemata entstehen, die schwer zu verstehen und zu pflegen sind. Solche Schemata unterscheiden sich in ihrem Aufbau deutlich von den entsprechenden DTDs.

XML-Schema

Entwurfsmuster 1b: Verweis auf globale Deklarationen

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <!-- definition of simple type elements -->
  <xsd:element name="title" type="xsd:string"/>
  <xsd:element name="author" type="xsd:string"/>
  ...
  <xsd:element name="qualification" type="xsd:string"/>
  <xsd:attribute name="isbn" type="xsd:string"/>
```

XML-Schema

Entwurfsmuster 1b: Verweis auf globale Deklarationen

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <!-- definition of simple type elements -->
  <xsd:element name="title" type="xsd:string"/>
  <xsd:element name="author" type="xsd:string"/>
  ...
  <xsd:element name="qualification" type="xsd:string"/>
  <xsd:attribute name="isbn" type="xsd:string"/>

  <!-- definition of complex type elements -->
  <xsd:element name="character">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="name"/>
        ...
        <xsd:element ref="qualification"/>
        <!-- simple type elements are referenced with "ref" attribute -->
        <!-- definition of cardinality when elements are referenced -->
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

XML-Schema

Entwurfsmuster 1b: Verweis auf globale Deklarationen (Fortsetzung)

```
  <xsd:element name="book">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="title"/>
        <xsd:element ref="author"/>
        <xsd:element ref="character" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute ref="isbn"/>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

Bemerkungen:

- Wie bei dem „Russian Doll Design“ werden auch hier die Elementtypen entsprechend ihrer Verwendung im Instanzdokument deklariert. Schachtelungen werden dadurch vermieden, dass – ausgehend von den Blättern (also bottom-up) – für jede Ebene des Dokumentbaums eine Liste der in dieser Ebene neu hinzukommenden Elementtypen erstellt wird.
  Die Deklaration von Elementtypen mit Kindelementen geschieht mittels Verweise auf Elementtypen aus dieser Liste. Vergleiche hierzu das „Russian Doll Design“, bei dem statt der Verweise die gesamte Deklaration eingebettet wird.
- Man könnte die Entwurfsmuster (1a) und (1b) als „Entwurf auf Basis anonymer Datentypen“ bezeichnen: die Deklarationen der Elementtypen für `character` und `book` basieren auf einem direkt *(inline, ad-hoc)* definierten, anonymen Datentyp. Sind mehrere Elementtypen gleich aufgebaut, so besitzen sie bis auf ihren Namen die gleiche Deklaration, was zu Redundanz führt. Die Alternative besteht in der Definition eines benannten Datentyps.
- Durch das Referenzierungskonzept existiert eine erste Möglichkeit zur Wiederverwendung bereits im Schema definierter Elementtypen. Bei dieser einfachen Form der Textersetzung werden die Elementtypen literal, also mit ihrem Namen eingebunden. Im Grunde genommen ist diese Art der Wiederverwendung bereits mit den Mitteln von DTDs möglich. [Jeckle 2004]
- “Another authoring style, applicable when all element names are unique within a namespace, is to create schemas in which all elements are global. This is similar in effect to the use of `<!ELEMENT>` in a DTD.” [W3C]

XML-Schema

Entwurfsmuster 2: Definition benannter Datentypen

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <!-- definition of simple types -->
  <xsd:simpleType name="nameType">
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="32"/>
    </xsd:restriction>
  </xsd:simpleType>
```

XML-Schema

Entwurfsmuster 2: Definition benannter Datentypen

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <!-- definition of simple types -->
  <xsd:simpleType name="nameType">
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="32"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="sinceType">
    <xsd:restriction base="xsd:date"/>
  </xsd:simpleType>

  <xsd:simpleType name="descType">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>

  <xsd:simpleType name="isbnType">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[0-9]10"/>
    </xsd:restriction>
  </xsd:simpleType>
```

XML-Schema

Entwurfsmuster 2: Definition benannter Datentypen (Fortsetzung)

```
<!-- definition of complex types -->
<xsd:complexType name="characterType">
  <xsd:sequence>
    <xsd:element name="name" type="nameType"/>
    ...
    <xsd:element name="qualification" type="descType"/>
  </xsd:sequence>
</xsd:complexType>
```

XML-Schema

Entwurfsmuster 2: Definition benannter Datentypen (Fortsetzung)

```
  <!-- definition of complex types -->
  <xsd:complexType name="characterType">
    <xsd:sequence>
      <xsd:element name="name" type="nameType"/>
      ...
      <xsd:element name="qualification" type="descType"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="bookType">
    <xsd:sequence>
      <xsd:element name="title" type="nameType"/>
      <xsd:element name="author" type="nameType"/>
      <xsd:element name="character" type="characterType" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="isbn" type="isbnType" use="required"/>
  </xsd:complexType>




</xsd:schema>
```

XML-Schema

Entwurfsmuster 2: Definition benannter Datentypen (Fortsetzung)

```
  <!-- definition of complex types -->
  <xsd:complexType name="characterType">
    <xsd:sequence>
      <xsd:element name="name" type="nameType"/>
      ...
      <xsd:element name="qualification" type="descType"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="bookType">
    <xsd:sequence>
      <xsd:element name="title" type="nameType"/>
      <xsd:element name="author" type="nameType"/>
      <xsd:element name="character" type="characterType" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="isbn" type="isbnType" use="required"/>
  </xsd:complexType>

  <!-- The "book" element is of the data type "bookType" -->
  <xsd:element name="book" type="bookType"/>

</xsd:schema>
```

XML-Schema

Gegenüberstellung der Entwurfsmuster

1. Deklaration eines Elementtyps mit anonymem Datentyp:

```
<xsd:element name="bondCar">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="color" type="xsd:string"/>
      <xsd:element name="enginepower" type="xsd:integer"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

2. Deklaration eines Elementtyps mit benanntem Datentyp:

```
<xsd:complexType name="automobile">
  <xsd:sequence>
    <xsd:element name="color" type="xsd:string"/>
    <xsd:element name="enginepower" type="xsd:integer"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:element name="bondCar" type="automobile"/>
```

XML-Schema

Quellen zum Nachlernen und Nachschlagen im Web: Referenz

- D. C. Fallside. *XML Schema Part 0: Primer.*
  W3C Recommendation. www.w3.org/TR/xmlschema-0
- S. Gao et al. *XML Schema Definition Language (XSD) 1.1 Part 1: Structures.*
  W3C Recommendation. www.w3.org/TR/xmlschema11-1/
- D. Peterson et al. *XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes.*
  W3C Recommendation. www.w3.org/TR/xmlschema11-2/
- MSDN-Library. *Referenz zu XML-Schemata (XSD).*
  msdn2.microsoft.com/de-de/library/ms256235

XML-Schema

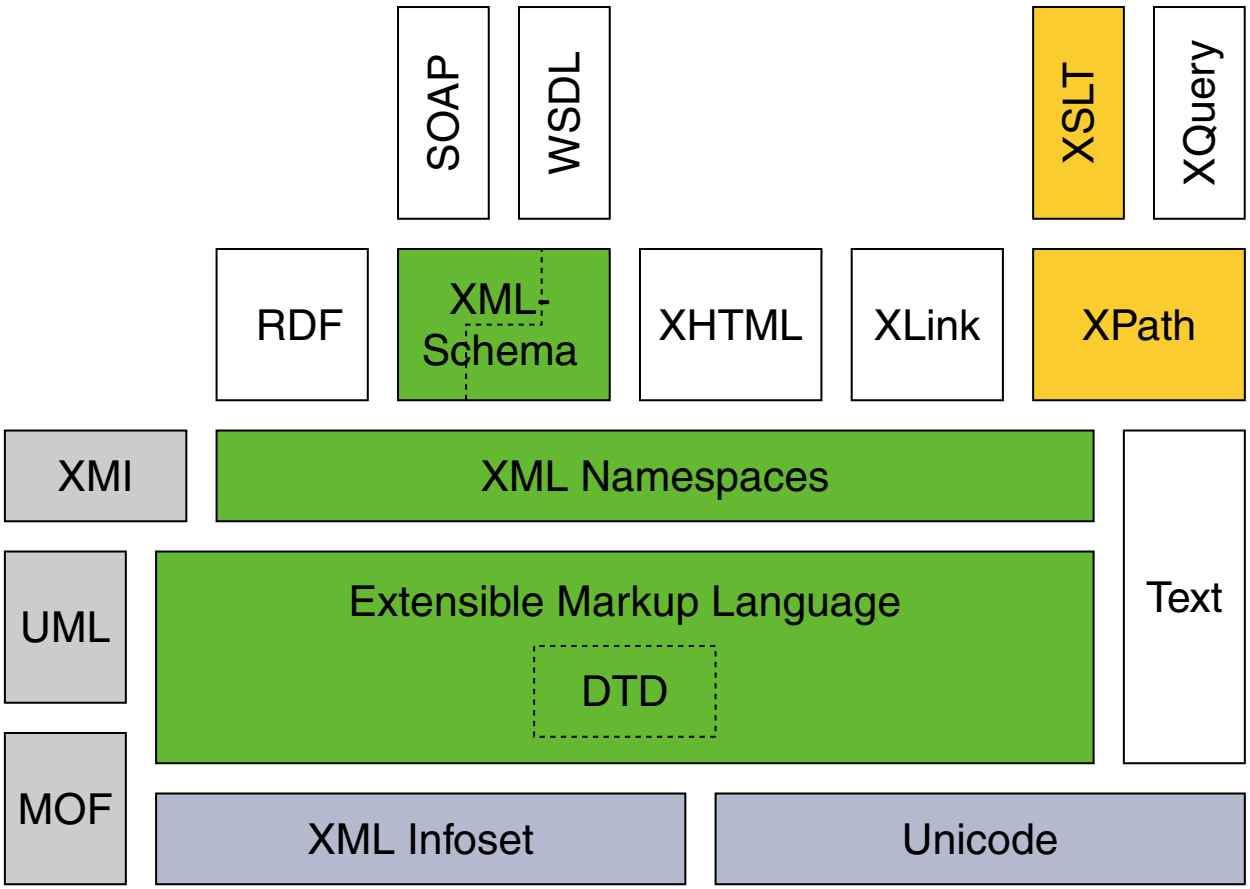
Quellen zum Nachlernen und Nachschlagen im Web: Usage

- E. v. d. Vlist. *Using W3C XML Schema.*
  www.xml.com/lpt/a/2000/11/29/schemas/part1.html
- Cover Pages.
  xml.coverpages.org/schemas.html
- Liquid Technologies.
  www.liquid-technologies.com/Tutorials/XmlSchemas
- W3 Schools.
  www.w3schools.com/schema

Kapitel WT:III (Fortsetzung)

Die XSL-Familie

Einordnung [Jeckle 2004]



Die XSL-Familie

Historie

> *"XSL is a family of recommendations for defining XML document transformation and presentation. It consists of three parts . . . "* [W3C]

1999 XSL, Extensible Stylesheet Language. [W3C FAQ, TR]

XSL wurde aufgeteilt in:

1999 XSLT 1.0. Transformation von XML-Dokumenten.

1999 XPath 1.0. Zugriff und Referenzierung in XML-Dokumenten.

2000 XSL-FO 1.0. Formatting Objects zum Layout von Seiten.

2007 XSLT 2.0 Recommendation. [W3C Overview, TR]

2010 XPath 2.0 Recommendation. [W3C]

2010 XQuery 1.0 Recommendation. [W3C Overview, TR]

2012 XSL-FO 2.0 Working Draft. [W3C]

Bemerkungen:

- XSL *(Extensible Style Language)* ist eine Formatierungssprache, die einem mit XML-basierten Markup versehenen Dokument eine Layout-Vorschrift zuordnet.
- XSL benutzt dasselbe Formattierungsmodell wie CSS *(Cascading Stylesheets)* und ermöglicht die Einbettung von Programm-Code zur Durchführung komplexer Ausgabetransformationen.
- CSS versus XML. Why two Style Sheet languages? [W3C]

Die XSL-Familie

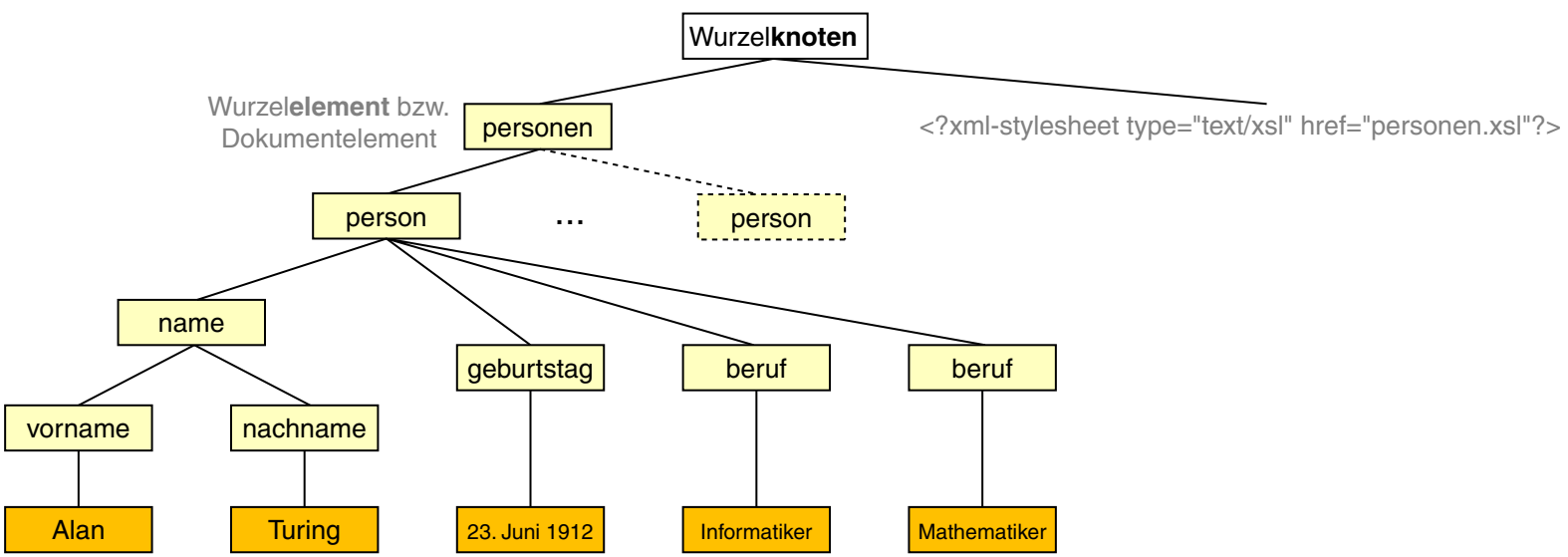
Verwendung von XPath

XSLT	Finden und Auswählen von Elementen im Eingabedokument, die in das Ausgabedokument kopiert werden.
XQuery	Finden und Auswählen von Elementen.
XPointer	Identifikation einer Stelle im XML-Dokument, auf die ein XLink verweist.
XML-Schema	Formulierung von Constraints hinsichtlich der Eindeutigkeit oder der Identität von Elementen.
XForm	Bindung von Formularsteuerungen an Instanzdaten; Formulierung von Werte-Constraints und Berechnungen.

Die XSL-Familie

XML-Knotentypen unter dem XPath-Modell

1. Wurzelknoten
2. Elementknoten
3. Textknoten
4. Attributknoten
5. Kommentarknoten
6. Verarbeitungsanweisungsknoten
7. Namensraumknoten



Bemerkungen:

- Der Wurzel*knoten* eines XML-Dokuments ist nicht identisch mit dem Wurzel*element*: Der Wurzelknoten entspricht dem *Document Information Item* des XML Information Sets. Das Wurzelelement hingegen ist das erste benannte Element des Dokuments und wird durch ein *Element Information Item* dargestellt.
- XPath dient der Navigation in Dokumenten und der Auswahl von Dokumentbestandteilen; XPath ist keine Datenmanipulationssprache.
- XPath-Ausdrücke können zu einzelnen Knoten (XML-Element, XML-Attribut), zu Knotenmengen, zu Zeichenketten, zu Zahlen und zu Bool'schen Werten evaluieren. Deshalb stellt XPath Funktionen zum Zugriff auf Knotenmengen und zur Manipulation verschiedener Datentypen zur Verfügung.
- Wiederholung. Das W3C hat mittlerweile drei Datenmodelle für XML-Dokumente definiert: XML Information Set, XPath, Document Object Model (DOM). Das XPath-Datenmodell basiert auf einer Baumstruktur, die bei der Abfrage eines XML-Dokuments durchlaufen wird und ist dem XML Information Set ähnlich; DOM ist der Vorläufer beider Datenmodelle. DOM und das XPath-Datenmodell können als Interpretationen des XML Information Sets betrachtet werden. [MSDN]

Die XSL-Familie

XPath-Lokalisierungspfade

- Ein Lokalisierungspfad spezifiziert eine eventuell leere *Menge* von Knoten in einem XML-Dokument.
- Ein Lokalisierungspfad setzt sich aus aufeinander folgenden Lokalisierungsschritten *(Location steps)* zusammen.
- Jeder Lokalisierungsschritt wird relativ zu einem bestimmten Knoten des XML-Dokuments ausgewertet, der dann als aktueller Knoten *(Current node)* oder Kontextknoten bezeichnet wird.

Die XSL-Familie

XPath-Lokalisierungspfade

- Ein Lokalisierungspfad spezifiziert eine eventuell leere *Menge* von Knoten in einem XML-Dokument.
- Ein Lokalisierungspfad setzt sich aus aufeinander folgenden Lokalisierungsschritten *(Location steps)* zusammen.
- Jeder Lokalisierungsschritt wird relativ zu einem bestimmten Knoten des XML-Dokuments ausgewertet, der dann als aktueller Knoten *(Current node)* oder Kontextknoten bezeichnet wird.

- Lokalisierungsschritte werden durch Schrägstriche *(Slashes)* getrennt:

$$\ldots /\mathit{Schritt\_i} /\mathit{Schritt\_i{+}1} / \ldots$$

- Beginnt ein Lokalisierungspfad mit einem Schrägstrich, bezeichnet dieser den Wurzelknoten. Der Wurzelknoten ist dann aktueller Knoten zum ersten Lokalisierungsschritt:

$$/\mathit{Schritt\_1}$$

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

(a) `/personen/person`

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

(a) `/personen/person`

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

(a) `/personen/person`

(b) `/personen/person[1]/name/vorname`

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

(a) `/personen/person`

(b) `/personen/person[1]/name/vorname`

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

(a) `/personen/person`

(b) `/personen/person[1]/name/vorname`

(c) `/personen/person[1]/beruf`

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

(a) `/personen/person`

(b) `/personen/person[1]/name/vorname`

(c) `/personen/person[1]/beruf`

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

Allgemeine Form eines Lokalisierungsschritts:

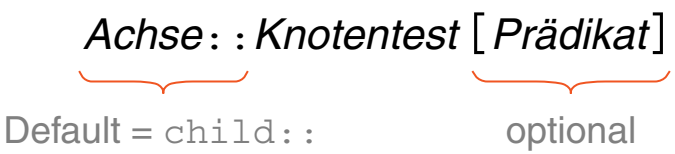
Achse::*Knotentest* [*Prädikat*]

Default = `child::` optional

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

Allgemeine Form eines Lokalisierungsschritts:



1. Achsen. Spezifizieren Knotenmengen relativ zum aktuellen Knoten. Es werden 13 Achsen unterschieden.
2. Knotentests. Filtern die durch eine Achse (1.) spezifizierte Knotenmenge weiter. Hierzu gibt es für jeden Knotentyp ein Testschema.

   Beispiele:

   - ein qualifizierender Name ∼ Test auf Knoten mit diesem Namen
   - die Funktion `text()` ∼ Test auf Textknoten

3. Prädikate. Filtern die durch Achse (1.) und Knotentest (2.) spezifizierte Knotenmenge weiter. Jeder gültige XPath-Ausdruck kann Prädikat sein.

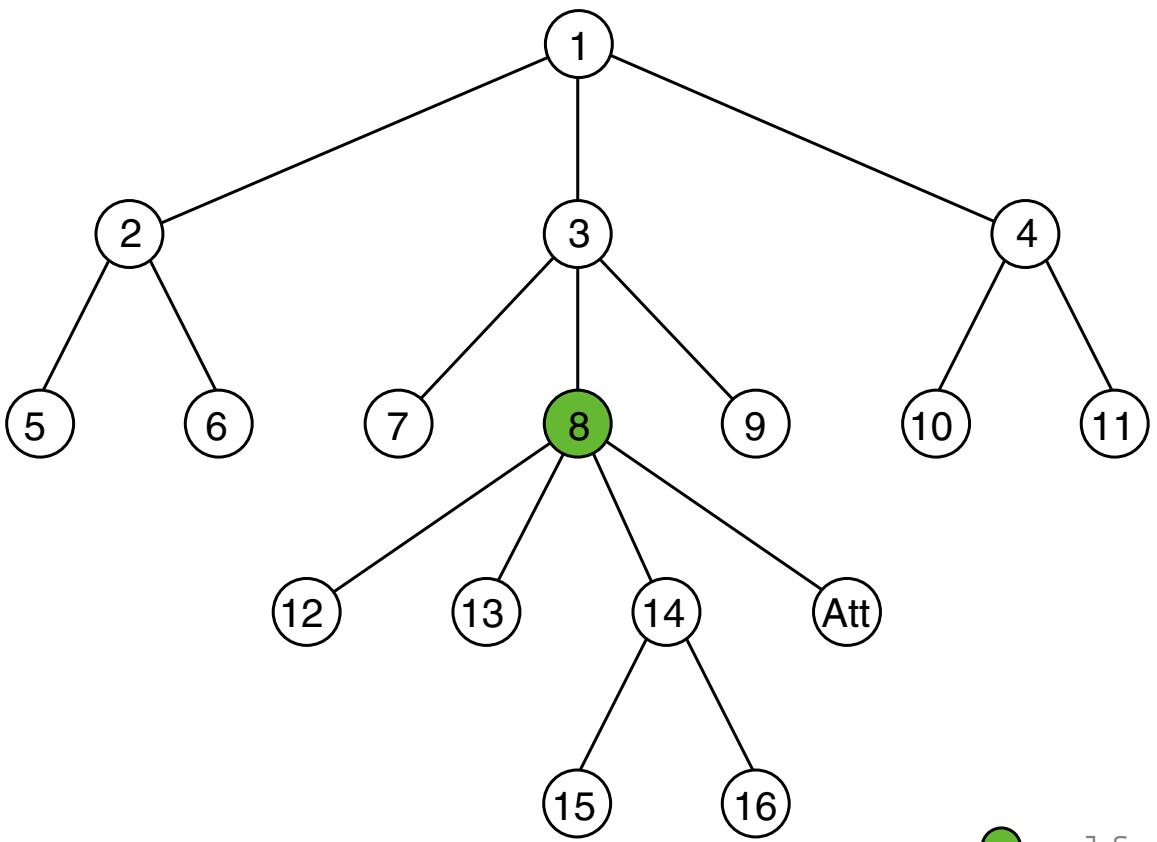
   Beispiele: Test auf Kindknoten, Position bzw. Index

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

Illustration wichtiger Achsen [Jeckle 2004] :

