Semantic Web Technologies I

Lehrveranstaltung im WS08/09

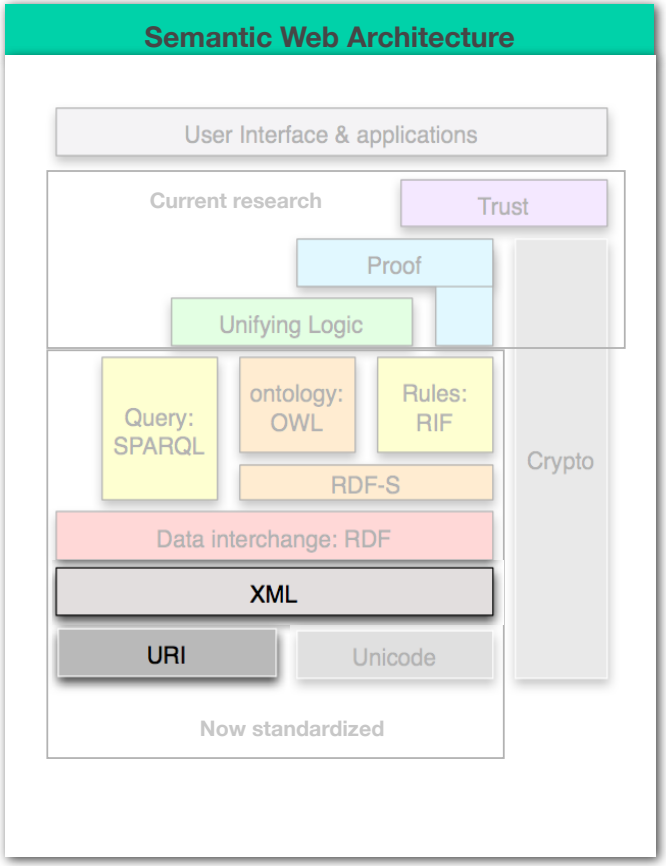
M.Sc. Markus Krötzsch
PD Dr. Pascal Hitzler
Dr. Sebastian Rudolph

AIFB

XML und URIs

Dr. Sebastian Rudolph

- Einleitung und Ausblick
- XML und URIs
- Einführung in RDF
- RDF Schema
- Logik - Grundlagen
- Semantik von RDF(S)
- OWL - Syntax und Intuition
- OWL - Semantik und Reasoning
- SPARQL - Syntax und Intuition
- Semantik von SPARQL und konjunktive Anfragen
- OWL 1.1 - Syntax und Semantik
- Semantic Web und Regeln



Agenda

- XML - Motivation/Idee
- XML - Syntax
- URIs
- Namensräume
- Schemata in XML

Agenda

- XML - Motivation/Idee
- XML - Syntax
- URIs
- Namensräume
- Schemata in XML

Annotation mit Markupsprachen

- Grundidee des Markup: versehen von (unstrukturiertem) Text mit zusätzlicher Information (bzw. Struktur)
- synonym: *auszeichnen*, auch: *annotieren* von Text
- Text = Daten
  Zusatzinformation = *Metadaten*

Annotation mit Markupsprachen

- häufige Markup-Strategie: Einschließen des zu annotierenden Textes in sogenannte *tags* (engl.: Etikett, Schild):