```
<node1>

  <node2>
    <node5/>
    <node6/>
  </node2>

  <node3>
    <node7/>
    <node8 Att="42">
      <node12/>
      <node13/>
      <node14>
        <node15/>
        <node16/>
      </node14>
    </node8>
    <node9/>
  </node3>

  <node4>
    <node10/>
    <node11/>
  </node4>

</node1>
```

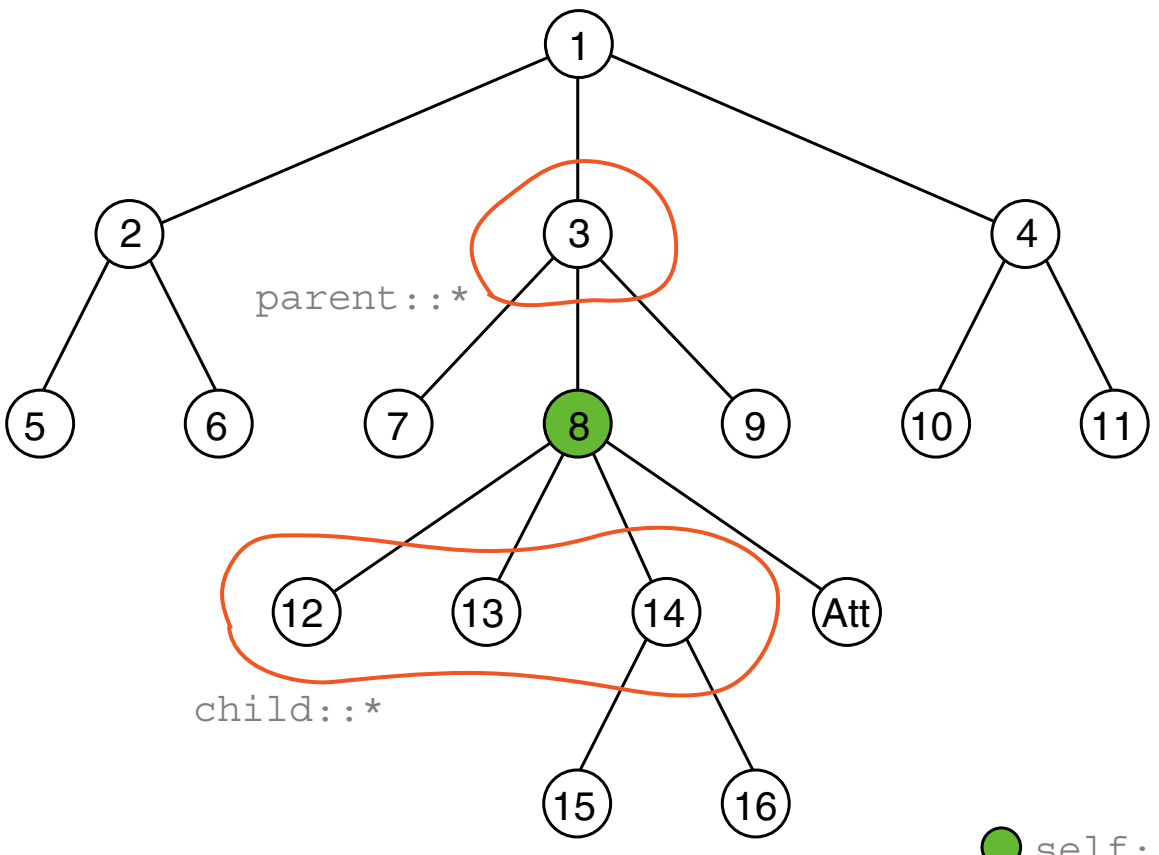


Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

Illustration wichtiger Achsen [Jeckle 2004] :

```
<node1>

  <node2>
    <node5/>
    <node6/>
  </node2>

  <node3>
    <node7/>
    <node8 Att="42">
      <node12/>
      <node13/>
      <node14>
        <node15/>
        <node16/>
      </node14>
    </node8>
    <node9/>
  </node3>

  <node4>
    <node10/>
    <node11/>
  </node4>

</node1>
```

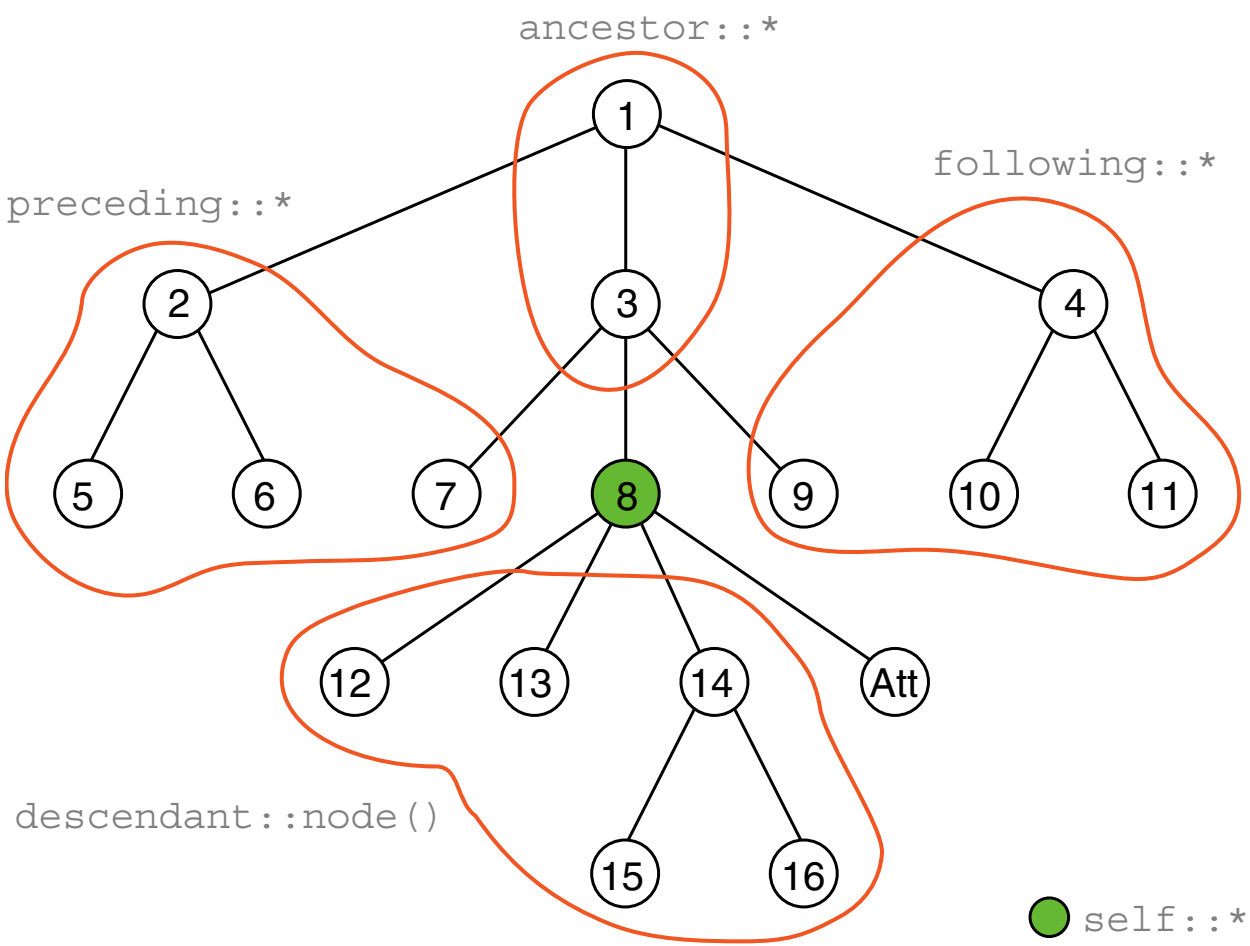


Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

Illustration wichtiger Achsen [Jeckle 2004] :

```
<node1>

  <node2>
    <node5/>
    <node6/>
  </node2>

  <node3>
    <node7/>
    <node8 Att="42">
      <node12/>
      <node13/>
      <node14>
        <node15/>
        <node16/>
      </node14>
    </node8>
    <node9/>
  </node3>

  <node4>
    <node10/>
    <node11/>
  </node4>

</node1>
```



Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

- Schreibweisen häufig verwendeter Lokalisierungsschritte:

kurz	lang	Semantik
`/./`	`/self::node()/`	Kontextknoten bzw. aktueller Knoten
`/../`	`/parent::node()/`	Elternknoten des aktuellen Knotens
`//`	`/descendant-or-self::node()/`	alle Nachkommen des aktuellen Knotens einschließlich diesem

- Wildcards für Namen in Knotentests:

`node()`	beliebiger Knotentyp (kein Attributknoten)
`*`	Elementknoten
`text()`	Textknoten
`comment()`	Kommentarknoten
`processing-instruction()`	Verarbeitungsanweisungsknoten
`@*`	Attributknoten

- Spezifikation von alternativen Lokalisierungspfaden:

  Pfad_1 | *Pfad_2* | ... | *Pfad_n*

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

(a) `//person/name/descendant::*`

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

(a) `//person/name/descendant::*`

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

(a) `//person/name/descendant::*`

(b) `//geburtstag/parent::*/name[1]`

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

(a) `//person/name/descendant::*`

(b) `//geburtstag/parent::*/name[1]`

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

(a) `//person/name/descendant::*`

(b) `//geburtstag/parent::*/name[1]`

(c) `/personen/child::name`

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

(a) `//person/name/descendant::*`

(b) `//geburtstag/parent::*/name[1]`

(c) `/personen/child::name`

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

(a) `//person/name/descendant::*`

(b) `//geburtstag/parent::*/name[1]`

(c) `/personen/child::name`

(d) `/personen/descendant::name`

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

(a) `//person/name/descendant::*`

(b) `//geburtstag/parent::*/name[1]`

(c) `/personen/child::name`

(d) `/personen/descendant::name`

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

(a) `//person/name/descendant::*`

(b) `//geburtstag/parent::*/name[1]`

(c) `/personen/child::name`

(d) `/personen/descendant::name`

(e) `//person[geburtstag!='unknown']`

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Beispiele für Lokalisierungspfade:

(a) `//person/name/descendant::*`

(b) `//geburtstag/parent::*/name[1]`

(c) `/personen/child::name`

(d) `/personen/descendant::name`

(e) `//person[geburtstag!='unknown']`

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

Algorithmus zur Auswertung eines Lokalisierungspfades:

. . . / *Schritt_i* / *Schritt_i+1* / . . .

1. Die Auswertung der Lokalisierungsschritte geschieht von links nach rechts.
2. Jeder Lokalisierungsschritt spezifiziert eine Knotenmenge M.

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

Algorithmus zur Auswertung eines Lokalisierungspfades:

. . . / *Schritt_i* / *Schritt_i+1* / . . .

1. Die Auswertung der Lokalisierungsschritte geschieht von links nach rechts.
2. Jeder Lokalisierungsschritt spezifiziert eine Knotenmenge M.
3. Jeder Knoten n der Knotenmenge M_i des Lokalisierungsschritts i wird als Kontextknoten hinsichtlich des Lokalisierungsschritts $i+1$ interpretiert und spezifiziert im Lokalisierungsschritt $i+1$ die Knotenmenge M_{i_n}.
4. Die Vereinigung der Mengen M_{i_n}, $n \in M_i$, bildet die Knotenmenge M_{i+1} des Lokalisierungsschritts $i+1$.

Die XSL-Familie

Lokalisierungspfade (Fortsetzung)

Knoten

```
1
2  <?xml version="1.0" standalone="no" ?>
3  <?xml-stylesheet type="text/xsl" ...?>

4  <personen>
5    <person>
6      <name>
7        <vorname>Alan</vorname>
8        <nachname>Turing</nachname>
6      </name>
9      <geburtstag>23. Juni 1912</geburtstag>
10     <beruf>Mathematiker</beruf>
11     <beruf>Informatiker</beruf>
5    </person>

12   <person>
13     <name>
14       <vorname>Judea</vorname>
15       <nachname>Pearl</nachname>
13     </name>
16     <geburtstag>unknown</geburtstag>
17     <beruf>Informatiker</beruf>
12   </person>
4  </personen>
```

Illustration des Algorithmus:

```
//person/name/descendant::*
```

$M_1 = \{1\}$

$M_2 = \{1, \ldots, 17\}$

$M_3 = \{5, 12\}$

$M_4 = \{6, 13\}$

$M_5 = \{7, 8, 14, 15\}$

Die XSL-Familie

Lokalisierungspfade (Fortsetzung)

Knoten

```
 1
 2 <?xml version="1.0" standalone="no" ?>
 3 <?xml-stylesheet type="text/xsl" ...?>

 4 <personen>
 5   <person>
 6     <name>
 7       <vorname>Alan</vorname>
 8       <nachname>Turing</nachname>
 6     </name>
 9     <geburtstag>23. Juni 1912</geburtstag>
10     <beruf>Mathematiker</beruf>
11     <beruf>Informatiker</beruf>
 5   </person>

12   <person>
13     <name>
14       <vorname>Judea</vorname>
15       <nachname>Pearl</nachname>
13     </name>
16     <geburtstag>unknown</geburtstag>
17     <beruf>Informatiker</beruf>
12   </person>
 4 </personen>
```

Illustration des Algorithmus:

`/ /person/name/descendant::*`

$M_1 = \{1\}$

$M_2 = \{1, \ldots, 17\}$

$M_3 = \{5, 12\}$

$M_4 = \{6, 13\}$

$M_5 = \{7, 8, 14, 15\}$

Die XSL-Familie

Lokalisierungspfade (Fortsetzung)

Knoten

```
 1
 2 <?xml version="1.0" standalone="no" ?>
 3 <?xml-stylesheet type="text/xsl" ...?>

 4 <personen>
 5   <person>
 6     <name>
 7       <vorname>Alan</vorname>
 8       <nachname>Turing</nachname>
 6     </name>
 9     <geburtstag>23. Juni 1912</geburtstag>
10     <beruf>Mathematiker</beruf>
11     <beruf>Informatiker</beruf>
 5   </person>

12   <person>
13     <name>
14       <vorname>Judea</vorname>
15       <nachname>Pearl</nachname>
13     </name>
16     <geburtstag>unknown</geburtstag>
17     <beruf>Informatiker</beruf>
12   </person>
 4 </personen>
```

Illustration des Algorithmus:

`//person/name/descendant::*`

$M_1 = \{1\}$

$M_2 = \{1, \ldots, 17\}$

$M_3 = \{5, 12\}$

$M_4 = \{6, 13\}$

$M_5 = \{7, 8, 14, 15\}$

Die XSL-Familie

Lokalisierungspfade (Fortsetzung)

Knoten

```
 1
 2 <?xml version="1.0" standalone="no" ?>
 3 <?xml-stylesheet type="text/xsl" ...?>

 4 <personen>
 5   <person>
 6     <name>
 7       <vorname>Alan</vorname>
 8       <nachname>Turing</nachname>
 6     </name>
 9     <geburtstag>23. Juni 1912</geburtstag>
10     <beruf>Mathematiker</beruf>
11     <beruf>Informatiker</beruf>
 5   </person>

12   <person>
13     <name>
14       <vorname>Judea</vorname>
15       <nachname>Pearl</nachname>
13     </name>
16     <geburtstag>unknown</geburtstag>
17     <beruf>Informatiker</beruf>
12   </person>
 4 </personen>
```

Illustration des Algorithmus:

`//person/name/descendant::*`

$M_1 = \{1\}$

$M_2 = \{1, \ldots, 17\}$

$M_3 = \{5, 12\}$

$M_4 = \{6, 13\}$

$M_5 = \{7, 8, 14, 15\}$

Die XSL-Familie

Lokalisierungspfade (Fortsetzung)

Knoten

```
1
2  <?xml version="1.0" standalone="no" ?>
3  <?xml-stylesheet type="text/xsl" ...?>

4  <personen>
5    <person>
6      <name>
7        <vorname>Alan</vorname>
8        <nachname>Turing</nachname>
6      </name>
9      <geburtstag>23. Juni 1912</geburtstag>
10     <beruf>Mathematiker</beruf>
11     <beruf>Informatiker</beruf>
5    </person>

12   <person>
13     <name>
14       <vorname>Judea</vorname>
15       <nachname>Pearl</nachname>
13     </name>
16     <geburtstag>unknown</geburtstag>
17     <beruf>Informatiker</beruf>
12   </person>
4  </personen>
```

Illustration des Algorithmus:

`//person/name/descendant::*`

$M_1 = \{1\}$

$M_2 = \{1, \ldots, 17\}$

$M_3 = \{5, 12\}$

$M_4 = \{6, 13\}$

$M_5 = \{7, 8, 14, 15\}$

Die XSL-Familie

Lokalisierungspfade (Fortsetzung)

Knoten

```
1
2  <?xml version="1.0" standalone="no" ?>
3  <?xml-stylesheet type="text/xsl" ...?>

4  <personen>
5    <person>
6      <name>
7        <vorname>Alan</vorname>
8        <nachname>Turing</nachname>
6      </name>
9      <geburtstag>23. Juni 1912</geburtstag>
10     <beruf>Mathematiker</beruf>
11     <beruf>Informatiker</beruf>
5    </person>

12   <person>
13     <name>
14       <vorname>Judea</vorname>
15       <nachname>Pearl</nachname>
13     </name>
16     <geburtstag>unknown</geburtstag>
17     <beruf>Informatiker</beruf>
12   </person>
4  </personen>
```

Illustration des Algorithmus:

`//person/name/descendant::*`

$M_1 = \{1\}$

$M_2 = \{1, \ldots, 17\}$

$M_3 = \{5, 12\}$

$M_4 = \{6, 13\}$

$M_5 = \{7, 8, 14, 15\}$

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

Komplexes Beispiel:

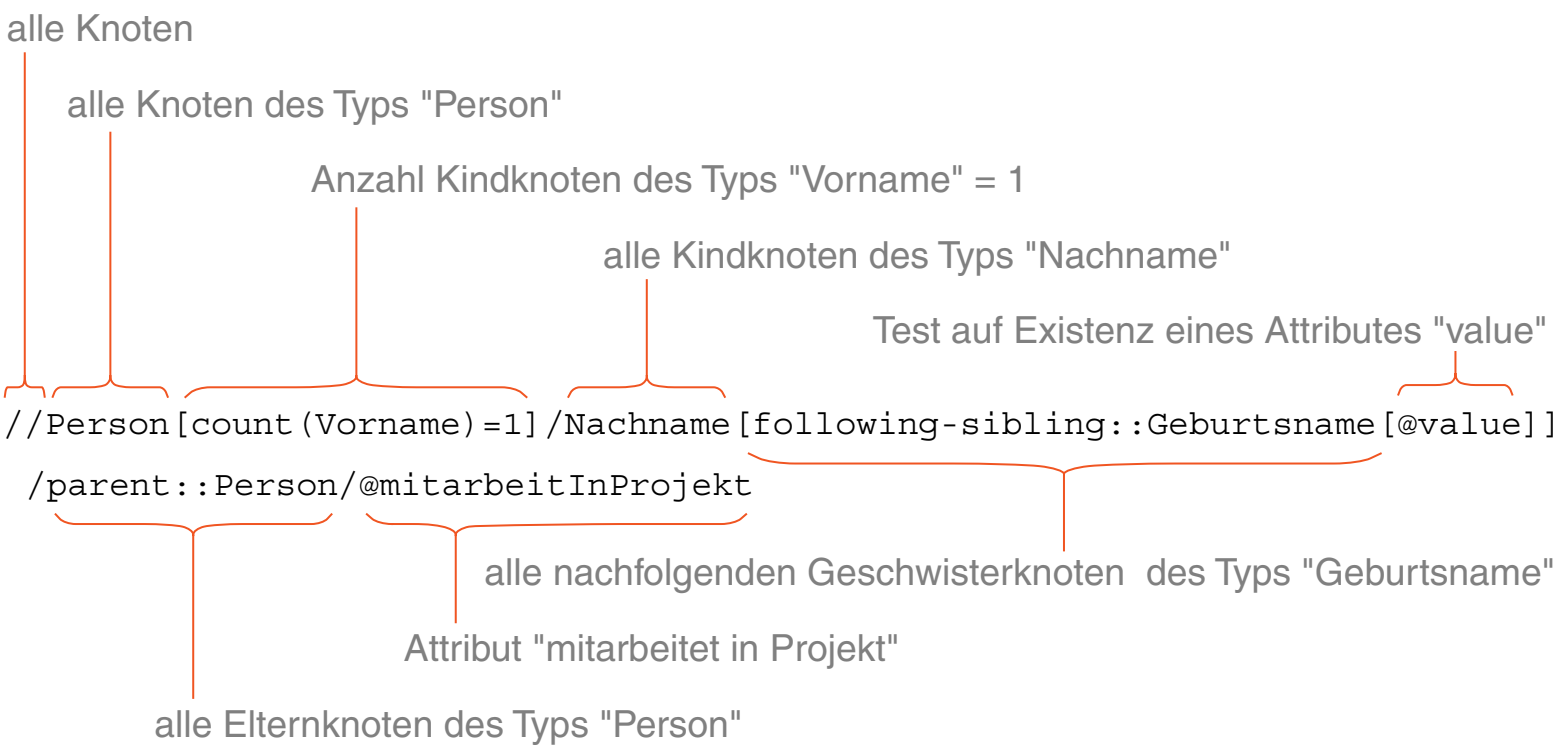
```
//Person[count(Vorname)=1]/Nachname[following-sibling::Geburtsname[@value]]
 /parent::Person/@mitarbeitInProjekt
```

[Jeckle 2004]

Die XSL-Familie

XPath-Lokalisierungspfade (Fortsetzung)

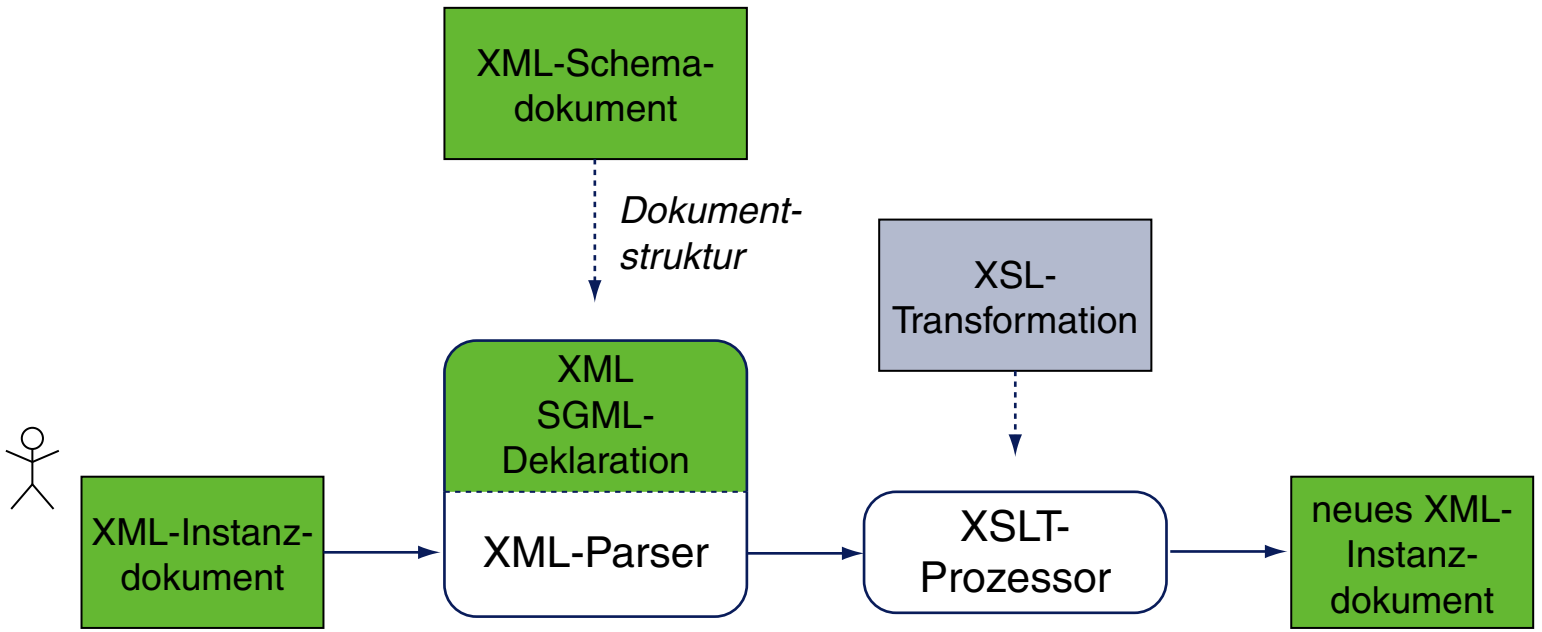
Komplexes Beispiel:



[Jeckle 2004]

Die XSL-Familie

XSL Transformation



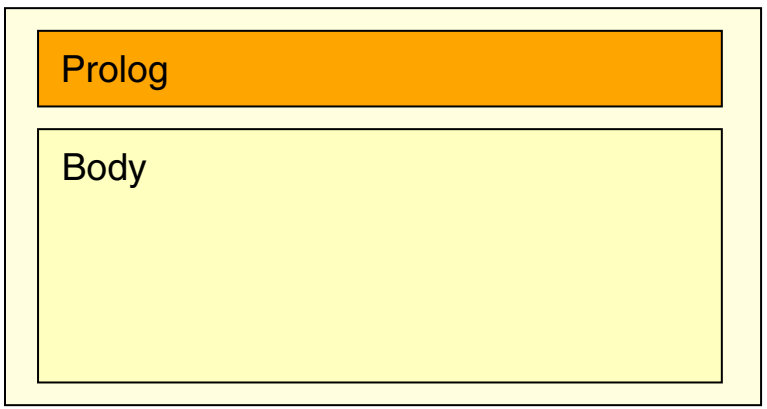
XSLT ist eine Turing-vollständige Programmiersprache zur Transformation wohlgeformter XML-Dokumente in andere XML-Dokumente. Ein XSLT-Programm liegt üblicherweise als XSL-Stylesheet vor.

Die Transformation umfasst die Selektion von Teilen des Eingabedokuments, deren Umordnung sowie die Generierung neuer Inhalte aus den bestehenden.

Die XSL-Familie

Aufbau eines XSL-Stylesheets

XSL-Stylesheets sind XML-Dokumente:




```
<?xml version="1.0"?>


<xsl:stylesheet xmlns:xsl=...>
  <xsl:template match=...>
      ...
  </xsl:template>
  ...
</xsl:stylesheet>
```

- Wurzelelement jedes XSL-Schemas ist das Element `<xsl:stylesheet>` oder synonym `<xsl:transform>`.
- Alle Definitionen von Transformationsvorschriften sind Kindelemente von `<xsl:stylesheet>` bzw. `<xsl:transform>`.
- Vergleiche hierzu XML-Schema und XML-Grundlagen.

Bemerkungen:

- Das Vokabular zur Definition von XSL-Stylesheets gehört zum Namensraum `http://www.w3.org/1999/XSL/Transform`. Übliches Präfix ist `xsl:`, kann aber beliebig gewählt werden. Wird der offizielle Namensraum gebunden, ist auch das Attribut `version="1.0"` anzugeben.
- Die Dateiendung einer XSL-Stylesheet-Datei ist `.xsl`.
- Aufbau einer realen Turingmaschine.

Die XSL-Familie

XML-Beispieldokument

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Die XSL-Familie

Elemente eines XSL-Stylesheets

Das einfachste Stylesheet:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
</xsl:stylesheet>
```

Die XSL-Familie

Elemente eines XSL-Stylesheets

Das einfachste Stylesheet:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
</xsl:stylesheet>
```

Die XSL-Familie

Elemente eines XSL-Stylesheets

Das einfachste Stylesheet:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
</xsl:stylesheet>
```

Angewandt auf das Beispieldokument:

```
<?xml version="1.0" encoding="UTF-8"?>

     Alan
     Turing

    23. Juni 1912
    Mathematiker
    Informatiker

     Judea
     Pearl

    unknown
    Informatiker
```

Bemerkungen:

- In diesem Beispiel enthält das Stylesheet keine matchende Template-Regel. Die Ausgabe entsteht, weil in einer solchen Situation vom XSLT-Prozessor das Built-in-Template zur Ausgabe von Text- und Attributknoten angewandt wird.
- Diejenigen Konstrukte eines XML-Dokuments, die nicht zu einem der sieben Knotentypen des XPath-Modells gehören, werden unverändert übernommen. Hierzu zählt u.a. die `<?xml ?>`-Deklaration.
- Die Verknüpfung von XML-Dokument und XSL-Stylesheet kann explizit, in Form von Parametern für den XSLT-Prozessor, aber auch implizit geschehen: Die Zeile `<?xml-stylesheet type="text/xsl" href="..."?>` im Prolog eines XML-Dokuments deklariert ein Stylesheet. Vergleiche hierzu die Stylesheet-Deklaration in HTML-Dokumenten.
- Aufruf des XSLT-Prozessors Xalan über die Kommandozeile:

  ```
  java org.apache.xalan.xslt.Process -in personen.xml -xsl tiny.xsl
  ```

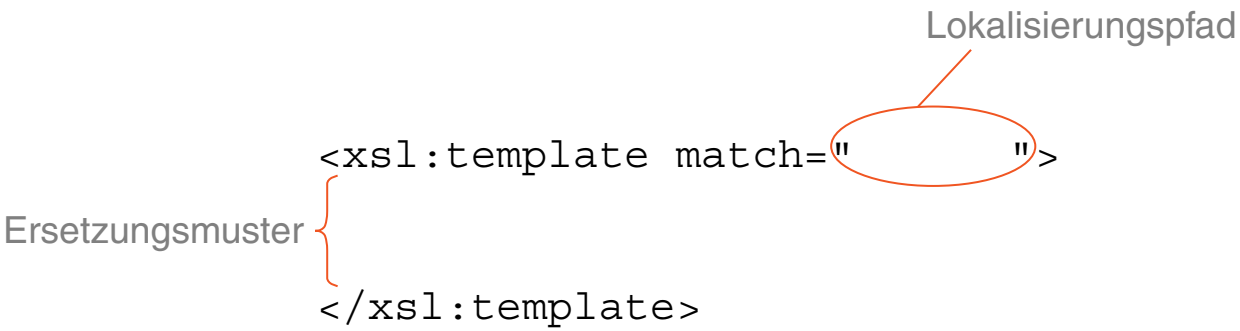
  Hierfür muss der Ort der Xalan-Bibliothek `xalan.jar` im Classpath spezifiziert sein. Alternativ der Aufruf mit expliziter Angabe der Xalan-Bibliothek:

  ```
  java -cp /usr/share/java/xalan.jar ...
  ```

Die XSL-Familie

Elemente eines XSL-Stylesheets (Fortsetzung)

Wichtigstes Stylesheet-Element ist die Template-Regel *(Template)*:

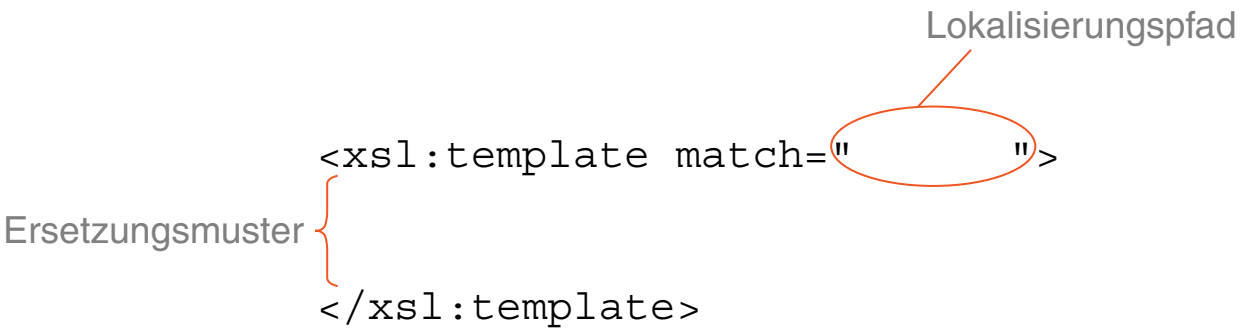


- Der Lokalisierungspfad des `match`-Attributs spezifiziert (ausgehend von dem Kontextknoten) eine Knotenmenge M.
- Eine Template-Regel matched einen Knoten n genau dann, falls zu irgend einem Zeitpunkt der Verarbeitung n ein Element der Menge M wird.

Die XSL-Familie

Elemente eines XSL-Stylesheets (Fortsetzung)

Wichtigstes Stylesheet-Element ist die Template-Regel *(Template)*:



- Der Lokalisierungspfad des `match`-Attributs spezifiziert (ausgehend von dem Kontextknoten) eine Knotenmenge M.
- Eine Template-Regel matched einen Knoten n genau dann, falls zu irgend einem Zeitpunkt der Verarbeitung n ein Element der Menge M wird.
- Wird ein Template auf einen Knoten n angewandt, behandelt das Ersetzungsmuster den gesamten Teilbaum des XML-Dokuments, der Knoten n als Wurzel hat. Dieser Teilbaum gilt als abgearbeitet.

Bemerkungen:

- Der Wert des `match`-Attributes im `<xsl:template>`-Element ist ein Lokalisierungspfad in eingeschränkter XPath-Syntax.
- Um die Knotenmenge M zu spezifizieren, für deren Elemente eine Template-Regel matched, sind alternative Pfadangaben möglich. Beispielsweise spezifizieren die Ausdrücke `match`="//*Elementname*" und `match`="*Elementname*" dieselbe Knotenmenge. D.h., ein relativer Lokalisierungspfad des `<xsl:template>`-Elements kann wie der entsprechende absolute, durch "//" eingeleitete Lokalisierungspfad aufgefasst werden.

Die XSL-Familie

Elemente eines XSL-Stylesheets (Fortsetzung)

Stylesheet mit literaler Ausgabe:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=...>

  <xsl:template match="person">
    <xsl:text>Person found!</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Angewandt auf das Beispieldokument:

Die XSL-Familie

Elemente eines XSL-Stylesheets (Fortsetzung)

Stylesheet mit literaler Ausgabe:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=...>

  <xsl:template match="person">
    <xsl:text>Person found!</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Angewandt auf das Beispieldokument:

```
Person found!

Person found!
```

Die XSL-Familie

Elemente eines XSL-Stylesheets (Fortsetzung)

Stylesheet zum Kopieren der Elemente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=...>

  <xsl:template match="person">
    <xsl:copy-of select="self::*"/>
  </xsl:template>

</xsl:stylesheet>
```

Angewandt auf das Beispieldokument:

Die XSL-Familie

Elemente eines XSL-Stylesheets (Fortsetzung)

Stylesheet zum Kopieren der Elemente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=...>

  <xsl:template match="person">
    <xsl:copy-of select="self::*"/>
  </xsl:template>

</xsl:stylesheet>
```

Angewandt auf das Beispieldokument:

```
<person>
  <name>
    <vorname>Alan</vorname>
    <nachname>Turing</nachname>
  </name>
  <geburtstag>23. Juni 1912</geburtstag>
  <beruf>Mathematiker</beruf>
  <beruf>Informatiker</beruf>
</person>
...
```

Die XSL-Familie

Elemente eines XSL-Stylesheets (Fortsetzung)

Stylesheet zum Kopieren der Elementinhalte:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=...>

  <xsl:template match="person">
    <xsl:value-of select="self::*"/>
  </xsl:template>

</xsl:stylesheet>
```

Angewandt auf das Beispieldokument:

Die XSL-Familie

Elemente eines XSL-Stylesheets (Fortsetzung)

Stylesheet zum Kopieren der Elementinhalte:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=...>

  <xsl:template match="person">
    <xsl:value-of select="self::*"/>
  </xsl:template>

</xsl:stylesheet>
```

Angewandt auf das Beispieldokument:

```
    Alan
    Turing

  23. Juni 1912
  Mathematiker
  Informatiker

  ...
```

Die XSL-Familie

Elemente eines XSL-Stylesheets (Fortsetzung)

Stylesheet zur Elementselektion mittels leerer Template-Regeln:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=...>

  <xsl:template match="name">
    <xsl:value-of select="nachname"/>
    <xsl:text>, </xsl:text>
    <xsl:value-of select="vorname"/>
  </xsl:template>

  <xsl:template match="geburtstag"/>
  <xsl:template match="beruf"/>

</xsl:stylesheet>
```

Angewandt auf das Beispieldokument:

Die XSL-Familie

Elemente eines XSL-Stylesheets (Fortsetzung)

Stylesheet zur Elementselektion mittels leerer Template-Regeln:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=...>

  <xsl:template match="name">
    <xsl:value-of select="nachname"/>
    <xsl:text>, </xsl:text>
    <xsl:value-of select="vorname"/>
  </xsl:template>

  <xsl:template match="geburtstag"/>
  <xsl:template match="beruf"/>

</xsl:stylesheet>
```

Angewandt auf das Beispieldokument:

```
Turing, Alan
Pearl, Judea
```

Vergleiche die Elementselektion durch explizite Verarbeitungssteuerung.

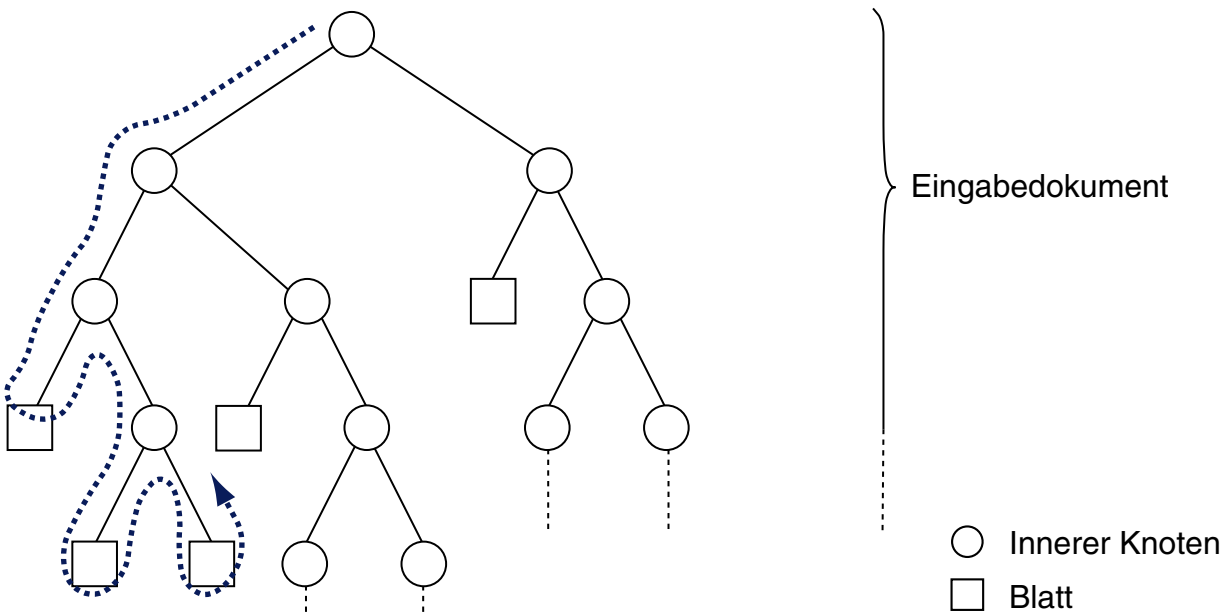
Bemerkungen (Wiederholung):

- Matched eine Template-Regel einen Knoten im XML-Dokument, so gilt der Knoten einschließlich des zugehörigen Teilbaums als abgearbeitet. Mit leeren Template-Regeln kann man Knoten und Teilbäume filtern, die nicht in der Ausgabe erscheinen sollen.
- Matched keine Template-Regel des Stylesheets einen Knoten im XML-Dokument, wird vom XSLT-Prozessor das Built-in-Template zur Ausgabe von Text- und Attributknoten angewandt.

Die XSL-Familie

XSLT-Prozessor: Verarbeitungsstrategie

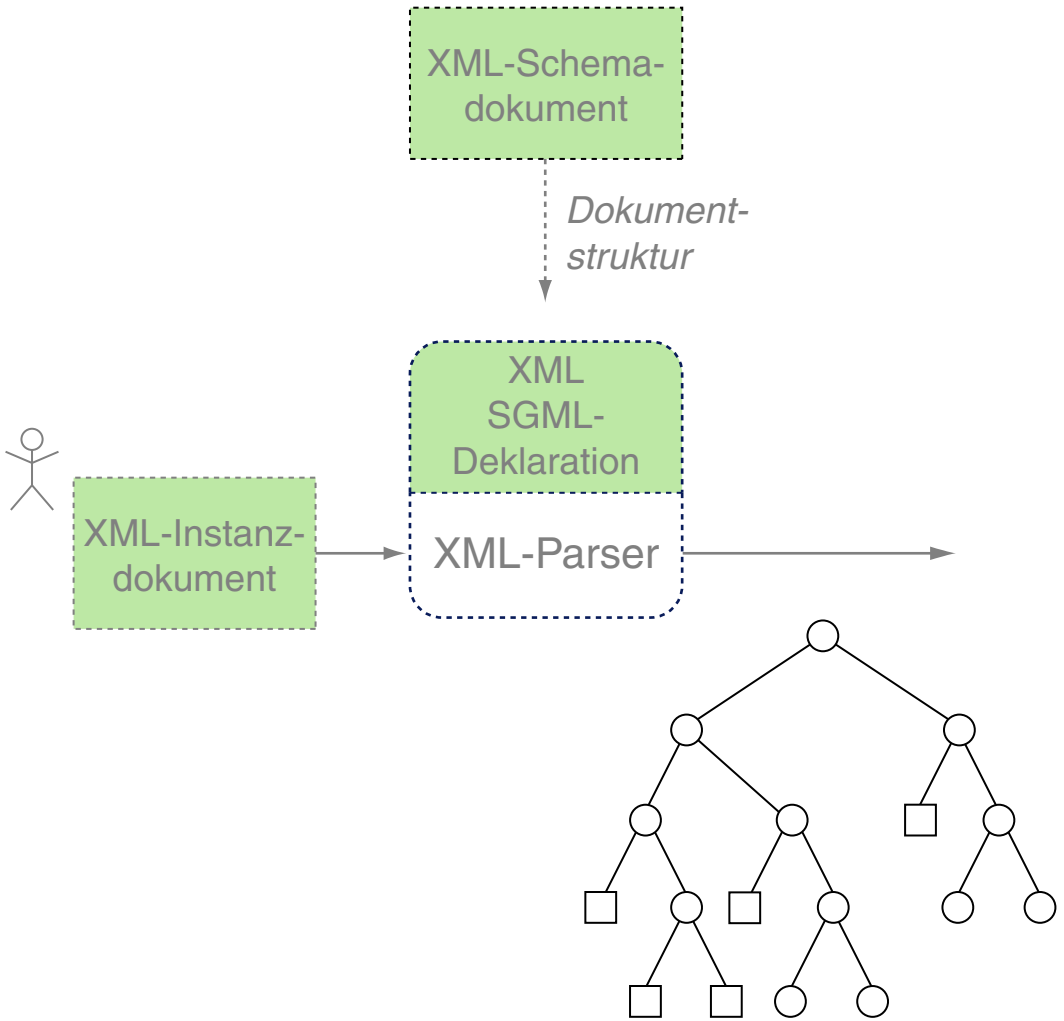
Standardmäßig durchläuft der XSLT-Prozessor den aus dem Eingabedokument erzeugten Baum ausgehend vom Wurzelknoten in Preorder-Reihenfolge.



Während des Traversierungsvorgangs wird für jeden besuchten Knoten das speziellste, matchende Template gesucht und angewandt. So transformiert der XSLT-Prozessor einen XML-Quellbaum in einen XML-Zielbaum.

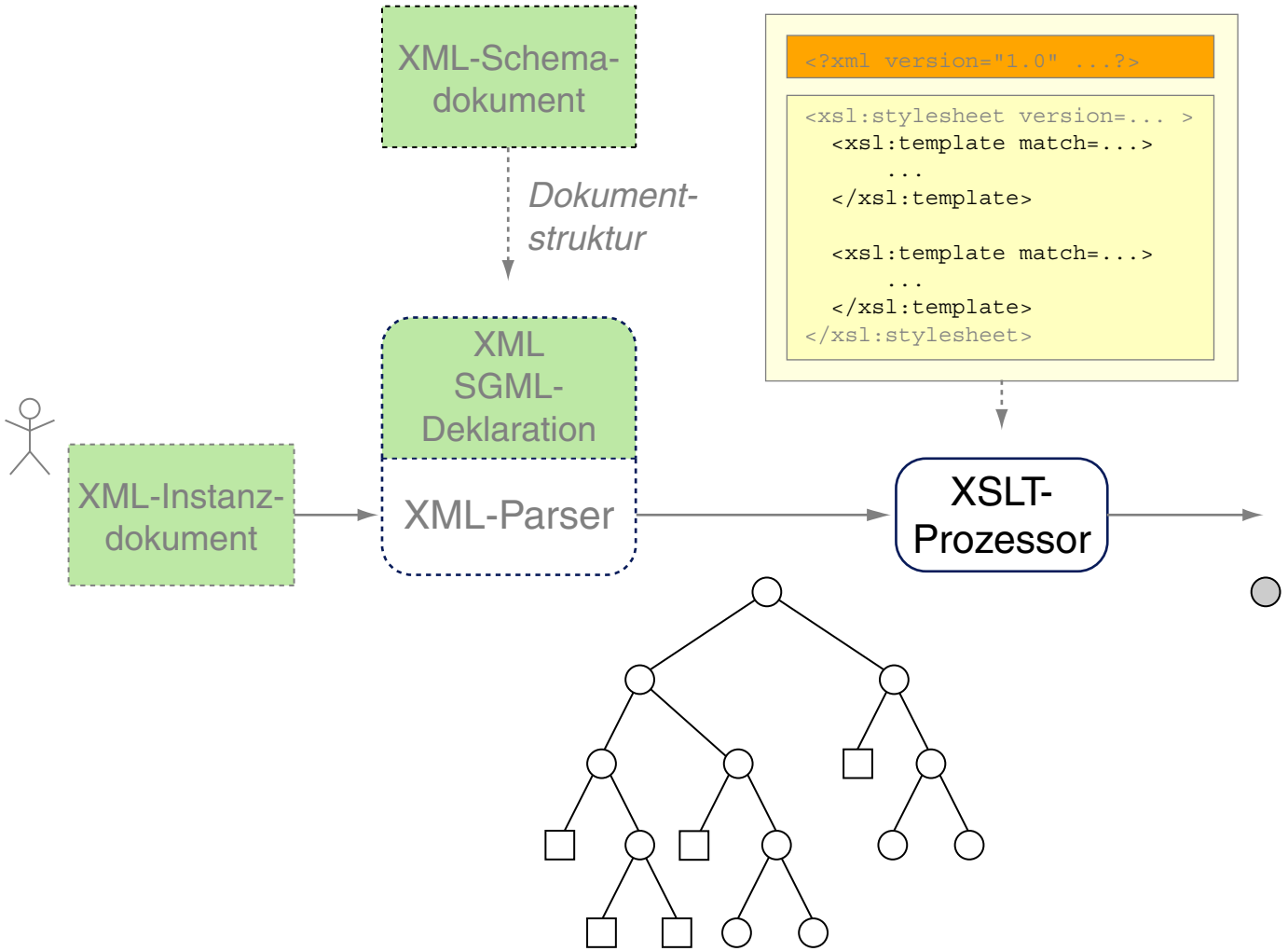
Die XSL-Familie

XSLT-Prozessor: Verarbeitungsstrategie (Fortsetzung)



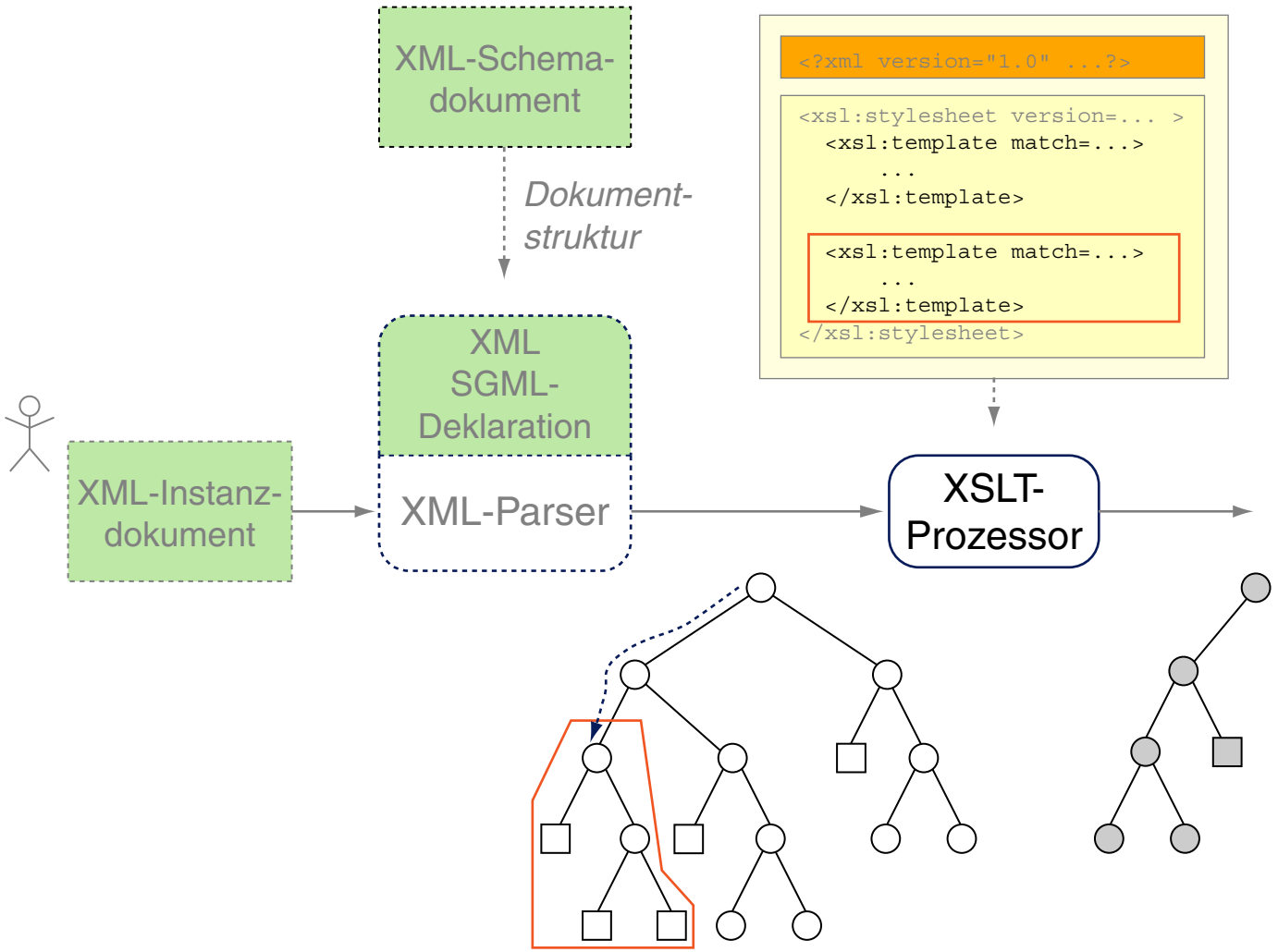
Die XSL-Familie

XSLT-Prozessor: Verarbeitungsstrategie (Fortsetzung)



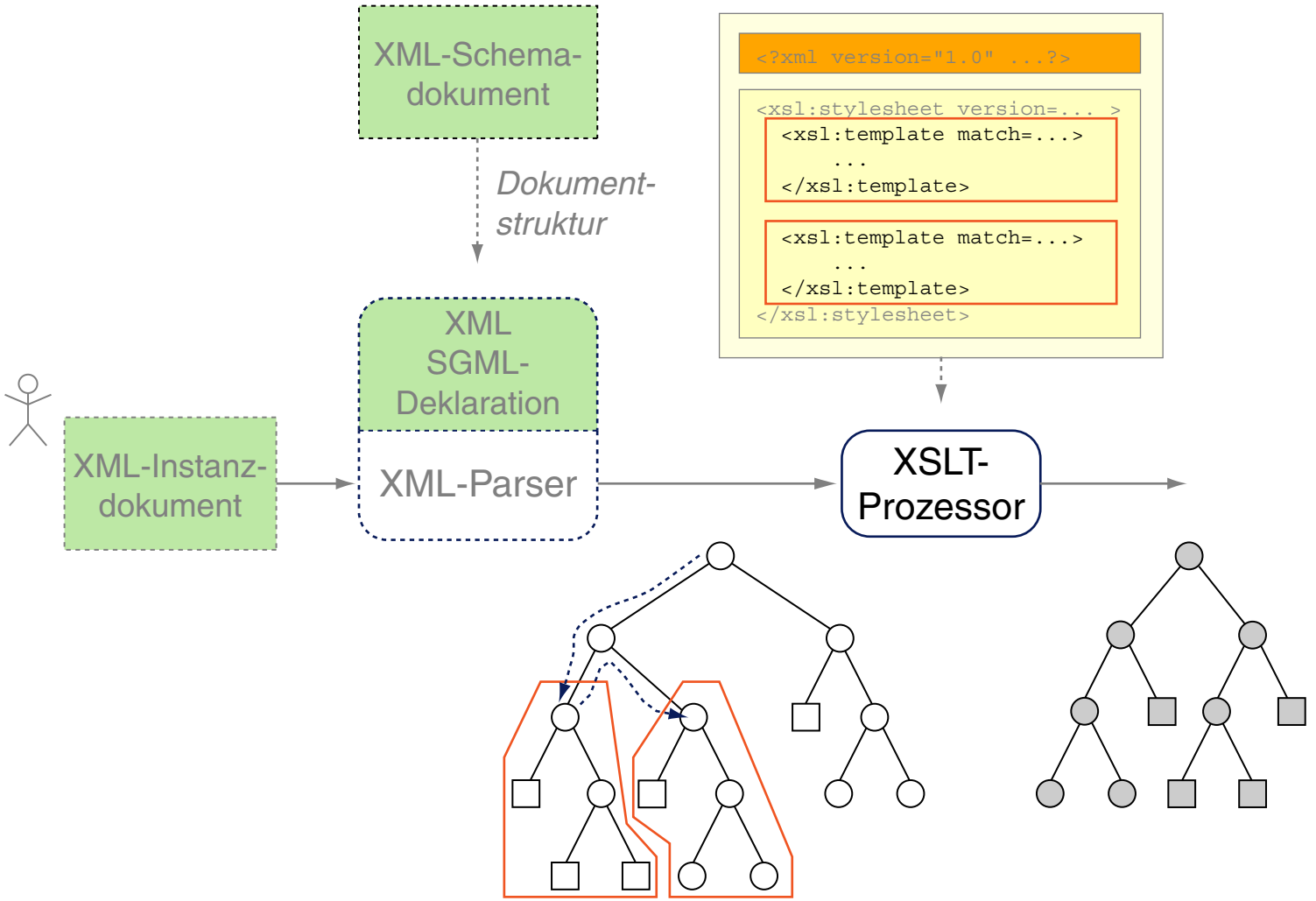
Die XSL-Familie

XSLT-Prozessor: Verarbeitungsstrategie (Fortsetzung)



Die XSL-Familie

XSLT-Prozessor: Verarbeitungsstrategie (Fortsetzung) [CSS-Verarbeitung]



Bemerkungen:

- Aus Verarbeitungssicht spielt die Reihenfolge der Template-Regeln in einem XSL-Stylesheet keine Rolle: die Verarbeitung wird ausschließlich durch die *Reihenfolge der Elemente im Eingabedokument* bestimmt.
- Ein Anwendungskonflikt liegt vor, wenn Lokalisierungspfade von verschiedenen Template-Regeln t_1, t_2 einen Knoten n in ihrer spezifizierten Knotenmengen M_{t_1}, M_{t_2} enthalten. In diesem Fall kommt das Template t_x, $x \in \{1, 2\}$, mit dem speziellsten Pfad im `match`-Attribut zur Anwendung: $|M_{t_x}| \leq \min\{|M_{t_1}|, |M_{t_2}|\}$

Die XSL-Familie

XSLT-Prozessor: Verarbeitungsstrategie (Fortsetzung)

Stylesheet zur Elementselektion mit expliziter Verarbeitungssteuerung:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=...>

  <xsl:template match="person">
    <xsl:apply-templates select="name"/>
  </xsl:template>

  <xsl:template match="name">
    <xsl:value-of select="nachname"/>
    <xsl:text>, </xsl:text>
    <xsl:value-of select="vorname"/>
  </xsl:template>

</xsl:stylesheet>
```

Angewandt auf das Beispieldokument:

Die XSL-Familie

XSLT-Prozessor: Verarbeitungsstrategie (Fortsetzung)

Stylesheet zur Elementselektion mit expliziter Verarbeitungssteuerung:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=...>

  <xsl:template match="person">
    <xsl:apply-templates select="name"/>
  </xsl:template>

  <xsl:template match="name">
    <xsl:value-of select="nachname"/>
    <xsl:text>, </xsl:text>
    <xsl:value-of select="vorname"/>
  </xsl:template>

</xsl:stylesheet>
```

Angewandt auf das Beispieldokument:

```
Turing, Alan
Pearl, Judea
```

Vergleiche die Elementselektion mittels leerer Template-Regeln.

Bemerkungen:

- Das `<xsl:apply-templates>`-Element startet für die mit dem `select`-Attribut spezifizierte Knotenmenge erneut einen Preorder-Durchlauf zur Anwendung der Template-Regeln des Stylesheets.
- Der Wert des `select`-Attributes im `<xsl:apply-templates>`-Element ist ein Lokalisierungspfad in eingeschränkter XPath-Syntax. Weil sich so beliebige Knoten im Dokument spezifizieren lassen, ermöglicht das `<xsl:apply-templates>`-Element die mehrmalige Verarbeitung von Knoten, also auch die Erzeugung von Endlosschleifen.
- Falls keine andere Achse angegeben ist, setzt der Lokalisierungspfad des `<xsl:apply-templates>`-Elements den Pfad des matchenden Knoten fort. Das heißt, die Ausdrücke `select=“./`*Elementname*” und `select=“`*Elementname*” spezifizieren dieselbe Knotenmenge.
- Enthält das `<xsl:apply-templates>`-Element kein `select`-Attribut, so gelten per Default die Kindknoten (`child::`-Achse) des matchenden Knoten als spezifiziert.

Die XSL-Familie

XSLT-Prozessor: Verarbeitungsstrategie (Fortsetzung)

Algorithm: xsl:apply-templates

Input: `select`. XPath expression or empty string.
n_c. Context node.
T. XSL stylesheet with templates.

Output: –

`xsl:apply-templates(select`, n_c, T)

1. `nodes` = *evalXpath*(`select`, n_c)
2. `LOOP`
3. `IF nodes` = $\emptyset$ `THEN RETURN`
4. n = *pop*(`nodes`)
5. t = *mostSpecificTemplate*(T, n)
6. `IF` $t \neq$ `Null`
   `THEN` *executeTemplate*(t, n)
   `ELSE` *executeBuiltInTemplate*(n)
7. `ENDLOOP`

Bemerkungen:

- Die Funktionen *executeTemplate*(t, n) und *executeBuiltInTemplate*(n) wenden das Ersetzungsmuster eines `<xsl:template>`-Elements auf den Knoten n an.
- Der Preorder-Durchlauf entsteht durch den rekursiven Aufruf von `xsl:apply-templates()` in Schritt 6 – entweder durch benutzerdefinierte `<xsl:apply-templates>`-Elemente in t oder durch Anwendung eines Built-in-Templates.
- Der XSLT-Prozessor verwaltet intern das XML Information Set des zu verarbeitenden XML-Dokuments und stellt der Funktion `xsl:apply-templates()` den Kontextknoten n_c und das Stylesheet T zur Verfügung.

Die XSL-Familie

XSLT-Prozessor: Beispiele zur Verarbeitungsstrategie

Stylesheet mit zweifacher Verarbeitung der `<name>`-Kindelemente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=...>

  <xsl:template match="person">
    <xsl:apply-templates select="name"/>
    <xsl:apply-templates select="name"/>
  </xsl:template>

  <xsl:template match="name">
    <xsl:value-of select="nachname"/>
    <xsl:text>, </xsl:text>
    <xsl:value-of select="vorname"/>
  </xsl:template>

</xsl:stylesheet>
```

Angewandt auf das Beispieldokument:

Die XSL-Familie

XSLT-Prozessor: Beispiele zur Verarbeitungsstrategie

Stylesheet mit zweifacher Verarbeitung der `<name>`-Kindelemente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=...>

  <xsl:template match="person">
    <xsl:apply-templates select="name"/>
    <xsl:apply-templates select="name"/>
  </xsl:template>

  <xsl:template match="name">
    <xsl:value-of select="nachname"/>
    <xsl:text>, </xsl:text>
    <xsl:value-of select="vorname"/>
  </xsl:template>

</xsl:stylesheet>
```

Angewandt auf das Beispieldokument:

```
Turing, AlanTuring, Alan
Pearl, JudeaPearl, Judea
```

Die XSL-Familie

XSLT-Prozessor: Beispiele zur Verarbeitungsstrategie (Fortsetzung)

Stylesheet zur wiederholten Verarbeitung aller `<name>`-Elemente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=...>

  <xsl:template match="person">
    <xsl:apply-templates select="//name"/>
  </xsl:template>

  <xsl:template match="name">
    <xsl:value-of select="nachname"/>
    <xsl:text>, </xsl:text>
    <xsl:value-of select="vorname"/>
  </xsl:template>

</xsl:stylesheet>
```

Angewandt auf das Beispieldokument:

Die XSL-Familie

XSLT-Prozessor: Beispiele zur Verarbeitungsstrategie (Fortsetzung)

Stylesheet zur wiederholten Verarbeitung aller `<name>`-Elemente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=...>

  <xsl:template match="person">
    <xsl:apply-templates select="//name"/>
  </xsl:template>

  <xsl:template match="name">
    <xsl:value-of select="nachname"/>
    <xsl:text>, </xsl:text>
    <xsl:value-of select="vorname"/>
  </xsl:template>

</xsl:stylesheet>
```

Angewandt auf das Beispieldokument:

```
  Turing, AlanPearl, Judea
  Turing, AlanPearl, Judea
```

Die XSL-Familie

XSLT-Prozessor: Beispiele zur Verarbeitungsstrategie (Fortsetzung)

Stylesheet, dessen matchende Template-Regel die leere Knotenmenge liefert:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=...>

  <xsl:template match="person">
    <xsl:apply-templates select="nachname"/>
  </xsl:template>

  <xsl:template match="name">
    <xsl:value-of select="nachname"/>
    <xsl:text>, </xsl:text>
    <xsl:value-of select="vorname"/>
  </xsl:template>

</xsl:stylesheet>
```

Angewandt auf das Beispieldokument:

Die XSL-Familie

XSLT-Prozessor: Beispiele zur Verarbeitungsstrategie (Fortsetzung)

Stylesheet, dessen Verarbeitung in eine Endlosschleife führt:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=...>

  <xsl:template match="person">
    <xsl:apply-templates select="/personen/person"/>
  </xsl:template>

  <xsl:template match="name">
    <xsl:value-of select="nachname"/>
    <xsl:text>, </xsl:text>
    <xsl:value-of select="vorname"/>
  </xsl:template>

</xsl:stylesheet>
```

Angewandt auf das Beispieldokument:

Die XSL-Familie

XSLT-Prozessor: Beispiele zur Verarbeitungsstrategie (Fortsetzung)

Stylesheet, dessen Verarbeitung in eine Endlosschleife führt:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=...>

  <xsl:template match="person">
    <xsl:apply-templates select="/personen/person"/>
  </xsl:template>

  <xsl:template match="name">
    <xsl:value-of select="nachname"/>
    <xsl:text>, </xsl:text>
    <xsl:value-of select="vorname"/>
  </xsl:template>

</xsl:stylesheet>
```

Angewandt auf das Beispieldokument:

```
(Location of error unknown)XSLT Error (java.lang.StackOverflowError):
null
```

Die XSL-Familie

XSLT-Prozessor: Built-in-Templates

1. Built-in-Template, das die rekursive Verarbeitung garantiert, falls kein matchendes Template im Stylesheet existiert:

```
<xsl:template match="*|/">
  <xsl:apply-templates/>
</xsl:template>
```

2. Built-in-Template zur Ausgabe von Text- und Attributknoten:

```
<xsl:template match="text()|@*">
  <xsl:value-of select="."/>
</xsl:template>
```

3. Built-in-Template, das die Kommentare matched und ignoriert:

```
<xsl:template match="processing-instruction()|comment()"/>
```

   Vergleiche die Elementselektion mittels leerer Template-Regeln.

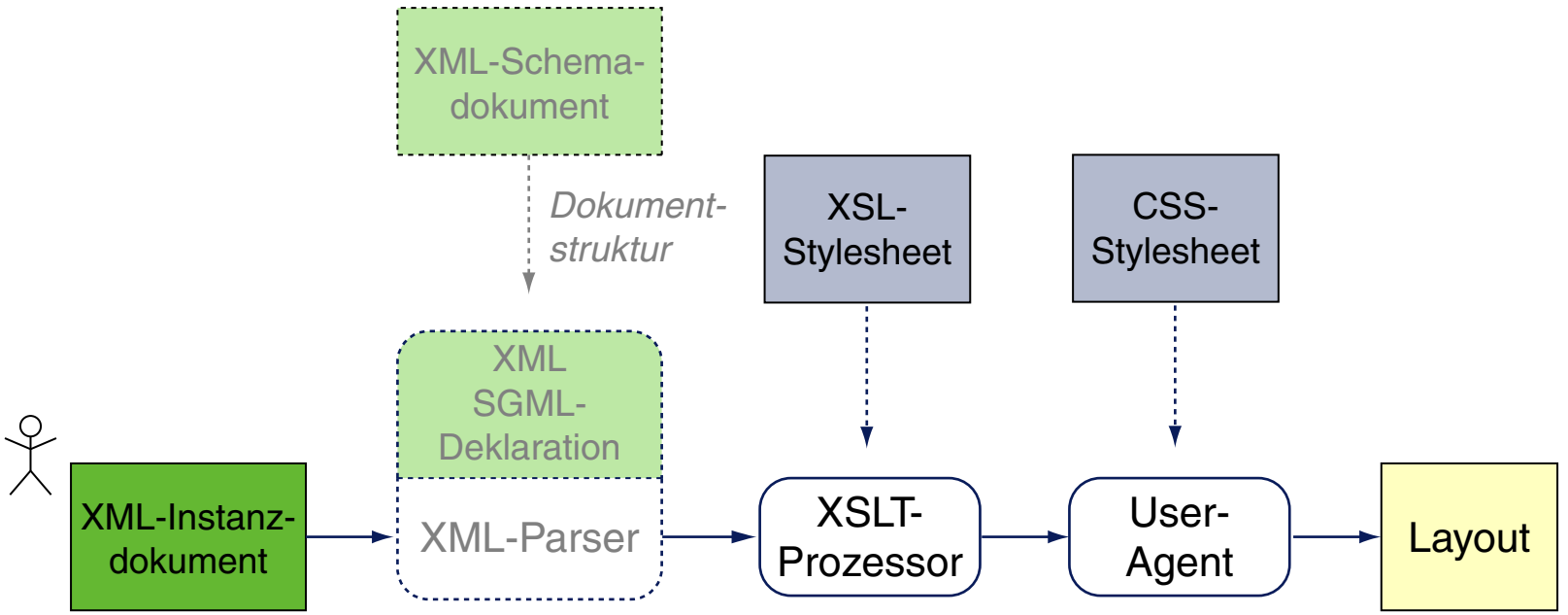
Die XSL-Familie

Weitere XSLT-Konzepte

- Template-Modi zur Charakterisierung von Verarbeitungsphasen
- benannte Templates zur Realisierung direkter Aufrufe
- Nummerierung und Sortierung von Ausgabeelementen
- bedingte Verarbeitung und Schleifen
- Import anderer Stylesheets

Die XSL-Familie

XML-Dokumentenverarbeitung: Erzeugung von HTML-Dokumenten



Vergleiche hierzu den Standardprozess der XSL Transformation.

Die XSL-Familie

Erzeugung von HTML-Dokumenten: Zusammenspiel mit CSS

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Die XSL-Familie

Erzeugung von HTML-Dokumenten: Zusammenspiel mit CSS (Fortsetzung)

```
<xsl:template match="personen">
  <html>
    <head>
      <title>
        <xsl:text>Personen</xsl:text>
      </title>
    </head>
    <body>
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>
```

Die XSL-Familie

Erzeugung von HTML-Dokumenten: Zusammenspiel mit CSS (Fortsetzung)