  <Tag-Bezeichner>... Text ...</Tag-Bezeichner>

  öffnendes Tag *schließendes Tag*

- Zusatzinformation wird von verarbeitenden Programmen gelesen und interpretiert

Annotation mit Markupsprachen

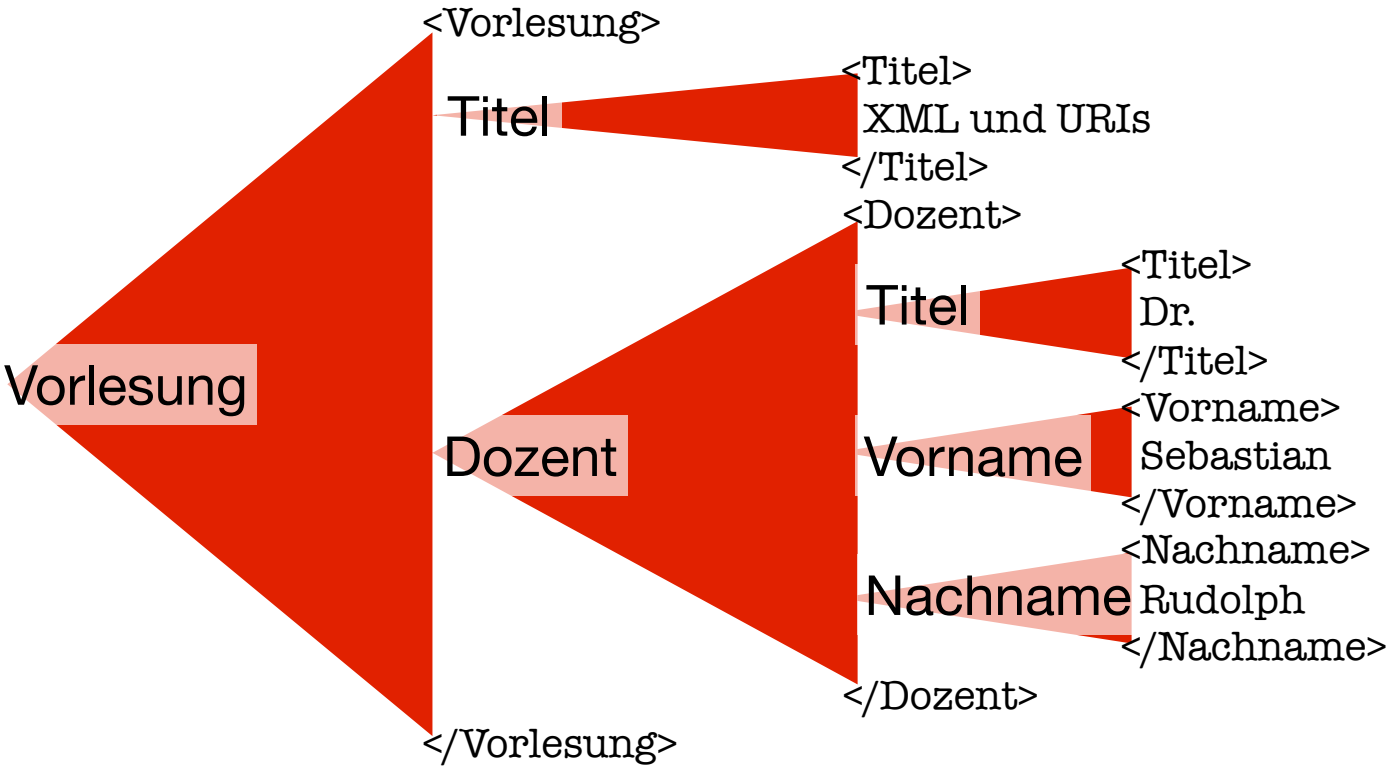
- prominentestes Beispiel: HTML
  tags kodieren Darstellungsinformationen:
  Dieses Buch hat den Titel Semantic Web Grundlagen.
- Darstellung im Browser:
  Dieses Buch hat den Titel **Semantic Web Grundlagen**.
- Strategie auch geeignet zur inhaltlichen Annotation, z.B.:
  <Buch>Dieses Buch</Buch> hat den Titel <Titel>Semantic Web Grundlagen</Titel>.

Annotation mit Markupsprachen

- Verschachtelung von Tags erlaubt:

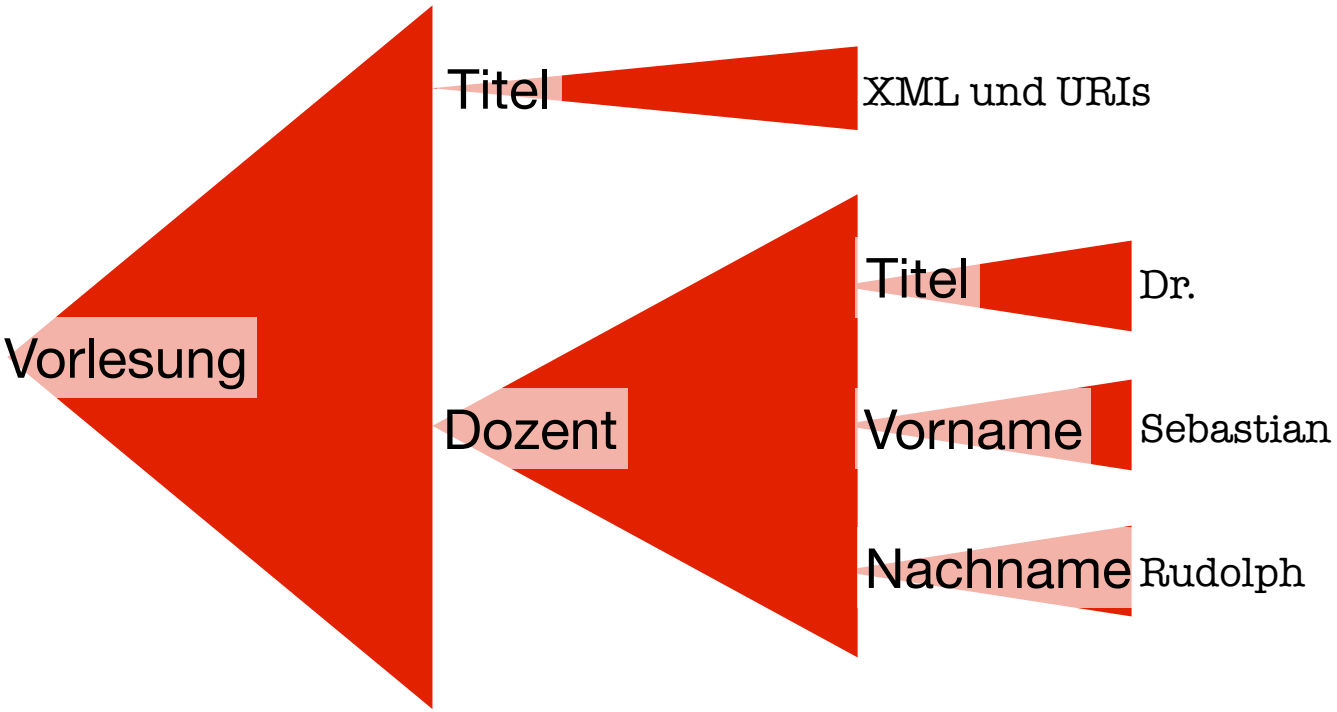
```
<Vorlesung>
    <Titel>                 <Titel>
      XML und URIs           XML und URIs
    </Titel>                </Titel>
    <Dozent>                <Dozent>
      <Titel>                                  <Titel>
        Dr.                                     Dr.
      </Titel>                                 </Titel>
      <Vorname>                                <Vorname>
        Sebastian                               Sebastian
      </Vorname>                               </Vorname>
      <Nachname>                               <Nachname>
        Rudolph                                 Rudolph
      </Nachname>                              </Nachname>
    </Dozent>               </Dozent>
</Vorlesung>
```

Annotation mit Markupsprachen



Annotation mit Markupsprachen

Baumstruktur



Agenda

- XML - Motivation/Idee
- XML - Syntax
- URIs
- Namensräume
- Schemata in XML

XML

- eXtensible Markup Language
- Ursprung: strukturierter Text (HTML4.0 ∈ XML ⊂ SGML)
- Web-Standard (W3C) zum Datenaustausch:
  - Ein- und Ausgabedaten von Anwendungen können mittels XML beschrieben werden
  - Industrie muss sich nur noch auf standardisierte Beschreibung (= Vokabular) einigen
- Komplementärsprache zu HTML:
  - HTML beschreibt die Präsentation
  - XML beschreibt den Inhalt

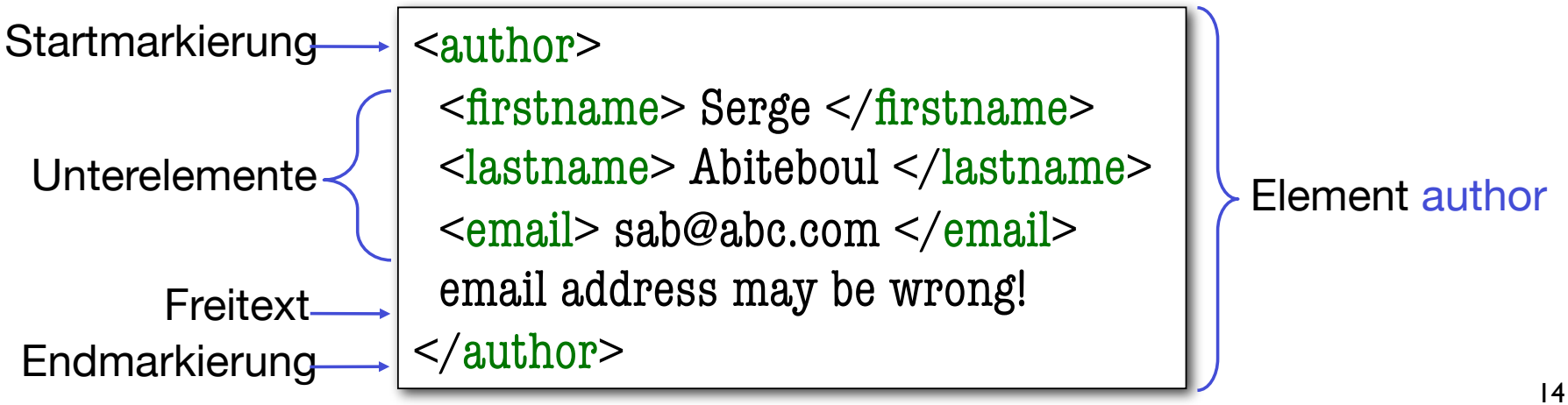
XML-Syntax (1) Präambel

- XML-Dokument ist Textdokument
- beginnt mit Deklaration, die Versionsnummer des verwendeten Standards und optional die Zeichenkodierung enthält, z.B.:

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

XML-Syntax (2) – XML-Element

- XML-Element (engl. element):
  - Beschreibung eines Objekts, die durch passende Markierungen (tags) geklammert ist
  - Inhalt eines Elements: Text und/oder weitere Elemente (beliebige Schachtelung möglich)
  - Leere Elemente: <year></year> kurz: <year/>
  - "äußerstes" Element heißt Wurzelelement (nur eines pro XML-Dokument)



XML-Syntax (3) – XML-Attribut

- XML-Attribut (engl. attribute):
  - Name-Zeichenkettenwert-Paar in Start- oder selbstschließendem Tag
  - Assoziiert mit einem Element
  - Alternative Möglichkeit, Daten zu beschreiben

Attribut email

```
<author email="sab@abc.com">
  <firstname> Serge </firstname>
  <lastname> Abiteboul </lastname>
</author>
```

Weitere denkbare Beschreibung derselben Daten:

```
<author firstname="Serge" lastname="Abiteboul" email="sab@abc.com"/>
```

XML vs. HTML

- HTML: festes Vokabular (Menge von tags) und Semantik (die Darstellung von Text)
- XML: freie Bezeichner zur Beschreibung von anwendungsspezifischer Syntax und Semantik
- XML⊂SGML

```
<h1> Bib </h1>
  <p>
    <i> Foundations of Databases </i>
     Serge Abiteboul
    < br > Addison Wesley, 1997
  <p>
    ...
```

HTML

```
<Bib id="o1">
  <paper id="o12">
    <title> Foundations of Databases </title>
    <author>
      <firstname> Serge </firstname>
      <lastname> Abiteboul </lastname>
    </author>
    <year> 1997 </year>
    <publisher> Addison Wesley </publisher>
  </paper>
  ...
</Bib>
```

XML

Agenda

- XML - Motivation/Idee
- XML - Syntax
- URIs
- Namensräume
- Schemata in XML

URIs - Idee

- URI = Uniform Resource Identifier
- dienen zur weltweit eindeutigen Bezeichnung von Ressourcen
- Ressource kann jedes Objekt sein, was (im Kontext der gegebenen Anwendung) eine klare Identität besitzt (z.B. Bücher, Orte, Menschen, Verlage, Beziehungen zwischen diesen Dingen, abstrakte Konzepte usw.)
- in bestimmten Domänen ähnliches bereits realisiert: ISBN für Bücher

URIs - Syntax

- Erweiterung des URL-Konzeptes; nicht jede URI bezeichnet aber ein Webdokument (umgekehrt wird als URI für Webdokumente häufig deren URL verwendet)
- Beginnt mit dem sogenannten URI-Schema das durch ":" vom nachfolgenden Teil getrennt ist (z.B.: http, ftp, mailto)
- häufig hierarchisch aufgebaut

Selbstdefinierte URIs

- nötig, wenn für eine Ressource (noch) keine URI existiert (bzw. bekannt ist)
- Strategie zur Vermeidung von (ungewollten) Überschneidungen: Nutzung von `http`-URIs einer eigenen Webseite
- ermöglicht auch Ablegen einer Dokumentation zur URI an dieser Stelle

Beschreibendes vs. Beschriebenes

- Trennung von URI für Ressource und deren Dokumentation durch URI-Referenzen (durch "#" angehängte Fragmente) oder content negotiation
- z.B.: als URI für Shakespeares "Othello"
  http://de.wikipedia.org/wiki/Othello
  nicht geeignet, besser
  http://de.wikipedia.org/wiki/Othello#URI

Agenda

- XML - Motivation/Idee
- XML - Syntax
- URIs
- Namensräume
- Schemata in XML

XML-Namensräume: Motivation

- XML-Dokumente besitzen Element- und Attributnamen (“Markup Vocabulary”) mit allgemeiner Gültigkeit
- Eine XML-Anwendung basiert auf allgemeiner Interpretation dieser Namen
- Ein XML-Dokument soll Markup-Vokabular aus mehreren 'Dictionaries’ enthalten können. (Erinnerung: XML-Dokument muss keine DTD haben.)
- Namespaces zur Vermeidung von Namenskonflikten.