```
<xsl:template match="personen">
  <html>
    <head>
      <title>
        <xsl:text>Personen</xsl:text>
      </title>
    </head>
    <body>
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>
```

Die XSL-Familie

Erzeugung von HTML-Dokumenten: Zusammenspiel mit CSS (Fortsetzung)

```
<xsl:template match="personen">
  <html>
    <head>
      <title>
        <xsl:text>Personen</xsl:text>
      </title>
    </head>
    <body>
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>
```

Die XSL-Familie

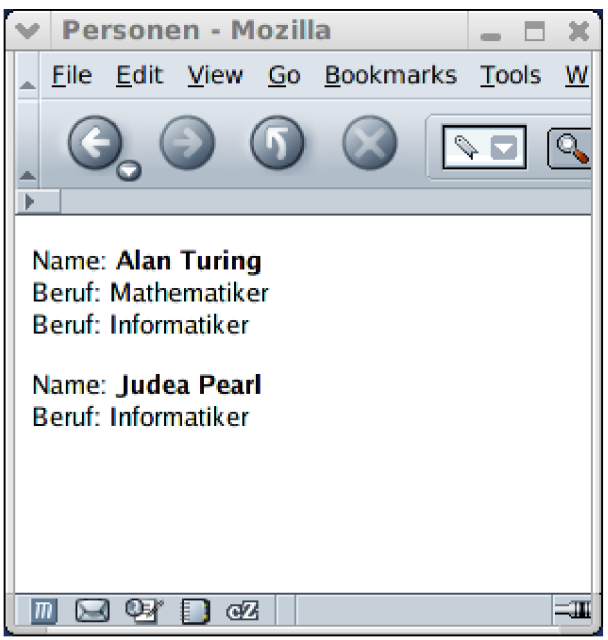
Erzeugung von HTML-Dokumenten: Zusammenspiel mit CSS (Fortsetzung)

```
<xsl:template match="personen">
  <html>
    <head>
      <title>
        <xsl:text>Personen</xsl:text>
      </title>
    </head>
    <body>
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>

<xsl:template match="name">
  <p/>
  <div>
    <xsl:text>Name: </xsl:text>
    <span style="font-weight:bold">
      <xsl:value-of select="self::*"/>
    </span>
  </div>
</xsl:template>
...
```

Die XSL-Familie

Erzeugung von HTML-Dokumenten: Zusammenspiel mit CSS (Fortsetzung)

```
<xsl:template match="personen">
  <html>
    <head>
      <title>
        <xsl:text>Personen</xsl:text>
      </title>
    </head>
    <body>
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>

<xsl:template match="name">
  <p/>
  <div>
    <xsl:text>Name: </xsl:text>
    <span style="font-weight:bold">
      <xsl:value-of select="self::*"/>
    </span>
  </div>
</xsl:template>
...
```



Die XSL-Familie

Erzeugung von HTML-Dokumenten: Zusammenspiel mit CSS (Fortsetzung)

```
<xsl:template match="personen">
  <html>
    <head>
      <title>
        <xsl:text>Personen</xsl:text>
      </title>
      <link rel="stylesheet" type="text/css" href="personen.css"/>
    </head>
    <body>
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>

<xsl:template match="name">
  <p/>
  <div>
    <xsl:text>Name: </xsl:text>
      <xsl:value-of select="self::*"/>
  </div>
</xsl:template>
...
```

Bemerkungen:

- Eine Anwendung nach diesem Prinzip sind die FAQs des W3C:
  Aus der XML-Source faq.xml gemäß der DTD faq.dtd wird mittels des Stylesheets faqxsl.xsl das HTML-Dokument faq.html erzeugt. Weil in faq.xml das Stylesheet faq.css verlinkt ist, zeigt der Browser nicht den XML-Dokumentenbaum an.

Die XSL-Familie

Erzeugung von HTML-Dokumenten: Datenaufbereitung

CD-Datenbank als XML-Beispieldokument [w3schools] :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>

  ...

  <cd>
    <title>Greatest Hits</title>
    <artist>Dolly Parton</artist>
    <company>RCA</company>
    <price>9.90</price>
    <year>1982</year>
  </cd>
</catalog>
```

Die XSL-Familie

Erzeugung von HTML-Dokumenten: Datenaufbereitung (Fortsetzung)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" ...>

<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="catalog/cd">
      <tr>
        <td><xsl:value-of select="title"/></td>
        <td><xsl:value-of select="artist"/></td>
      </tr>
      </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Die XSL-Familie

Erzeugung von HTML-Dokumenten: Datenaufbereitung (Fortsetzung)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" ...>

<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="catalog/cd">
      <tr>
        <td><xsl:value-of select="title"/></t
        <td><xsl:value-of select="artist"/></
      </tr>
      </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

[w3schools xml, xsl]

Die XSL-Familie

Erzeugung von HTML-Dokumenten: Datenaufbereitung (Fortsetzung)

Filtern mit XPath:

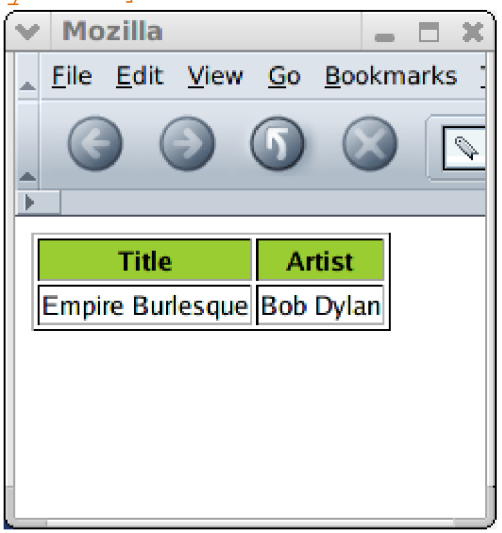
```
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="catalog/cd[artist='Bob Dylan']">
      <tr>
        <td><xsl:value-of select="title"/></td>
        <td><xsl:value-of select="artist"/></td>
      </tr>
      </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>
```

Die XSL-Familie

Erzeugung von HTML-Dokumenten: Datenaufbereitung (Fortsetzung)

Filtern mit XPath:

```
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="catalog/cd[artist='Bob Dylan']">
      <tr>
        <td><xsl:value-of select="title"/></td>
        <td><xsl:value-of select="artist"/></td>
      </tr>
      </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>
```



[w3schools xml, xsl]

Die XSL-Familie

Erzeugung von HTML-Dokumenten: Datenaufbereitung (Fortsetzung)

Sortieren:

```
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="catalog/cd">
      <xsl:sort select="artist"/>
      <tr>
        <td><xsl:value-of select="title"/></td>
        <td><xsl:value-of select="artist"/></td>
      </tr>
      </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>
```

Die XSL-Familie

Erzeugung von HTML-Dokumenten: Datenaufbereitung (Fortsetzung)

Sortieren:

```
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="catalog/cd">
      <xsl:sort select="artist"/>
      <tr>
        <td><xsl:value-of select="title"/></td>
        <td><xsl:value-of select="artist"/></td>
      </tr>
      </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>
```

Die XSL-Familie

Erzeugung von HTML-Dokumenten: Datenaufbereitung (Fortsetzung)

Verwenden von Bedingungen:

```
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="catalog/cd">
      <xsl:if test="price &gt; 10">
        <tr>
          <td><xsl:value-of select="title"/></td>
          <td><xsl:value-of select="artist"/></td>
        </tr>
      </xsl:if>
      </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>
```

Die XSL-Familie

Erzeugung von HTML-Dokumenten: Datenaufbereitung (Fortsetzung)

Verwenden von Bedingungen:

```
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="catalog/cd">
      <xsl:if test="price &gt; 10">
        <tr>
          <td><xsl:value-of select="title"/></td>
          <td><xsl:value-of select="artist"/></td
        </tr>
      </xsl:if>
      </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>
```

Die XSL-Familie

Erzeugung von HTML-Dokumenten: Datenaufbereitung (Fortsetzung)

Verwenden von Bedingungen:

```
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      ...
      <xsl:for-each select="catalog/cd">
      <tr>
        <td><xsl:value-of select="title"/></td>
        <xsl:choose>
          <xsl:when test="price &gt; 10">
            <td bgcolor="#ff00ff"><xsl:value-of select="artist"/></td>
          </xsl:when>
          <xsl:otherwise>
            <td><xsl:value-of select="artist"/></td>
          </xsl:otherwise>
        </xsl:choose>
      </tr>
      </xsl:for-each>
    </table>
  ...
```

Die XSL-Familie

Erzeugung von HTML-Dokumenten: Datenaufbereitung (Fortsetzung)

Verwenden von Bedingungen:

```
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      ...
      <xsl:for-each select="catalog/cd">
      <tr>
        <td><xsl:value-of select="title"/></td>
        <xsl:choose>
          <xsl:when test="price &gt; 10">
            <td bgcolor="#ff00ff"><xsl:value-of
          </xsl:when>
          <xsl:otherwise>
            <td><xsl:value-of select="artist"/>
          </xsl:otherwise>
        </xsl:choose>
      </tr>
      </xsl:for-each>
    </table>
  ...
```

Die XSL-Familie

XML-Dokumentenverarbeitung: Elementinhalte anpassen [DOM-API]

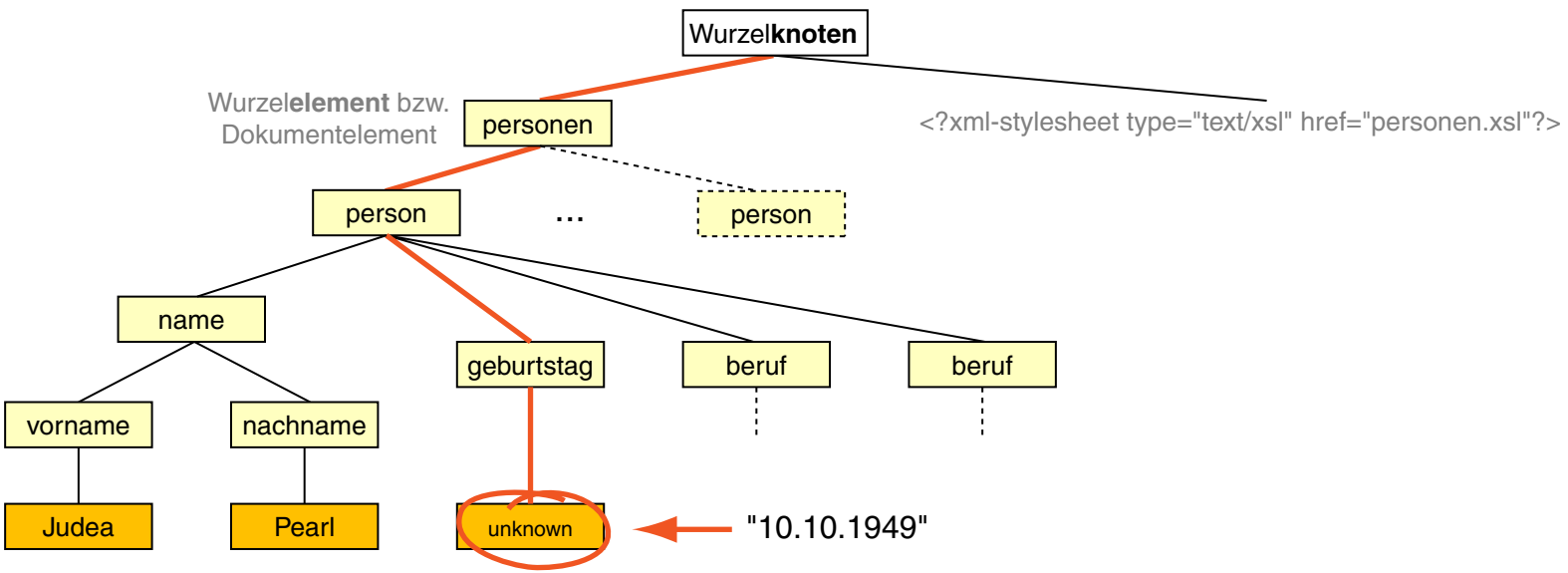
```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

Die XSL-Familie

XML-Dokumentenverarbeitung: Elementinhalte anpassen (Fortsetzung)

Aufgabe:

1. Die Person „Judea Pearl“ finden.
2. Seinen Geburtstag auf einen bestimmten Wert setzen.



Die XSL-Familie

XML-Dokumentenverarbeitung: Elementinhalte anpassen (Fortsetzung)

Stylesheet:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=...>

  <xsl:template match="@*|node()"> [W3C]
    <xsl:copy><xsl:apply-templates select="@*|node()|"/></xsl:copy>
  </xsl:template>









</xsl:stylesheet>
```

Die XSL-Familie

XML-Dokumentenverarbeitung: Elementinhalte anpassen (Fortsetzung)

Stylesheet:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=...>

  <xsl:template match="@*|node()"> [W3C]
    <xsl:copy><xsl:apply-templates select="@*|node()|"/></xsl:copy>
  </xsl:template>

  <xsl:template match= "person[name/nachname='Pearl' and
       name/vorname='Judea']/geburtstag/text()">
    <xsl:text>10.10.1949</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Die XSL-Familie

XML-Dokumentenverarbeitung: Elementinhalte anpassen (Fortsetzung)

Stylesheet:

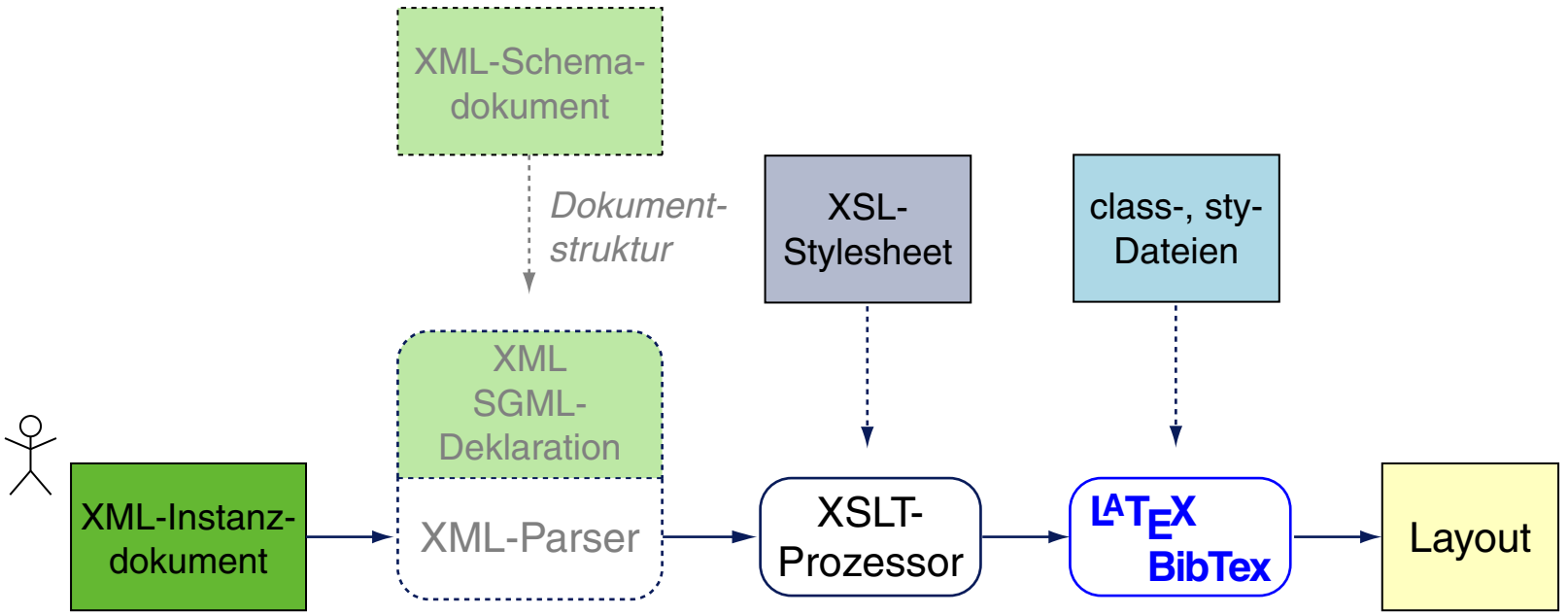
```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=...>
  <xsl:template match="@*|node()"> [W3C]
    <xsl:copy><xsl:apply-templates select="@*|node()|"/></xsl:copy>
  </xsl:template>

  <xsl:template match= "person[name/nachname='Pearl' and
       name/vorname='Judea']/geburtstag/text()">
    <xsl:text>10.10.1949</xsl:text>
  </xsl:template>
</xsl:stylesheet>
```

Angewandt auf das Beispieldokument:

```
...
  <name>
    <vorname>Judea</vorname>
    <nachname>Pearl</nachname>
  </name>
  <geburtstag>10.10.1949</geburtstag>
...
```

Die XSL-Familie

XML-Dokumentenverarbeitung: Prozesskette für Printmedien



Vergleiche hierzu

- den Standardprozess der XSL Transformation
- und die HTML-Prozesskette.

Die XSL-Familie

Prozesskette für Printmedien: Erzeugung von Latex-Dokumenten

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" href="xml2latex.xsl"?>

<book>

<section>
  <title>Eine Überschrift</title>

  Hier ist der Fließtext ...
</section>

</book>
```

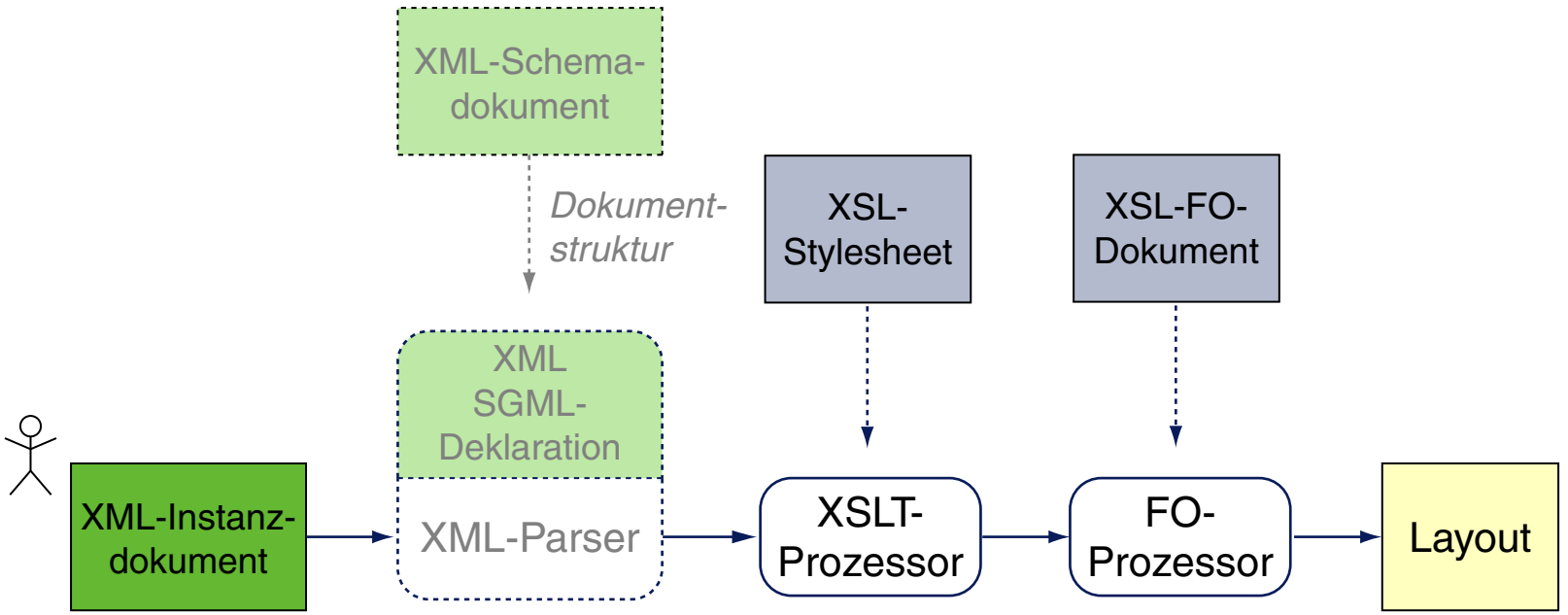
Die XSL-Familie

Prozesskette für Printmedien: Erzeugung von Latex-Dokumenten (Fortsetzung)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=...>

<xsl:template match="/">
  \documentclass{article}
  \usepackage[T1]{fontenc}
  \usepackage[english,german]{babel}
  \begin{document}
  <xsl:apply-templates/>
  \end{document}
</xsl:template>

<xsl:template match="section">
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="title">
  \section{<xsl:value-of select="self::*"/>}
</xsl:template>

...

</xsl:stylesheet>
```

Die XSL-Familie

XML-Dokumentenverarbeitung: Erzeugung beliebiger Formate mit XSL-FO



Vergleiche hierzu

- den Standardprozess der XSL Transformation,
- die HMTL-Prozesskette
- und die Latex-Prozesskette.

Die XSL-Familie

Quellen zum Nachlernen und Nachschlagen im Web: Referenz

- M. Kay. *XSL Transformations (XSLT) Version 2.0.*
  W3C Recommendation. www.w3.org/TR/xslt20
- A. Berglund et al. *XML Path Language (XPath) 2.0.*
  W3C Recommendation. www.w3.org/TR/xpath20
- W3C Glossar.
  www.w3.org/2003/glossary

Die XSL-Familie

Quellen zum Nachlernen und Nachschlagen im Web: Usage

- M. Nic, J. Jirat. *XPath Tutorial.*
  www.zvon.org
- Cover Pages.
  xml.coverpages.org/xsl.html
- W3 Schools.
  www.w3schools.com/xsl
- Apache. XSLT-Prozessor Xalan.
  xalan.apache.org
- Saxonica.com. XSLT and XQuery Processing.
  www.saxonica.com

Kapitel WT:III (Fortsetzung)

APIs für XML-Dokumente

XML-Dokumente sind uns bisher in serialisierter Form, z.B. als Inhalte von Dateien, begegnet. Ihre Manipulation durch Programme erfordert:

1. eine geeignete Repräsentation im Hauptspeicher eines Rechners
2. Methoden zum Zugriff und zur Manipulation ➜ API

> *"An application programming interface (API) is a set of routines, data structures, object classes, and/or protocols provided by libraries and/or operating system services in order to support the building of applications."* [Wikipedia, June 2009]

APIs für XML-Dokumente

XML-Dokumente sind uns bisher in serialisierter Form, z.B. als Inhalte von Dateien, begegnet. Ihre Manipulation durch Programme erfordert:

1. eine geeignete Repräsentation im Hauptspeicher eines Rechners
2. Methoden zum Zugriff und zur Manipulation ➜ API

> *"An application programming interface (API) is a set of routines, data structures, object classes, and/or protocols provided by libraries and/or operating system services in order to support the building of applications."* [Wikipedia, June 2009]

API-Technologien zum Zugriff und zur Manipulation von XML-Dokumenten:

- DOM, Document Object Model
- SAX, Simple API for XML
- StAX, Streaming API for XML [XML.com]
- XPP, Common API for XML Pull Parsing
- XML Data Binding

APIs für XML-Dokumente

DOM: Historie

> *"The Document Object Model (DOM) is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure, and style of documents."* [W3C: DOM, FAQ]

1998 DOM Level 1. Funktionen zum Zugriff auf XML 1.0- und HTML-Dokumente. [W3C]

2000 DOM Level 2. Level 1 plus Namensräume, Style Sheets, Events, Views, etc. [W3C: Core, Events, HTML, Style, Traversal, Views]

2003 Java JDK 1.4 mit DOM-Implementierung. [Javadoc]

2004 DOM Level 3. Level 2 plus XPath, Methoden zum Laden und Speichern, Validierung, etc. [W3C: Core, Events, Load & Save, Schemas Validation, Views, XPath,]

2012 DOM4. Working Draft. [W3C]

Bemerkungen:

- DOM entstand aus der Bestrebung, Java- und JavaScript-Programme zwischen Browsern austauschbar zu machen. Vorangegangen waren herstellerspezifische Ideen und Implementierungen für das sogenannte „Dynamic HTML“. [W3C]
- Level 1 und Level 2 von DOM enthalten noch keine Spezifikation dafür, wie ein Dokument in eine DOM-Struktur geladen oder aus ihr heraus gespeichert werden kann: sie setzen das Vorhandensein der Dokumente in einer Browser-Umgebung voraus.
- www.quirksmode.org zeigt eine sehr gute Übersicht mit W3C DOM Compatibility Tables für die verbreiteten Browser.
- W3C-Service aus dem Jahr 2003, der den aufrufenden Browser *(User agent)* dahingehend analysiert, welche DOM-Module unterstützt werden. Der Service implementiert eine Abfrage der JavaScript-Version des Browsers: DOM-Module sind an den Möglichkeiten des JavaScript-Zugriffs erkennbar.