XML-Namensräume

- XML Namespaces sind ähnlich zu Modul-Konzepten in Programmiersprachen
- Disambiguierung von Tag-Namen durch Verwendung unterschiedlicher “Prefixe”
- Ein Prefix wird vom lokalen Namen separiert durch ein “:”, so entstehen prefix:name Tags
- Namespace-Bindungen werden von manchen Werkzeugen ignoriert, sog. “flache Namespaces”

Namensraum-Bindungen

- Prefixe werden belegt mit Namespace URIs, indem ein Attribut xmlns:prefix bei dem relevanten Element oder einem seiner Vorgängerelemente eingefügt wird: prefix:name1, …, prefix:namen
- Der Wert des xmlns:prefix-Attributes ist eine URI, welche (für XML Schemata) auf eine Beschreibung auf eine Beschreibung der Namespace Syntax verweisen kann aber nicht muss
- Ein Element kann Bindings nutzen für mehrere (unterschiedliche) Namespaces durch Verwendung separater Attribute xmlns:prefix1, …, xmlns:prefixm

Beispiel: Ohne Namensräume

```
<Vorlesung>
    <Titel>
      XML und URIs
    </Titel>
    <Dozent>
      <Titel>
        Dr.
      </Titel>
      <Vorname>
        Sebastian
      </Vorname>
      <Nachname>
        Rudolph
      </Nachname>
    </Dozent>
</Vorlesung>
```

Titel ist mehrdeutig verwendeter Tagname

Zwei verschiedene Namensräume

```
<Vorlesung xmlns:lv="http://www.semantic-web-Grundlagen/Lehrveranstaltungen"
           xmlns:person:="http://www.semantic-web-Grundlagen/Person" >
    < lv:Titel>
      XML und URIs
    </lv:Titel>
    < lv:Dozent>
       < person:Titel>
          Dr.
       </person:Titel>
       < person:Vorname>
          Sebastian
       </person:Vorname>
       < person:Nachname>
          Rudolph
       </person:Nachname>
    </lv:Dozent>
</lv:Vorlesung>
```

Titel wurde disambiguiert durch Verwendung der Prefixe lv und person

Agenda

- XML - Motivation/Idee
- XML - Syntax
- URIs
- Namensräume
- Schemata in XML

Motivation

- **XML-Dokument:**
  - Ein Text-Dokument, das XML-Beschreibungen enthält
  - Datenbank-Sichtweise: sozusagen die Datenbasis
- **Wohlgeformtes XML-Dokument:**
  - Alle Elemente sind korrekt mit Start- und End-Tags geklammert
  - Dokument enthält genau ein Wurzelelement
  - Wohlgeformte Dokumente dürfen aber immer noch unstrukturierten Freitext enthalten
- **Gültiges (engl. valid) XML-Dokument:**
  - Wohlgeformtes XML-Dokument, das zu einem assoziierten *Schema* uneingeschränkt konform ist
  - Mittels eines Schemas kann man also die Gültigkeit eines XML-Dokumentes überprüfen
  - Sinnvoll beim Datenaustausch (standardisierte Beschreibung)

Schemata in XML (Optional !)

- DTD – Document Type Definitions:
  - Einfache Grammatik für ein XML-Dokument
  - Ist Teil des XML-Standards
  - Erbe von SGML
- XML Schema:
  - Komplexere Datendefinitionssprache
  - Standard (sog. „W3C Recommendation“) in Ergänzung zu XML
  - Abwärtskompatibel zu DTD

XML-Schemata I: DTD

- Eine DTD definiert eine kontextfreie Grammatik für ein XML-Dokument
- Zuvor beliebige Elemente und Attribute werden auf eine definierte Auswahl und Struktur eingeschränkt

```
<bib>
  <paper id="o12">
    <title> Foundations of Databases </title>
    <author>
      <firstname> Serge </firstname>
      <lastname> Abiteboul </lastname>
    </author>
    <year> 1997 </year>
    <publisher> Addison Wesley </publisher>
  </paper>
  ...
</bib>
```

XML

```
<!DOCTYPE bib [
  <!ELEMENT bib (paper*)>
  <!ELEMENT paper (author+, year, publisher?)>
  <!ATTLIST paper id ID #REQUIRED>
  <!ELEMENT author (firstname*, lastname)>
  <!ATTLIST author age CDATA #IMPLIED>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT year (#PCDATA)>
  <!ELEMENT publisher (#PCDATA)>
  ...
]>
```

DTD

DTD – Deklaration von Elementen

- Beschreibt die Einschränkungen des Inhalts eines Elements
- Syntax:
  <!ELEMENT Name (Definition)>
- Einziger atomarer Typ: #PCDATA (Parsed Character DATA)
- (a,b,c): Liste von Unterelementen
- (a|b|c): Alternativen
- Kardinalitäten:
  - * keinmal oder beliebig oft
  - + einmal oder beliebig oft
  - ? kein- oder einmal (optional)
  - (ohne Angabe): genau einmal
- EMPTY : Erzwingen von leerem Element

```
<!DOCTYPE bib [
  <!ELEMENT bib (paper*)>
  <!ELEMENT paper (author+, year, publisher?)>
  <!ATTLIST paper id ID #REQUIRED>
  <!ELEMENT author (firstname*, lastname)>
  <!ATTLIST author age CDATA #IMPLIED>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT year (#PCDATA)>
  <!ELEMENT publisher (#PCDATA)>
  ...
]>
```

DTD – Deklaration von Elementen

- Beschreibt die Einschränkungen des Inhalts eines Elements
- Syntax:
  <!ELEMENT Name (Definition)>
- Einziger atomarer Typ: #PCDATA (Parsed Character DATA)
- (a,b,c): Liste von Unterelementen
- (a|b|c): Alternativen
- Kardinalitäten:
  - * keinmal oder beliebig oft
  - + einmal oder beliebig oft
  - ? kein- oder einmal (optional)
  - (ohne Angabe): genau einmal
- EMPTY : Erzwingen von leeren Elementen

Einleitung und
Festlegung des Wurzelelements bib

```
<!DOCTYPE bib [
  <!ELEMENT bib (paper*)>
  <!ELEMENT paper (author+, year, publisher?)>
  <!ATTLIST paper id ID #REQUIRED>
  <!ELEMENT author (firstname*, lastname)>
  <!ATTLIST author age CDATA #IMPLIED>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT year (#PCDATA)>
  <!ELEMENT publisher (#PCDATA)>
  ...
]>
```

DTD – Deklaration von Elementen

- Beschreibt die Einschränkungen des Inhalts eines Elements
- Syntax:
  <!ELEMENT Name (Definition)>
- Einziger atomarer Typ: #PCDATA (Parsed Character DATA)
- (a,b,c): Liste von Unterelementen
- (a|b|c): Alternativen
- Kardinalitäten:
  - * keinmal oder beliebig oft
  - + einmal oder beliebig oft
  - ? kein- oder einmal (optional)
  - (ohne Angabe): genau einmal
- EMPTY : Erzwingen von leeren Elementen

bib kann beliebig viele Elemente vom Typ paper enthalten

```
<!DOCTYPE bib [
  <!ELEMENT bib (paper*)>
  <!ELEMENT paper (author+, year, publisher?)>
  <!ATTLIST paper id ID #REQUIRED>
  <!ELEMENT author (firstname*, lastname)>
  <!ATTLIST author age CDATA #IMPLIED>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT year (#PCDATA)>
  <!ELEMENT publisher (#PCDATA)>
  ...
]>
```

DTD – Deklaration von Elementen

- Beschreibt die Einschränkungen des Inhalts eines Elements
- Syntax:
  <!ELEMENT Name (Definition)>
- Einziger atomarer Typ: #PCDATA (Parsed Character DATA)
- (a,b,c): Liste von Unterelementen
- (a|b|c): Alternativen
- Kardinalitäten:
  - * keinmal oder beliebig oft
  - + einmal oder beliebig oft
  - ? kein- oder einmal (optional)
  - (ohne Angabe): genau einmal
- EMPTY : Erzwingen von leeren Elementen

paper besteht aus
mindestens einem author
genau einem year und
einem optionalen publisher
in genau dieser Reihenfolge!

```
<!DOCTYPE bib [
  <!ELEMENT bib (paper*)>
  <!ELEMENT paper (author+, year, publisher?)>
  <!ATTLIST paper id ID #REQUIRED>
  <!ELEMENT author (firstname*, lastname)>
  <!ATTLIST author age CDATA #IMPLIED>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT year (#PCDATA)>
  <!ELEMENT publisher (#PCDATA)>
  ...
]>
```