APIs für XML-Dokumente

DOM: Konzepte

- DOM modelliert ein Dokument gemäß seiner Struktur als eine Hierarchie von Knoten. [W3C]
- Für die Knoten definiert DOM keine Datentypen, sondern Objekte. Die Spezifikation beschreibt Interfaces mit den für die Knotenobjekte erlaubten Operationen. [W3C]
- Die Semantik der Knotenobjekte orientiert sich am XML Information Set.

APIs für XML-Dokumente

DOM: Konzepte

- DOM modelliert ein Dokument gemäß seiner Struktur als eine Hierarchie von Knoten. [W3C]
- Für die Knoten definiert DOM keine Datentypen, sondern Objekte. Die Spezifikation beschreibt Interfaces mit den für die Knotenobjekte erlaubten Operationen. [W3C]
- Die Semantik der Knotenobjekte orientiert sich am XML Information Set.

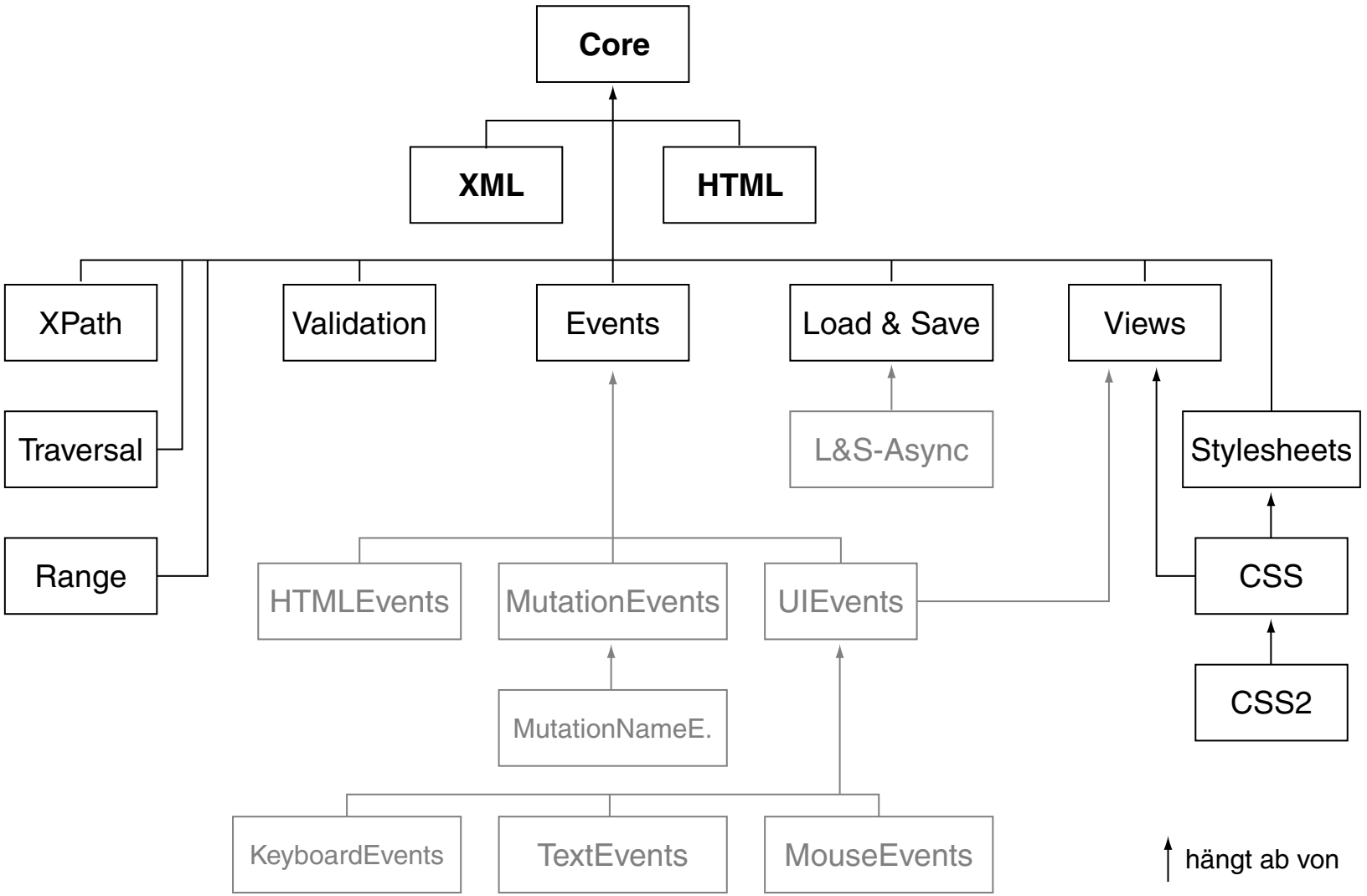
- Die DOM-Spezifikation ist neutral in Bezug auf Betriebssysteme und Programmiersprachen: die Interfaces sind in der *Interface Description Language* OMG IDL der Object Management Group, OMG, verfasst.
- Die sprachspezifische Umsetzung von DOM erfolgt durch sogenannte Language Bindings. [W3C: Java, ECMAScript]
- Das W3C stellt Language Bindings für Java und ECMAScript zur Verfügung.

Bemerkungen:

- Oft wird mit dem Begriff „DOM“ auch die Datenstruktur zur Repräsentation eines Dokuments bezeichnet – und nicht die Programmierschnittstelle (API) zum Zugriff auf die Datenstruktur. Diese Sicht motiviert sich aus dem Verständnis der objektorientierten Programmierung:
  “The name *Document Object Model* was chosen because it is an *object model* in the traditional object oriented design sense: documents are modeled using objects, and the model encompasses not only the structure of a document, but also the behavior of a document and the objects of which it is composed.” [W3C]
- “The Document Object Model does not define what information in a document is relevant or how information in a document is structured. For XML, this is specified by the XML Information Set. The DOM is simply an API to this information set.” [W3C]
- Wiederholung. Das W3C hat mittlerweile drei Datenmodelle für XML-Dokumente definiert: XML Information Set, XPath, Document Object Model (DOM). Das XPath-Datenmodell basiert auf einer Baumstruktur, die bei der Abfrage eines XML-Dokuments durchlaufen wird und ist dem XML Information Set ähnlich; DOM ist der Vorläufer beider Datenmodelle. DOM und das XPath-Datenmodell können als Interpretationen des XML Information Sets betrachtet werden. [MSDN]

APIs für XML-Dokumente

DOM: Struktur der API [W3C]



APIs für XML-Dokumente

DOM: Struktur der API [W3C] (Fortsetzung)

DOM-Modul	Beschreibung
Core	definiert die grundlegenden Interfaces für den Zugriff und die Manipulation von Objekten in strukturierten Dokumenten
HTML	Interfaces für Programme und Scriptsprachen, um Struktur und Inhalt von HTML 4.01- und XHTML 1.0-Dokumenten zu manipulieren
XML	Spezielle XML-Interfaces, die über das Core- und HTML-Modul hinausgehen

APIs für XML-Dokumente

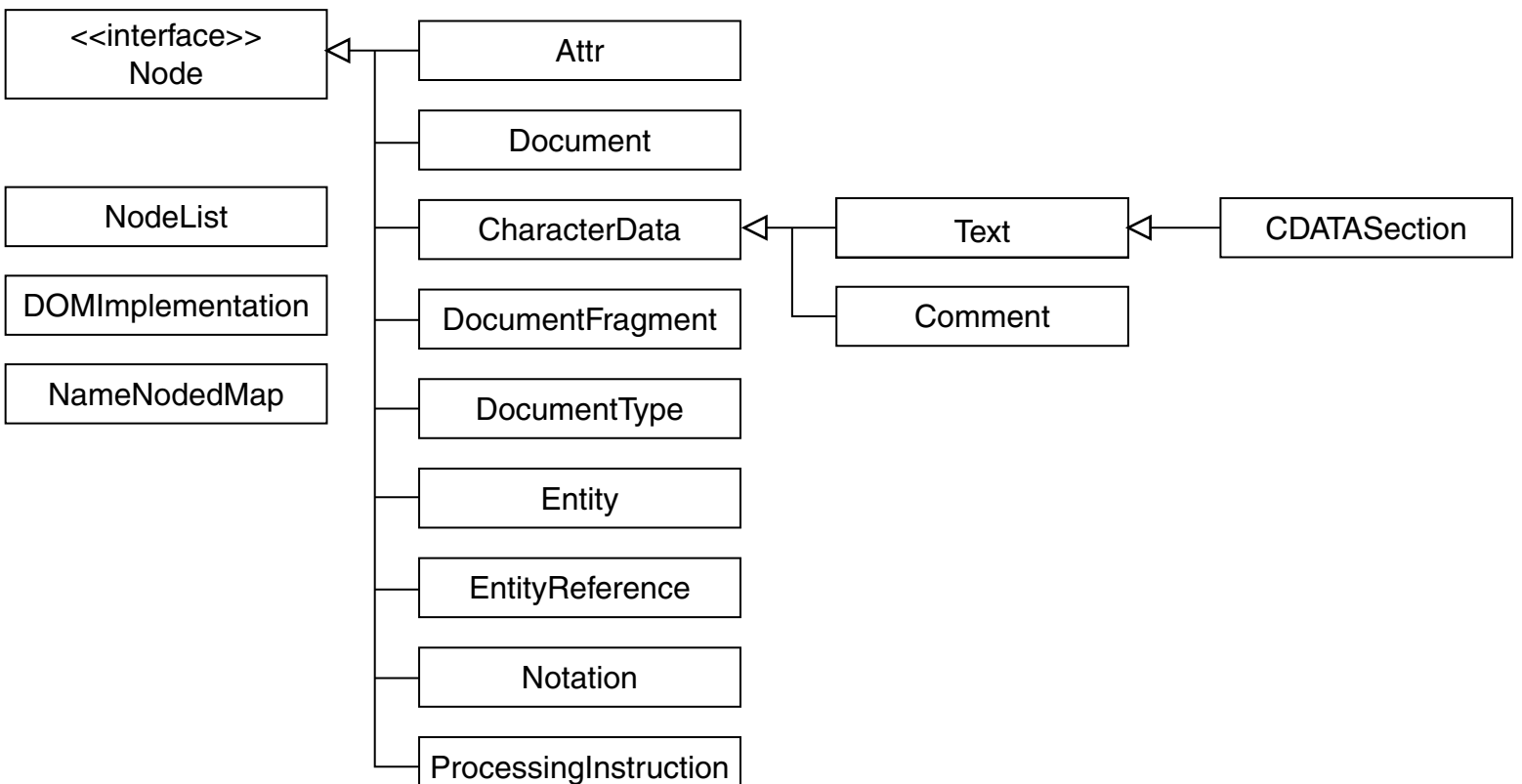
DOM: Struktur der API [W3C] (Fortsetzung)

DOM-Modul	Beschreibung
Core	definiert die grundlegenden Interfaces für den Zugriff und die Manipulation von Objekten in strukturierten Dokumenten
HTML	Interfaces für Programme und Scriptsprachen, um Struktur und Inhalt von HTML 4.01- und XHTML 1.0-Dokumenten zu manipulieren
XML	Spezielle XML-Interfaces, die über das Core- und HTML-Modul hinausgehen
XPath	Abbildung der DOM Sicht des XML Information Set auf XPath
Traversal	Interfaces zur Traversierung von Dokumentbäumen (u.a. TreeWalker)
Range	Interfaces zum selektierenden Dokumentzugriff (u.a. durch Maus)
Validation	Interfaces zum schemakonformen Ändern von Dokumenten
Events	Interfaces zur Behandlung von Ereignissen
Load & Save	plattform- und sprachunabhängige Interfaces zur Dokumentserialisierung
Views	Interfaces zur Realisierung verschiedener Sichten auf dasselbe Dokument.
StyleSheets	Interfaces zum Zugriff auf Informationen eines Stylesheets.
CSS, CSS2	Interfaces zum Zugriff auf Informationen einer CSS- bzw. CSS2-Beschreibung.

APIs für XML-Dokumente

DOM: Interfaces

Interface-Hierarchie der Knotentypen im DOM-Core-Modul:



IDL-Definitionen: [W3C: Node < CharacterData < Text < CDATASection]

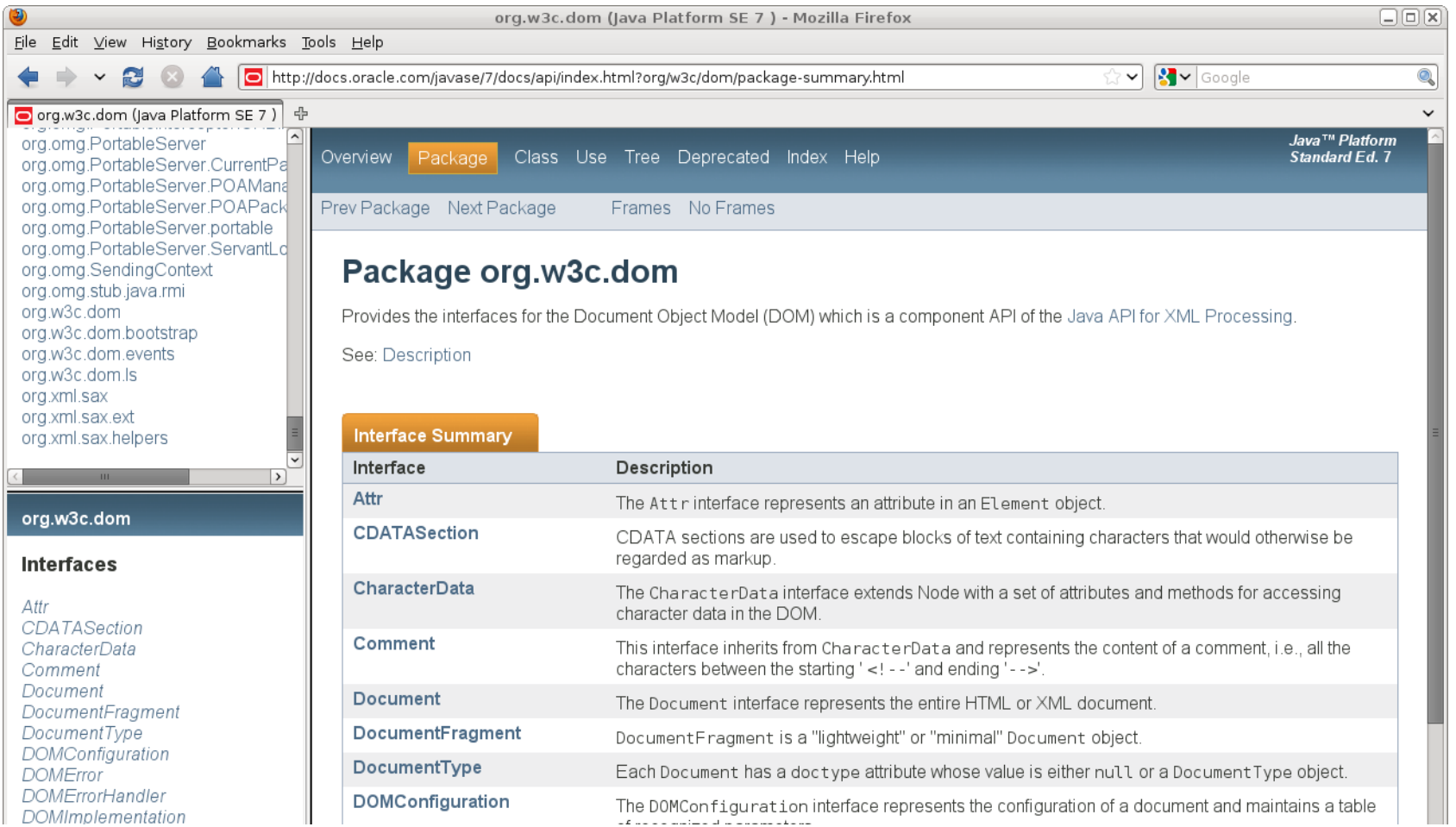
Bemerkungen:

- Die Interfaces der meisten DOM-Objekte sind von dem generischen `node`-Interface abgeleitet. Das `node`-Interface behandelt die gemeinsamen Anteile der verschiedenen Knoten eines XML-Baums.
- Über die Interfaces des DOM-HTML-Moduls greifen Scriptsprachen wie JavaScript oder JScript, Browser-Plug-Ins, ActiveX-Controls oder Java Applets auf HTML-Dokumente im Browser zu.

APIs für XML-Dokumente

DOM: Interfaces (Fortsetzung)

Java Language Binding. `org.w3c.dom`-Package [Javadoc, W3C] :



APIs für XML-Dokumente

DOM: Interfaces (Fortsetzung)

Java Language Binding. Methoden des `node`-Interface [Javadoc] :

Method Summary

Methods

Modifier and Type	Method and Description
`Node`	`appendChild(Node newChild)` Adds the node `newChild` to the end of the list of children of this node.
`Node`	`cloneNode(boolean deep)` Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes.
`short`	`compareDocumentPosition(Node other)` Compares the reference node, i.e.
`NamedNodeMap`	`getAttributes()` A `NamedNodeMap` containing the attributes of this node (if it is an `Element`) or `null` otherwise.
`String`	`getBaseURI()` The absolute base URI of this node or `null` if the implementation wasn't able to obtain an absolute URI.
`NodeList`	`getChildNodes()` A `NodeList` that contains all children of this node.
`Object`	`getFeature(String feature, String version)` This method returns a specialized object which implements the specialized APIs of the specified feature and version, as specified in .
`Node`	`getFirstChild()` The first child of this node.
`Node`	`getLastChild()` The last child of this node.
`String`	`getLocalName()` Returns the local part of the qualified name of this node.
`String`	`getNamespaceURI()` The namespace URI of this node, or `null` if it is unspecified (see).
`Node`	`getNextSibling()` The node immediately following this node.

APIs für XML-Dokumente

DOM: Anwendung

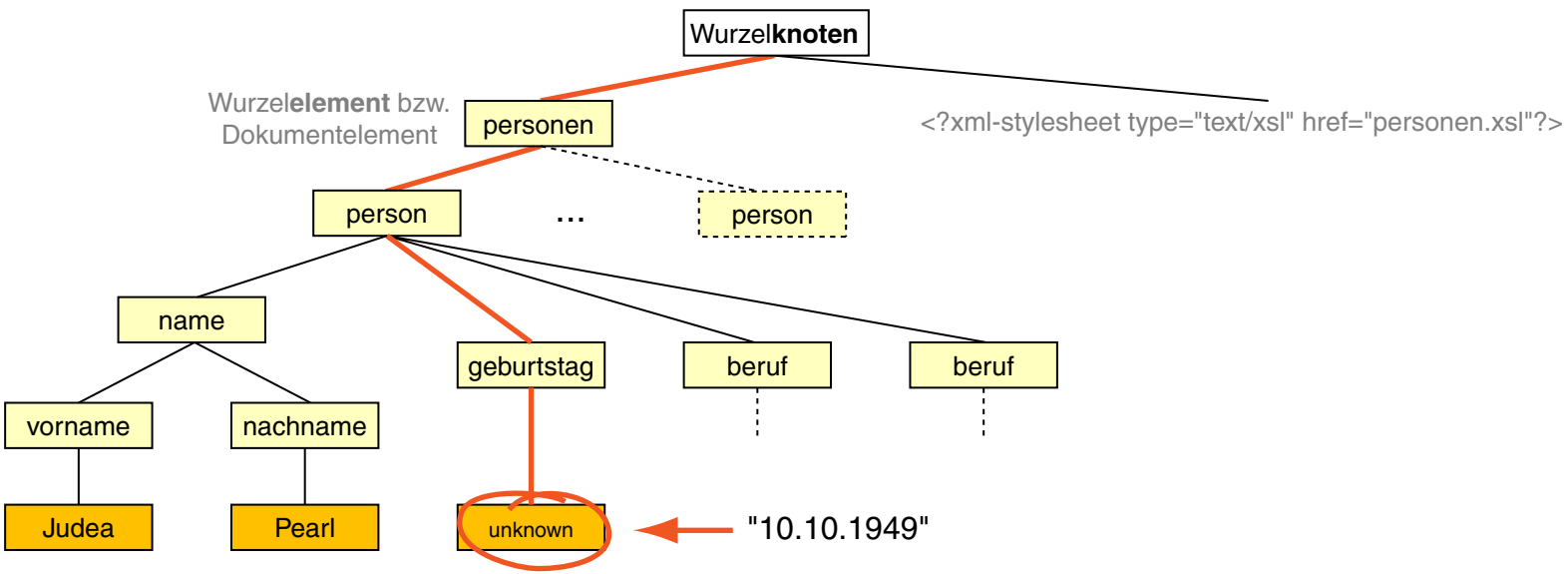
```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

APIs für XML-Dokumente

DOM: Anwendung (Fortsetzung)

Aufgabe:

1. Die Person „Judea Pearl“ finden.
2. Seinen Geburtstag auf einen bestimmten Wert setzen.



APIs für XML-Dokumente

DOM: Anwendung (Fortsetzung)

Java-Klasse:

```
package documentlanguages.xmlparser.dom;

import java.io.*;
import org.w3c.dom.*;
import org.xml.sax.*;
import javax.xml.*;

public class DomParserExample {

    public Document load(String filename)...

    public Node findPerson(Node node, String firstName, String lastName)...
    private boolean nodeMatchesPerson(Node n, String firstName, ...
    public void setBirthday(Node personNode, String birthday) ...

    public void save(Document docNode, String filename, String encoding)...