DTD – Deklaration von Elementen

- Beschreibt die Einschränkungen des Inhalts eines Elements
- Syntax:
  <!ELEMENT Name (Definition)>
- Einziger atomarer Typ: #PCDATA (Parsed Character DATA)
- (a,b,c): Liste von Unterelementen
- (a|b|c): Alternativen
- Kardinalitäten:
  - * keinmal oder beliebig oft
  - + einmal oder beliebig oft
  - ? kein- oder einmal (optional)
  - (ohne Angabe): genau einmal
- EMPTY : Erzwingen von leeren Elementen

firstname ist vom Typ Zeichenkette

```
<!DOCTYPE bib [
  <!ELEMENT bib (paper*)>
  <!ELEMENT paper (author+, year, publisher?)>
  <!ATTLIST paper id ID #REQUIRED>
  <!ELEMENT author (firstname*, lastname)>
  <!ATTLIST author age CDATA #IMPLIED>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT year (#PCDATA)>
  <!ELEMENT publisher (#PCDATA)>
  ...
]>
```

DTD – Deklaration von Attributen

- Name-Zeichenkettenwert-Paar
- Assoziiert mit einem Element
- Syntax:
  `<!ATTLIST Element Attributname1 Typ1 Zusatz1 Attributname2 ...>`
- Typ:
  - CDATA Zeichenkette
  - ID OID
  - IDREF Referenzen
  - IDREFS Menge von Referenzen
- Zusatz:
  - REQUIRED zwingend
  - IMPLIED optional
  - (Initialwert)

```
<!DOCTYPE bib [
  <!ELEMENT bib (paper*)>
  <!ELEMENT paper (author+, year, publisher?)>
  <!ATTLIST paper id ID #REQUIRED>
  <!ELEMENT author (firstname*, lastname)>
  <!ATTLIST author age CDATA #IMPLIED>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT year (#PCDATA)>
  <!ELEMENT publisher (#PCDATA)>
  ...
]>
```

DTD – Deklaration von Attributen

- Name-Zeichenkettenwert-Paar
- Assoziiert mit einem Element
- Syntax:
  `<!ATTLIST Element Attributname1 Typ1 Zusatz1 Attributname2 ...>`
- Typ:
  - CDATA Zeichenkette
  - ID OID
  - IDREF Referenzen
  - IDREFS Menge von Referenzen
- Zusatz:
  - REQUIRED zwingend
  - IMPLIED optional
  - (Initialwert)

paper besitzt ein Attribut id, eine OID, die zwingend mit einem eindeutigen Wert belegt werden muss.

```
<!DOCTYPE bib [
  <!ELEMENT bib (paper*)>
  <!ELEMENT paper (author+, year, publisher?)>
  <!ATTLIST paper id ID #REQUIRED>
  <!ELEMENT author (firstname*, lastname)>
  <!ATTLIST author age CDATA #IMPLIED>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT year (#PCDATA)>
  <!ELEMENT publisher (#PCDATA)>
  ...
]>
```

DTD – Deklaration von Attributen

- Name-Zeichenkettenwert-Paar
- Assoziiert mit einem Element
- Syntax:
  `<!ATTLIST Element Attributname1 Typ1 Zusatz1 Attributname2 ...>`
- Typ:
  - CDATA Zeichenkette
  - ID OID
  - IDREF Referenzen
  - IDREFS Menge von Referenzen
- Zusatz:
  - REQUIRED zwingend
  - IMPLIED optional
  - (Initialwert)

Ein author hat ein Attribut age, mit dem ihm eine Zeichenkette mit dem Wert für sein Alter zugewiesen werden kann (aber nicht muss!)

```
<!DOCTYPE bib [
  <!ELEMENT bib (paper*)>
  <!ELEMENT paper (author+, year, publisher?)>
  <!ATTLIST paper id ID #REQUIRED>
  <!ELEMENT author (firstname*, lastname)>
  <!ATTLIST author age CDATA #IMPLIED>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT year (#PCDATA)>
  <!ELEMENT publisher (#PCDATA)>
  ...
]>
```

DTD – OIDs und Referenzen

- DTDs erlauben die Deklaration von OIDs, Referenzen und Referenzmengen als Attribute

```
<family>
  <person id="jane" mother="mary" father="john">
    <name> Jane Doe </name>
  </person>
  <person id="john" children="jane jack">
    <name> John Doe </name>
  </person>
  <person id="mary" children="jane jack">
    <name> Mary Smith </name>
  </person>
  <person id="jack" mother="mary" father="john">
    <name> Jack Smith </name>
  </person>
</family>
```