    public static void main(String[] args) {...
}
```

APIs für XML-Dokumente

DOM: Anwendung (Fortsetzung)

`main`-Methode:

```
public static void main(String[] args) throws Exception {

    DomParserExample dpe = new DomParserExample();
    Document docNode =
        dpe.load("./bin/documentlanguages/xmlparser/personen.xml");

    Node personNode = dpe.findPerson(docNode, "Judea", "Pearl");
    dpe.setBirthday(personNode, "10.10.1949");

    dpe.save(docNode,
        "./bin/documentlanguages/xmlparser/personen-neu.xml", "UTF-8");
}
```

APIs für XML-Dokumente

DOM: Anwendung (Fortsetzung)

DOM-Parser instantiieren und Dokument in DOM-Objektmodell einlesen:

```
public Document load(String filename)
         throws ParserConfigurationException, IOException, SAXException {

    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    DocumentBuilder docBuilder = dbf.newDocumentBuilder();
    return docBuilder.parse(new File(filename));
}
```

APIs für XML-Dokumente

DOM: Anwendung (Fortsetzung)

Navigation im Dokumentbaum. DFS mit generischem `node`-Interface [Javadoc] :

```
public Node findPerson(Node node, String firstName, String lastName)
{
     if(nodeMatchesPerson(node,firstName,lastName))
     {
          return node;
     }
     else // Perform Depth First Search (DFS).
     {
          NodeList nodeList = node.getChildNodes();
          for(int i=0; i< nodeList.getLength(); ++i)
          {
               Node person = findPerson(nodeList.item(i),firstName,lastName);
               if(person!=null){return person;}
          }
          return null;
     }
}
```

Vergleiche XPath-Variante.

APIs für XML-Dokumente

DOM: Anwendung (Fortsetzung)

Navigation im Dokumentbaum. DFS mit generischem `node`-Interface:

```
private boolean nodeMatchesPerson(Node n, String firstName, String lastName){
     if(!n.getNodeName().equals("person")) {return false;}
     NodeList personChildren = n.getChildNodes();
     for(int i=0; i< personChildren.getLength(); ++i){
          Node personChild = personChildren.item(i);
          if(personChild.getNodeName().equals("name")){
               boolean FIRSTNAME_OK = false, LASTNAME_OK = false;
               NodeList nameChildren = personChild.getChildNodes();
               for(int j=0; j< nameChildren.getLength(); ++j){
                    Node nameChild = nameChildren.item(j);
                    if(nameChild.getNodeName().equals("vorname") &&
                       nameChild.getTextContent().equals(firstName))
                    {FIRSTNAME_OK = true;}
                    else if(nameChild.getNodeName().equals("nachname") &&
                            nameChild.getTextContent().equals(lastName))
                    {LASTNAME_OK = true;}
                    if(FIRSTNAME_OK && LASTNAME_OK) {return true;}
     }}}
     return false;
 }
```

APIs für XML-Dokumente

DOM: Anwendung (Fortsetzung)

Navigation im Dokumentbaum. DFS mit generischem `node`-Interface:

```
private boolean nodeMatchesPerson(Node n, String firstName, String lastName){
    if(!n.getNodeName().equals("person")) {return false;}
    NodeList personChildren = n.getChildNodes();
    for(int i=0; i< personChildren.getLength(); ++i){
        Node personChild = personChildren.item(i);
        if(personChild.getNodeName().equals("name")){
            boolean FIRSTNAME_OK = false, LASTNAME_OK = false;
            NodeList nameChildren = personChild.getChildNodes();
            for(int j=0; j< nameChildren.getLength(); ++j){
                Node nameChild = nameChildren.item(j);
                if(nameChild.getNodeName().equals("vorname") &&
                   nameChild.getTextContent().equals(firstName))
                {FIRSTNAME_OK = true;}
                else if(nameChild.getNodeName().equals("nachname") &&
                        nameChild.getTextContent().equals(lastName))
                {LASTNAME_OK = true;}
                if(FIRSTNAME_OK && LASTNAME_OK) {return true;}
    }}}
    return false;
 }
```

APIs für XML-Dokumente

DOM: Anwendung (Fortsetzung)

Navigation im Dokumentbaum. DFS mit generischem `node`-Interface:

```
private boolean nodeMatchesPerson(Node n, String firstName, String lastName){
    if(!n.getNodeName().equals("person")) {return false;}
    NodeList personChildren = n.getChildNodes();
    for(int i=0; i< personChildren.getLength(); ++i){
        Node personChild = personChildren.item(i);
        if(personChild.getNodeName().equals("name")){
            boolean FIRSTNAME_OK = false, LASTNAME_OK = false;
            NodeList nameChildren = personChild.getChildNodes();
            for(int j=0; j< nameChildren.getLength(); ++j){
                Node nameChild = nameChildren.item(j);
                if(nameChild.getNodeName().equals("vorname") &&
                   nameChild.getTextContent().equals(firstName))
                {FIRSTNAME_OK = true;}
                else if(nameChild.getNodeName().equals("nachname") &&
                        nameChild.getTextContent().equals(lastName))
                {LASTNAME_OK = true;}
                if(FIRSTNAME_OK && LASTNAME_OK) {return true;}
    }}}
    return false;
 }
```

APIs für XML-Dokumente

DOM: Anwendung (Fortsetzung)

`<geburtstag>`-Knoten im `<person>`-Knoten ändern. Generisches `node`-Interface:

```
public void setBirthday(Node personNode, String birthday){

    NodeList personChildren = personNode.getChildNodes();
    for(int i=0; i<personChildren.getLength(); ++i){
        Node personChild = personChildren.item(i);
        if(personChild.getNodeName().equals("geburtstag")){
            System.out.println("[DOM] Updating geburtstag: " +
                personChild.getTextContent()+" -> "+birthday);
            personChild.setTextContent(birthday);
        }
    }
}
```

APIs für XML-Dokumente

DOM: Anwendung (Fortsetzung)

`<geburtstag>`-Knoten im `<person>`-Knoten ändern. `Element`-Interface [Javadoc] :

```
public void setBirthday(Node personNode, String birthday) {

    Element person = (Element) personNode;
    NodeList birthdayElements = person.getElementsByTagName("geburtstag");
    if (birthdayElements.getLength()>0)
    {
        System.out.println("[DOM] Updating geburtstag: " +
            birthdayElements.item(0).getTextContent()+" -> "+birthday);
        birthdayElements.item(0).setTextContent(birthday);
    }
}
```

APIs für XML-Dokumente

DOM: Anwendung (Fortsetzung)

Dokumentmodell serialisieren:

```
public void save(Document docNode, String filename, String encoding)
          throws IOException, TransformerException,
          UnsupportedEncodingException {

     Transformer serializer =
          TransformerFactory.newInstance().newTransformer();

     serializer.setOutputProperty(OutputKeys.ENCODING, encoding);
     serializer.transform(new DOMSource(docNode),
          new StreamResult(new FileOutputStream(filename)));
}
```

APIs für XML-Dokumente

DOM: Anwendung (Fortsetzung)

Aufruf in der Shell:

```
[user@pc WORKINGDIR]$ java -cp CLASSPATH
 documentlanguages.xmlparser.dom.DomParserExample
```

```
[DOM] Updating geburtstag: unknown -> 10.10.1949
```

APIs für XML-Dokumente

DOM: Anwendung (Fortsetzung)

Navigation im Dokumentbaum. Direkter Zugriff mit `xpath`-Interface [Javadoc] :

```
public Node findPerson(Node node, String firstName, String lastName)
         throws XPathExpressionException{

     XPathFactory factory = XPathFactory.newInstance();
     XPath xpath = factory.newXPath();
     String path = "//person[name/vorname= '"+firstName+"' and " +
          "name/nachname='"+lastName +"']";
     return (Node) xpath.evaluate(path, node, XPathConstants.NODE);
}
```

Vergleiche DFS-Variante.

APIs für XML-Dokumente

SAX: Historie

SAX, die *Simple API for XML* entstand aus dem Wunsch, die Entwicklung von Programmen zur Verarbeitung von XML-Dokumenten (XML-Prozessoren) zu vereinfachen.

1997 Unter der Koordination von David Megginson entwickeln Teilnehmer der XML-DEV Mailing List einen einfachen, effizienten Parser.

1998 SAX 1.0 wird herausgegeben.

2002 SAX 2.02 wird herausgegeben. Unterstützung von Namensräumen.

2003 Java JDK 1.4 mit SAX2-Implementierung. [Javadoc]

Bemerkungen:

- SAX stellt einen „leichtgewichtigen“ Ansatz für die ereignisbasierte Verarbeitung von XML-Dokumenten dar. Diese Charakterisierung bezieht sich sowohl auf den Implementierungsaufwand der API selbst, als auch auf ihren Integrationsaufwand in eigene Applikationen.
- Ursprünglich entstand die SAX-API aus einer Sammlung generischer Java-Schnittstellen für XML-Parser. Inzwischen hat sie sich als eigenständige Möglichkeit zur Verarbeitung von XML-Dokumenten in verschiedenen Hochsprachen entwickelt. Neben den Umsetzungen für Java existieren auch Implementierungen für C++, Python, Perl und Eiffel.
- SAX ist kein Parser sondern ein Gerüst in Form einer Schnittstellensammlung, um Parser zu implementieren.
- David Megginson hat eine Firma gegründet und sich selbstständig gemacht.

APIs für XML-Dokumente

SAX: Konzepte

Verwendung eines SAX-Parsers in folgenden Schritten:

1. Instantiierung einer spezifischen Parser-Ausprägung.
   Stichwort: Factory-Pattern
2. Implementierung und Zuweisung eines Content-Handlers.
3. Aufruf des Parsers.

Konsequenzen:

- Das Dokument definiert die Ereignisse, auf die der Parser reagiert.
- Parse-Methoden werden nicht explizit vom Programmierer aufgerufen.
- Das Programm weist keinen erkennbaren Parse-Kontrollfluss auf.

Stichwort: Push-Prinzip

Bemerkungen:

- Die SAX-API impliziert keine spezielle Datenstruktur. Deshalb ist der Ansatz mit nur geringen Modifikationen auf viele Programmiersprachen übertragbar.
- Das Push-Prinzip stellt nur minimale Anforderungen an den Hauptspeicher: nur die Tiefe des Dokumentbaums und die Übergabeparameter der Callback-Funktionen sind verantwortlich für den variablen Teil des “Memory Footprint”.
- Aus dem Prinzip der SAX-Verarbeitung folgt, dass keine (in-memory) Modifikationen am Eingabedokument möglich sind. Modifikationen werden durch eine veränderte Ausgabe des Eingabedokuments realisiert. Der mögliche Umfang dieser Transformationen hängt davon ab, wieviel von dem Eingabedokument während des Parse-Vorgangs zwischenspeichert wird.
- Wikipedia-Artikel zur SAX-API.

APIs für XML-Dokumente

SAX: Struktur der API

1. Parser-Factory.
   Dient zur Erzeugung verschiedener Ausprägungen eines Parsers. Optionen sind u. a.: validierend, nicht-validierend, Namensraum-auswertend.

2. Parser.
   Definiert abstrakte Schnittstellen und bedient die Callback-Funktionen in diesen Schnittstellen beim Eintreffen der entsprechenden Ereignisse.

3. Schnittstellen.

(a) `ContentHandler`	Methoden zur Reaktion auf Dokumentereignisse
(b) `ErrorHandler`	Methoden zur Reaktion auf in der XML-Spezifikation definierte Fehlerereignisse: `warning, error, fatalError`
(c) `DTDHandler`	Methoden für Notation-Deklarationen und ungeparste Entities
(d) `EntityResolver`	Methoden zur Namensauflösung von Entities

APIs für XML-Dokumente

SAX: Struktur der API (Fortsetzung)

Wichtige Methoden (Callback-Funktionen) der `ContentHandler`-Schnittstelle:

Methode	**Beschreibung**
`startDocument()`	einmaliger Aufruf bei Beginn eines Dokuments
`startElement()`	Aufruf bei Beginn (öffnender Tag) eines Elements
`characters()`	Aufruf bei der Verarbeitung von Zeichenkettendaten innerhalb eines Elements
`ignorableWhitespace()`	Aufruf beim Auftreten ignorierbarer Leerzeichen.
`endElement()`	Aufruf bei Erreichen eines Elementendes
`endDocument()`	letztes Ereignis eines Parse-Vorgangs

APIs für XML-Dokumente

SAX: Anwendung

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

APIs für XML-Dokumente

SAX: Anwendung

```
<?xml version="1.0" standalone="no" ?>   -> startDocument()
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

APIs für XML-Dokumente

SAX: Anwendung

```
<?xml version="1.0" standalone="no" ?>   -> startDocument()
<?xml-stylesheet type="text/xsl" ...?>      processingInstruction()

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

APIs für XML-Dokumente

SAX: Anwendung

```
<?xml version="1.0" standalone="no" ?>   -> startDocument()
<?xml-stylesheet type="text/xsl" ...?>      processingInstruction()

<personen>                                  startElement()
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

APIs für XML-Dokumente

SAX: Anwendung

```
<?xml version="1.0" standalone="no" ?>   -> startDocument()
<?xml-stylesheet type="text/xsl" ...?>      processingInstruction()

<personen>                                  startElement()
  <person>                                  startElement()
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

APIs für XML-Dokumente

SAX: Anwendung

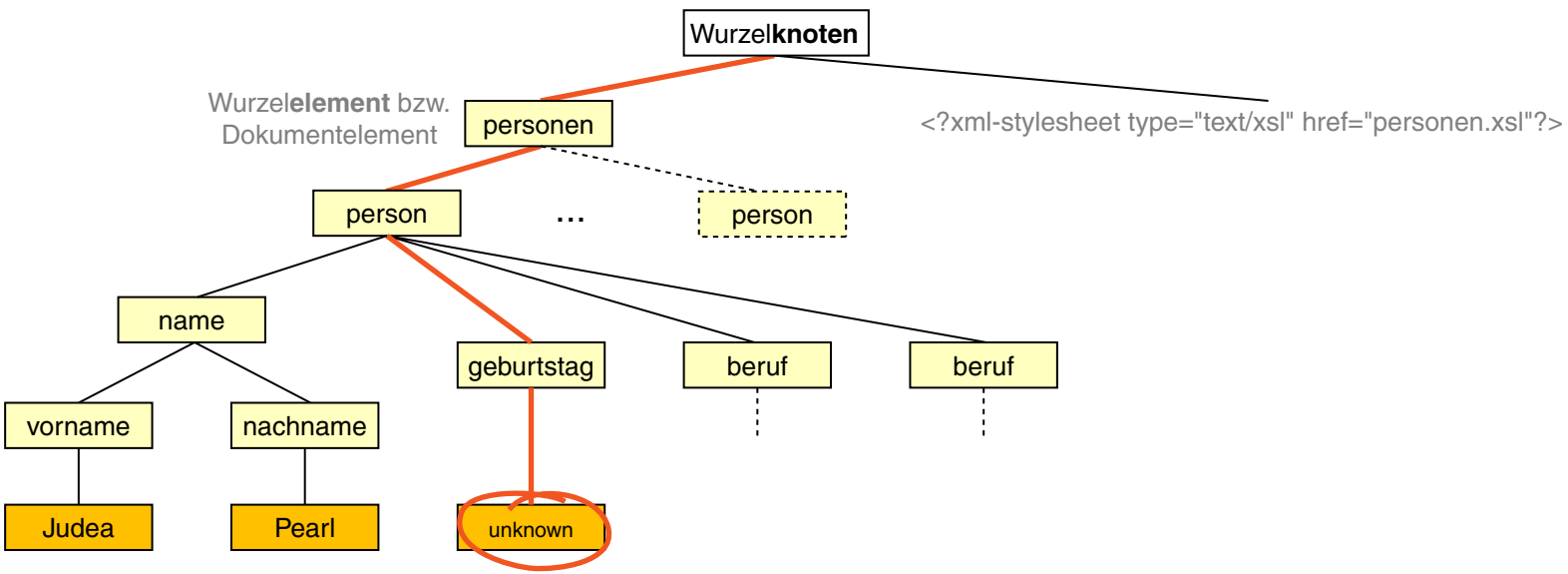
```
<?xml version="1.0" standalone="no" ?>   -> startDocument()
<?xml-stylesheet type="text/xsl" ...?>      processingInstruction()

<personen>                                  startElement()
  <person>                                  startElement()
    <name>                                  startElement()
      <vorname>Alan</vorname>               startElement()characters()...
      <nachname>Turing</nachname>           startElement()characters()...
    </name>                                 endElement()
    <geburtstag>23. Juni 1912</geburtstag>  startElement()characters()...
    <beruf>Mathematiker</beruf>             startElement()characters()...
    <beruf>Informatiker</beruf>             startElement()characters()...
  </person>                                 endElement()

  <person>                                  ...
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

APIs für XML-Dokumente

SAX: Anwendung (Fortsetzung)

Aufgabe:

1. Die Person „Judea Pearl“ finden.
2. Seinen Geburtstag ausgeben.



APIs für XML-Dokumente

SAX: Anwendung (Fortsetzung)

SAX-Parser instantiieren und XML-Ereignisstrom öffnen:

```
public class SAXParserExample {

    public void load(String filename, DefaultHandler handler)
            throws SAXException, IOException {
        XMLReader xr = XMLReaderFactory.createXMLReader();
        xr.setContentHandler(handler);
        xr.parse(filename);
    }

    public static void main(String[] args) {
        SAXParserExample spe = new SAXParserExample();
        try {

         DefaultHandler handler=new SAXParserExampleHandler("Judea", "Pearl");

         spe.load("./bin/documentlanguages/xmlparser/personen.xml", handler);


        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
}
```

APIs für XML-Dokumente

SAX: Anwendung (Fortsetzung)

Schnittstellenklasse mit eigenen Callback-Funktionen:

```
public class SAXParserExampleHandler extends DefaultHandler{

    boolean parseVorname=false;
    boolean parseNachname=false;
    boolean parseGeburtstag=false;
    String vorname, nachname, geburtstag, targetFirstName, targetLastName;

    public SAXParserExampleHandler(String firstName, String lastName){
        this.targetFirstName=firstName;
        this.targetLastName=lastName;
    }

    public void startElement(String uri, String localName, String qName,...

    public void endElement(String uri, String localName, String qName){...

    public void characters(char[] ch, int start, int length){...

    private String readString(char[] ch, int start, int length){...

}
```

Bemerkungen:

- Die Klasse `SAXParserExampleHandler` ist von der Klasse `DefaultHandler` abgeleitet, die für jede Callback-Funktion eine Default-Implementierung enthält, die nichts tut. Somit müssen nur diejenigen Methoden überschrieben werden, die genau die Ereignisse verarbeiten, an denen man interessiert ist.

APIs für XML-Dokumente

SAX: Anwendung (Fortsetzung)

Verarbeitung von Elementstart-Ereignissen (= Zustandsmarkierung):

```
public void startElement(String uri, String localName, String qName,
         Attributes attributes) {

    parseVorname=false;
    parseNachname=false;
    parseGeburtstag=false;

    if(qName.equals("vorname"))
         parseVorname=true;
    else if(qName.equals("nachname"))
         parseNachname=true;
    else if(qName.equals("geburtstag"))
         parseGeburtstag=true;

 }
```

APIs für XML-Dokumente

SAX: Anwendung (Fortsetzung)

Verarbeitung von Character-Data-Ereignissen (= Einlesen):

```
public void characters(char[] ch, int start, int length){

     if(parseVorname){
          vorname=readString(ch, start, length);
     }
     if(parseNachname){
          nachname=readString(ch, start, length);
     }
     if(parseGeburtstag){
          geburtstag=readString(ch, start, length);
     }

}
```

APIs für XML-Dokumente

SAX: Anwendung (Fortsetzung)

Verarbeitung von Elementende-Ereignissen:

```
public void endElement(String uri, String localName, String qName){

     parseVorname=false;
     parseNachname=false;
     parseGeburtstag=false;

     if(qName.equals("geburtstag")){
     // Then a pair (vorname,nachname) has already been parsed.
     // Check for a match.
     if(vorname.equals(targetFirstName) && nachname.equals(targetLastName)){
          System.out.println("[SAX] "+targetFirstName+" "
               +targetLastName+"'s Geburtstag: "+geburtstag);
          }
     }

}
```

APIs für XML-Dokumente

SAX: Anwendung (Fortsetzung)

Aufruf in der Shell:

```
[user@pc WORKINGDIR]$ java -cp CLASSPATH
 documentlanguages.xmlparser.sax.SAXParserExample
```

`[SAX] Judea Pearl's Geburtstag: unknown`

Bemerkungen:

- Um sinnvoll auf Ereignisse reagieren zu können, muss sich der Zustand gemerkt werden, in dem sich der Parser befindet. Im Beispiel: tritt ein Character-Data-Ereignis ein, so soll sich die Zeichenkette nur dann gemerkt werden, falls unmittelbar vorher der Start-Tag eines `<vorname>`-, `<nachname>`- oder `<geburtstag>`-Elements geparsed wurde.
  Stichwort: endlicher Automat
- Im Beispiel sind die Zustände des endlichen Automaten durch Variablen wie `parseVorname`, `parseNachname` oder `parseGeburtstag` codiert. Ein Elementende-Ereignis setzt den endlichen Automat wieder in seinen Anfangszustand.

APIs für XML-Dokumente

XML Data Binding: Historie

> XML Data Binding Specification: *"A facility for compiling an XML schema into one or more Java classes which can parse, generate, and validate documents that follow the schema."* [jcp.org: JSR 31]

1999 Java Specification Request 31 durchläuft erste Begutachtungsphase.

2000 CASTOR XML 0.8. Open Source Implementierung der XML Data Binding Spezifikation.

2004 Java JDK 1.5 mit Referenz-Implementierung von JAXB, der Java Architecture for XML Bindings. [Javadoc]

2006 JAXB 2.0 [jcp.org: JSR 222]

2012 JAXB 2.26 [GlassFish]

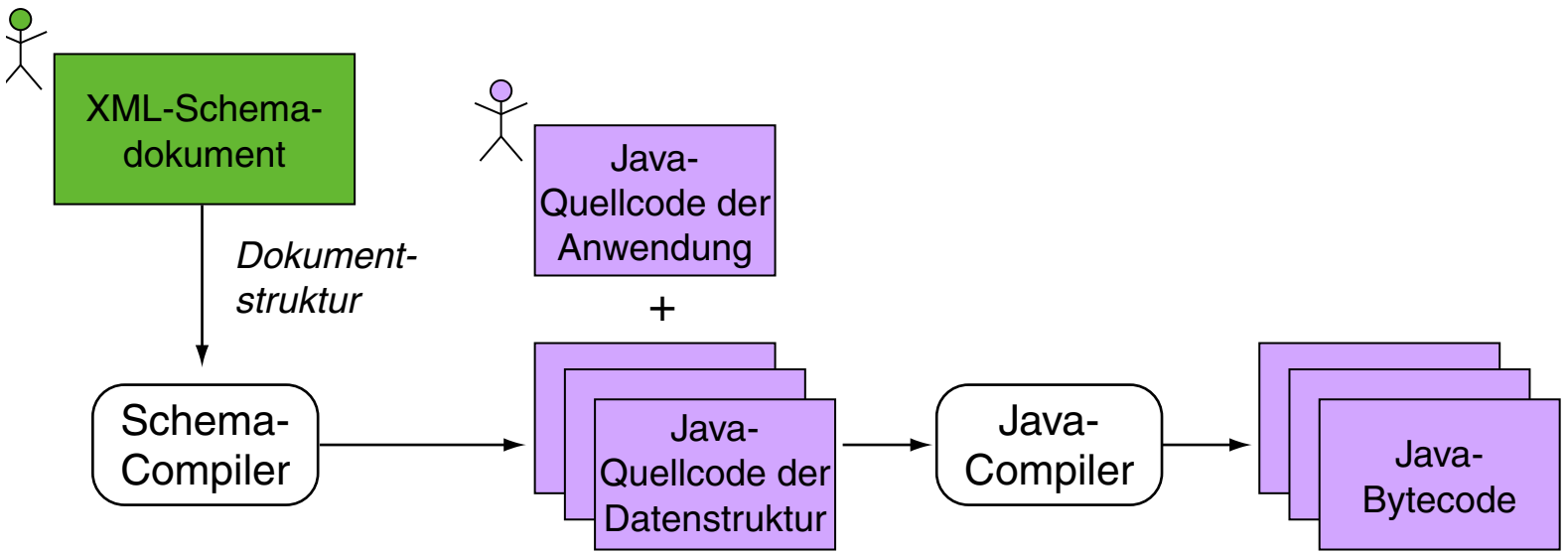
2012 CASTOR XML 1.3.3. [castor.codehaus.org]

Bemerkungen:

- Unter Data Binding versteht man die Abbildung einer gegebenen Datenstruktur (hier: XML-Schema) auf die Konzepte einer Zielsprache. Data Binding macht die Daten auf *Basis der Datenstrukturen der gewählten Zielsprache* verfügbar.
- Data Binding wird attraktiv, wenn Mechanismen für dessen Automatisierung existieren: die Konzepte der Zielsprache (Beschränkungen, Datenstruktur-Mapping, Setter / Getter-Methoden, etc.) werden aus Sicht des Programmierers transparent gehandelt.
- DOM versus XML Data Binding:
  "In the DOM approach, the parser creates a tree of objects that represents the content and organization of data in the document. The application can then navigate through the tree in memory to access the data it needs. DOM data, however, is contained in objects of a single type, linked according to the XML document's structure, with individual node objects containing an element, an attribute, a CDATA section, etc. Values are invariably provided as strings.
  Unmarshalling an XML document with the appropriate JAXB method also results in a tree of objects, with the significant difference being that the nodes in this tree correspond to XML elements, which contain attributes and the content as instance variables and refer to child elements by object references." [GlassFish]

APIs für XML-Dokumente

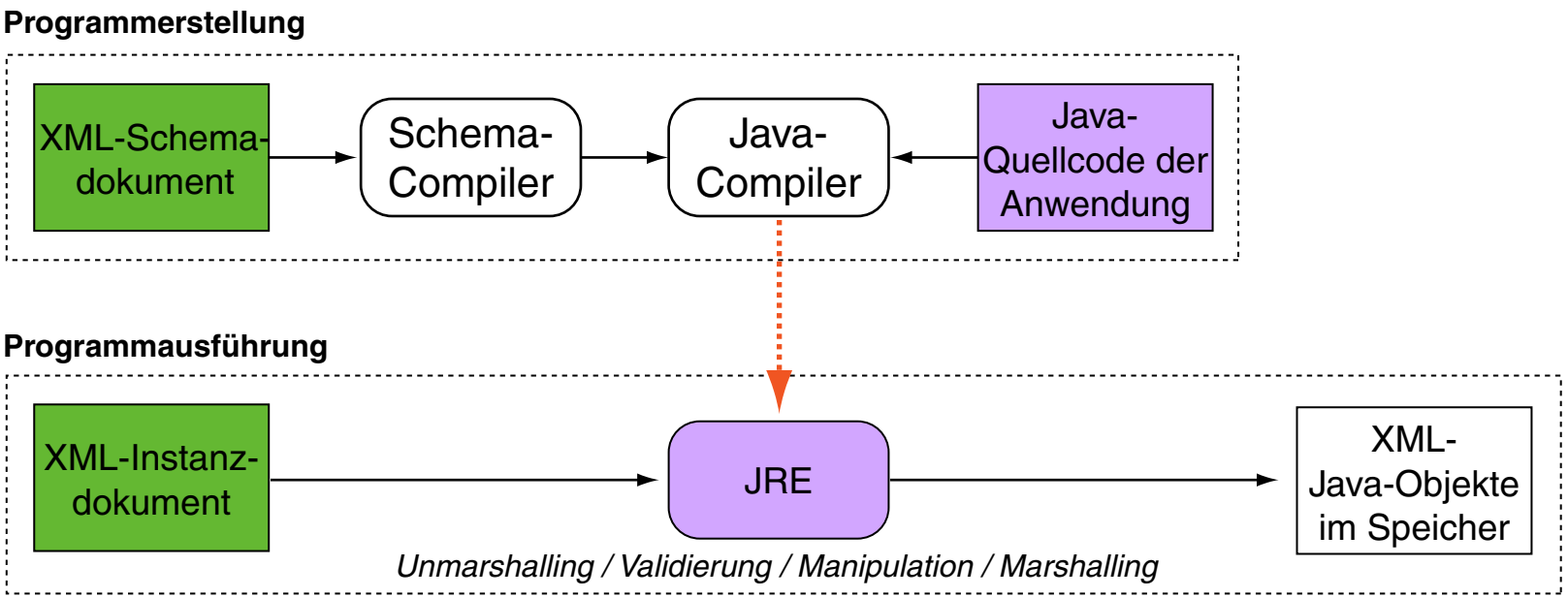
XML Data Binding: Konzepte [Oracle]



- Ein Schema-Compiler erzeugt aus dem XML-Schema Java-Klassen, die eine Repräsentation, den Zugriff und die Manipulation der Inhalte von Schema-validen XML-Dokumenten implementieren.
- Die eigene Anwendung setzt direkt auf den generierten Java-Klassen auf.

APIs für XML-Dokumente

XML Data Binding: Konzepte (Fortsetzung)



- XML-Schema für Programmerstellung, Instanzdokument für Programmausführung.
- Unmarshalling: Lese- und Validierungsvorgang, der eine XML-Datei aus einem Eingabestrom liest und die notwendigen Speicherobjekte erzeugt.
- Marshalling: Schreiben der Speicherobjekte als XML-Datei.

APIs für XML-Dokumente

XML Data Binding: Konzepte (Fortsetzung)

1. xjc.
   XML-Schema-Compiler. Erzeugt Java-Klassen, die die Struktur der Daten gemäß eines XML-Schemadokuments abbilden.

2. JAXBContext.
   Erzeugung einer Factory-Klasse für folgende Klassen:

   (a) `unmarshaller` — Bildet einen XML-Stream mit Hilfe von Java-Objekten (Instanzen der von `xjc` erzeugten Klassen) im Speicher ab.

   (b) `marshaller` — Serialisiert die gespeicherte Objektstruktur als XML-Stream.

   (c) `validator` — Validiert eine gegebene XML-Datei gemäß dem zugrunde liegenden XML-Schema des JAXBContext-Objekts.

3. Getter- und Setter-Methoden.
   Manipulation von Element- und Attributwerten.

APIs für XML-Dokumente

XML Data Binding: Anwendung

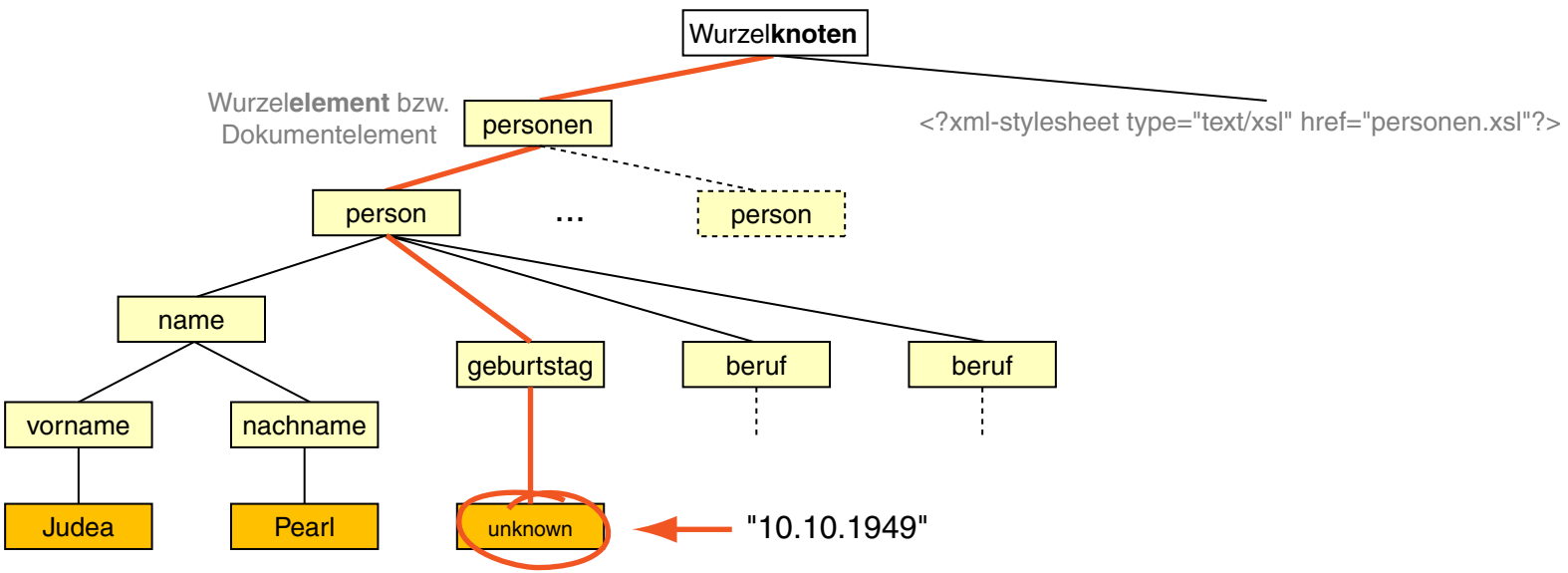
```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl" ...?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

APIs für XML-Dokumente

XML Data Binding: Anwendung (Fortsetzung)

Aufgabe:

1. Die Person „Judea Pearl“ finden.
2. Seinen Geburtstag auf einen bestimmten Wert setzen.



APIs für XML-Dokumente

XML Data Binding: Anwendung (Fortsetzung)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" ...>
  <xsd:complexType name="NameType">
    <xsd:sequence>
      <xsd:element name="vorname" type="xsd:string" />
      <xsd:element name="nachname" type="xsd:string" />
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="PersonType">
    <xsd:sequence>
      <xsd:element name="name" type="NameType" />
      <xsd:element name="geburtstag" type="xsd:string" />
      <xsd:element name="beruf" type="xsd:string" minOccurs="0" .../>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="PersonenType">
    <xsd:sequence>
      <xsd:element name="person" type="PersonType" minOccurs="0" .../>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:element name="personen" type="PersonenType" />
</xsd:schema>
```

APIs für XML-Dokumente

XML Data Binding: Anwendung (Fortsetzung)

Generierung der Elementtypklassen mit Getter- und Setter-Methoden (= Data Binding) mit dem Java-Schema-Compiler xjc:

1. Schema für die Datei `personen.xml`:
   SOURCEDIR`/documentlanguages/xmlparser/personen.xsd`

2. Angabe der Target-Package-Option `-p` derart, dass die generierten Java-Klassen im gewünschten Package eingegliedert sind:

```
[user@pc SOURCEDIR]$ xjc
 -p documentlanguages.xmlparser.jaxb.generated.personen
 documentlanguages/xmlparser/personen.xsd
```

APIs für XML-Dokumente

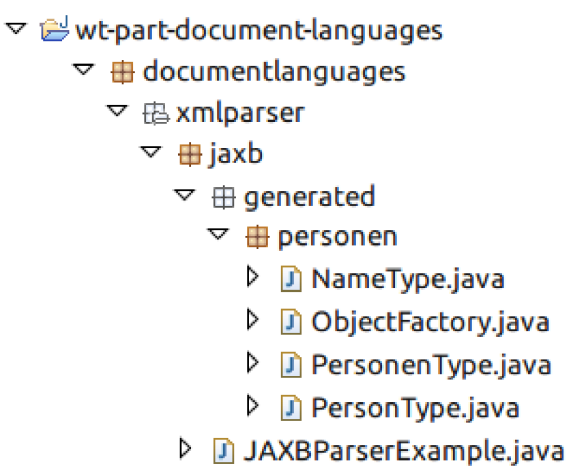
XML Data Binding: Anwendung (Fortsetzung)

Generierung der Elementtypklassen mit Getter- und Setter-Methoden (= Data Binding) mit dem Java-Schema-Compiler xjc:

1. Schema für die Datei `personen.xml`:
   SOURCEDIR`/documentlanguages/xmlparser/personen.xsd`

2. Angabe der Target-Package-Option `-p` derart, dass die generierten Java-Klassen im gewünschten Package eingegliedert sind:
   ```
   [user@pc SOURCEDIR]$ xjc
    -p documentlanguages.xmlparser.jaxb.generated.personen
    documentlanguages/xmlparser/personen.xsd
   ```

3. Die entstandene Package-Struktur [Konzepte] :



APIs für XML-Dokumente

XML Data Binding: Anwendung (Fortsetzung)

Java-Klasse:

```
package documentlanguages.xmlparser.jaxb;

import documentlanguages.xmlparser.jaxb.generated.personen.*;

import java.io.*;
import java.util.*;
import javax.xml.bind.*;

public class JAXBParserExample {

    public PersonenType load(String filename){...
    public boolean setBirthday(PersonenType personen, ...
    public void save(PersonenType personen, String filename){...

    public static void main(String[] args){
        JAXBParserExample pe=new JAXBParserExample();
        PersonenType personen=
            pe.load("./bin/documentlanguages/xmlparser/personen.xml");
        pe.setBirthday(personen, "Judea", "Pearl", "10.10.1949");
        pe.save(personen,
            "./bin/documentlanguages/xmlparser/personen-neu.xml");
    }
}
```

APIs für XML-Dokumente

XML Data Binding: Anwendung (Fortsetzung)

`<personen>`-Parser instantiieren, Dokument parsen und `<personen>`-Element im Speicher als Java-Objekt anlegen [Konzepte] [Javadoc] :

```
public PersonenType load(String filename){
     try {
          // Create JAXBContext.
          JAXBContext jc = JAXBContext.newInstance(
                    "./bin/documentlanguages.xmlparser.jaxb.generated.personen");
          // Create unmarshaller.
          Unmarshaller u = jc.createUnmarshaller();
          // Unmarshal an instance document into a tree of Java content objects
          // composed of classes from the xmlparser.generated.personen package.
          JAXBElement<PersonenType> personenElement =
               (JAXBElement<PersonenType>)
               u.unmarshal(new FileInputStream(filename));

          return personenElement.getValue();
     } catch( JAXBException je ) {
          throw new RuntimeException(je);
     } catch( IOException ioe ) {
          throw new RuntimeException(ioe);
     }
}
```

Bemerkungen:

- Für die Elementtypen des Schemas wurden vom Schema-Compiler nicht nur Accessor-Methoden, sondern auch passende Klassen definiert. Beispiel: die Klasse `PersonenType`, die in den Methoden `load()` und `save()` zur Typdeklaration verwendet wird.
- Abstraktion über Datentypen in Java mittels Generics:
  "... operate on objects of various types while providing compile-time type safety." [Oracle]

APIs für XML-Dokumente

XML Data Binding: Anwendung (Fortsetzung)

`<geburtstag>`-Element suchen und neu setzen:

```
public boolean setBirthday(PersonenType personen, String targetFirstName,
        String targetLastName, String birthday) {

    List<PersonType> personList = personen.getPerson();
    Iterator<PersonType> personIterator = personList.iterator();
    while (personIterator.hasNext()) {
        PersonType p = personIterator.next();
        NameType pName = p.getName();
        if (pName.getNachname().equals(targetLastName)
                && pName.getVorname().equals(targetFirstName)) {
            System.out.println("[JAXB] Updating \"geburtstag\": "
                    + p.getGeburtstag() + " -> " + birthday);
            p.setGeburtstag(birthday);
            return true;
        }
    }
    return false;

}
```

APIs für XML-Dokumente

XML Data Binding: Anwendung (Fortsetzung)

Geändertes `<personen>`-Element speichern:

```
public void save(PersonenType personen, String filename){
     try{
          // Create JAXBContext.
          JAXBContext jc = JAXBContext.newInstance(
               "./bin/documentlanguages.xmlparser.jaxb.generated.personen");
          // Create marshaller.
          Marshaller m=jc.createMarshaller();
          PrintWriter pw = new PrintWriter(new FileWriter(filename));
          // Produce formatted output.
          m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
          // Create <personen> element from personen and write to file.
          ObjectFactory of = new ObjectFactory();
          JAXBElement<PersonenType> personenElement=of.createPersonen(personen);
          m.marshal(personenElement, pw);
          pw.close();

     } catch(JAXBException je) {
          throw new RuntimeException(je);
     } catch(IOException ioe) {
          throw new RuntimeException(ioe);
     }
}
```

APIs für XML-Dokumente

XML Data Binding: Anwendung (Fortsetzung)

Aufruf in der Shell:

```
[user@pc SOURCEDIR]$ xjc
 -p documentlanguages.xmlparser.jaxb.generated.personen
 documentlanguages/xmlparser/personen.xsd
```

```
parsing a schema...
compiling a schema...
documentlanguages/xmlparser/jaxb/generated/personen/NameType.java
documentlanguages/xmlparser/jaxb/generated/personen/ObjectFactory.java
documentlanguages/xmlparser/jaxb/generated/personen/PersonType.java
documentlanguages/xmlparser/jaxb/generated/personen/PersonenType.java
```

APIs für XML-Dokumente

XML Data Binding: Anwendung (Fortsetzung)

Aufruf in der Shell:

```
[user@pc SOURCEDIR]$ xjc
 -p documentlanguages.xmlparser.jaxb.generated.personen
 documentlanguages/xmlparser/personen.xsd
```

```
parsing a schema...
compiling a schema...
documentlanguages/xmlparser/jaxb/generated/personen/NameType.java
documentlanguages/xmlparser/jaxb/generated/personen/ObjectFactory.java
documentlanguages/xmlparser/jaxb/generated/personen/PersonType.java
documentlanguages/xmlparser/jaxb/generated/personen/PersonenType.java
```

```
[user@pc SOURCEDIR]$ javac
 documentlanguages/xmlparser/jaxb/JAXBParserExample.java
```

APIs für XML-Dokumente

XML Data Binding: Anwendung (Fortsetzung)

Aufruf in der Shell:

```
[user@pc SOURCEDIR]$ xjc
 -p documentlanguages.xmlparser.jaxb.generated.personen
 documentlanguages/xmlparser/personen.xsd
```

```
parsing a schema...
compiling a schema...
documentlanguages/xmlparser/jaxb/generated/personen/NameType.java
documentlanguages/xmlparser/jaxb/generated/personen/ObjectFactory.java
documentlanguages/xmlparser/jaxb/generated/personen/PersonType.java
documentlanguages/xmlparser/jaxb/generated/personen/PersonenType.java
```

```
[user@pc SOURCEDIR]$ javac
 documentlanguages/xmlparser/jaxb/JAXBParserExample.java
```

```
[user@pc WORKINGDIR]$ java -cp CLASSPATH
 documentlanguages.xmlparser.jaxb.JAXBParserExample
```

```
[JAXB] Updating "geburtstag": unknown -> 10.10.1949
```

APIs für XML-Dokumente

Diskussion der API-Technologien

DOM repräsentiert ein XML-Dokument explizit als einen Objektbaum.

- – das gesamte Dokument befindet sich im Speicher
- \+ Random-Access und einfache Manipulation von Dokumentbestandteilen
- \+ Laden und Speichern ist in der API (Level 3) realisiert.
- – DOM-Objekte eignen sich nur bedingt als Datenstrukturen einer Applikation

SAX repräsentiert ein XML-Dokument als einen Strom von Ereignissen

- ❑ jedes Token begegnet einem einmal
- – kein Random-Access auf die Bestandteile eines Dokuments
- – Programmierer ist selbst verantwortlich für die Speicherung
- \+ es kann flexibel bei der Speicherung von Inhalten entschieden werden

APIs für XML-Dokumente

Diskussion der API-Technologien (Fortsetzung)

XML Data Binding generiert Klassen zur Manipulation von XML-Dokumenten.

- ❑ Hinsichtlich der Speicherung ist es mit DOM vergleichbar, ist eher besser.
- \+ Generierung von Datentypen (Klassen) und Zugriffsmethoden für XML-Elemente; die Navigation durch die Baumstruktur entfällt.
- \+ Die generierten Klassen sind direkt in der Applikation verwendbar.
- \+ Hinsichtlich Speicherplatz und Laufzeit ist der Ansatz optimal.
- – Änderung des XML-Schemas erfordert die Neugenerierung der Klassen.
- – Als Teil des Java Community Process ist es kein offener Standard.

Bemerkungen:

- DOM ist vorzuziehen, falls viele Manipulationen zu machen sind und falls (fremder) Scripting-Code einfachen Zugriff haben soll. Stichwort: Browser
- SAX ist vorzuziehen, falls Effizienz eine Rolle spielt oder falls die Verarbeitung hauptsächlich Datenstrom-orientiert ist.
- DOM und SAX lassen sich in *einer* Applikation kombinieren: ein SAX-Ereignisstrom kann als Eingabe für DOM genutzt werden.
- Die Java API for XML Processing (JAXP) stellt alle notwendigen Technologien für DOM, SAX und StAX bereit. [Oracle]
- Die Java Architecture for XML Binding (JAXB) stellt die Data-Binding-Technologien bereit. [GlassFish]

APIs für XML-Dokumente

Quellen zum Nachlernen und Nachschlagen im Web: Konzepte

- T. Robie. *What is the Document Object Model?*
  www.w3.org/TR/REC-DOM-Level-1/introduction.html
- W3C DOM Working Group. *DOM FAQ.*
  www.w3.org/DOM/faq.html
- W3C Glossar.
  www.w3.org/2003/glossary
- A. Le Hore et al. *DOM Level 3 Core Specification.*
  www.w3.org/TR/DOM-Level-3-Core
- W3 Schools.
  www.w3schools.com/dom

APIs für XML-Dokumente

Quellen zum Nachlernen und Nachschlagen im Web: Anwendung

- Oracle. *Java API for XML Processing (JAXP).*
  Oracle

- GlassFish. *Metro Web Service Stack.*
  metro.java.net

- GlassFish. *Java API for XML Processing (JAXP).*
  jaxp.java.net

- GlassFish. *Java Architecture for XML Binding (JAXB).*
  jaxb.java.net

- GlassFish. *Java API for XML Web Services (JAX-WS).*
  jax-ws.java.net

- *SAX Homepage.*
  www.saxproject.org

- *Castor Homepage.*
  castor.codehaus.org