XML

```
<!DOCTYPE family [
  <!ELEMENT family (person*)>
  <!ELEMENT person (name)>
  <!ELEMENT name (#PCDATA)>
  <!ATTLIST person
    id       ID       #REQUIRED
    mother   IDREF    #IMPLIED
    father   IDREF    #IMPLIED
    children IDREFS   #IMPLIED>
]>
```

DTD

Bewertung von DTDs

- DTDs definieren kontextfreie Grammatiken, rekursive Definitionen sind also möglich
- DTDs weisen bei der Definition eines Schemas jedoch einige Schwächen auf:
- Ungewollte Festlegung der Reihenfolge:

```
<!ELEMENT person ( name, phone ) >
```

- **Workaround:**

```
<!ELEMENT person ( (name, phone ) | ( phone, name ) ) >
```

- Kann teilweise zu vage werden:

```
<!ELEMENT person ( ( name | phone | email )* ) >
```

- Referenzen können nicht eingeschränkt (typisiert) werden
- Alle Elementnamen sind global in einem Namensraum

XML-Schemata II: XML-Schema

- Echter Schemamechanismus mit vielen Erweiterungen über DTDs hinaus
- Benutzt selbst wieder XML-Syntax zur Schemadefinition
- eigener Namensraum für Vokabular

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="bib">
  <xsd:complexType>
   <xsd:element name="paper" minOccurs="0" maxOccurs="unbounded">
    <xsd:complexType>
     <xsd:attribute name="id" type="ID" use="required"/>
     <xsd:sequence>
      <xsd:element name="author" type="authorType"
       maxOccurs="unbounded"/>
      <xsd:element name="year" type="xsd:string"/>
      <xsd:element name="publisher" type="xsd:string" minOccurs="0"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

XML-Schema

1:1-Abbildung (bis auf author)

```
<!DOCTYPE bib [
   <!ELEMENT bib (paper*)>
   <!ELEMENT paper (author+, year, publisher?)>
   <!ATTLIST paper id ID #REQUIRED>
   <!ELEMENT author (firstname*, lastname)>
   <!ATTLIST author age CDATA #IMPLIED>
   <!ELEMENT firstname (#PCDATA)>
   <!ELEMENT lastname (#PCDATA)>
   <!ELEMENT year (#PCDATA)>
   <!ELEMENT publisher (#PCDATA)>
   ...
]>
```

DTD

XML-Schema: Elemente

- Syntax: <xsd:element name="Name"/>
- Optionale Zusatzattribute:
  - Typ
    - type = "Typ" atomarer, einfacher oder komplexer Typname
  - Kardinalitäten (Vorgabe [1,1]):
    - minOccurs = "x" x ∈ {0, 1, n}
    - maxOccurs = "y" x ∈ {1, n, unbounded}
  - Wertvorgaben (schließen sich gegenseitig aus!):
    - default = "v" veränderliche Vorgabe
    - fixed = "u" unveränderliche Vorgabe

```
<xsd:element name="bib"/>
<xsd:element name="paper" minOccurs="0" maxOccurs="unbounded"/>
<xsd:element name="publisher" type="xsd:string" minOccurs="0"/>
```

XML-Schema

XML-Schema: Attribute

- Syntax: <xsd:attribute name="Name"/>
- Optionale Zusatzattribute:
  - Typ:
    - type = "Typ"
  - Existenz:
    - use = "optional" Kardinalität [0,1]
    - use = "required" Kardinalität [1,1]
  - Vorgabewerte:
    - use = "default" value = "v" veränderliche Vorgabe
    - use = "fixed" value = "u" unveränderliche Vorgabe

```
<xsd:attribute name="id" type="ID" use="required"/>
<xsd:attribute name="age" type="xsd:string" use="optional"/>
<xsd:attribute name="language" type="xsd:string" use="default" value="de"/>
```

XML-Schema: Typen

- In XML-Schema wird zwischen atomaren, einfachen und komplexen Typen unterschieden
- Atomare Typen:
  - Eingebaute Elementartypen wie int oder string
- Einfache Typen:
  - Haben weder eingebettete Elemente noch Attribute
  - In der Regel von atomaren Typen abgeleitet
- Komplexe Typen:
  - Dürfen Elemente und Attribute besitzen
- Zusätzlich kann man noch folgende Unterscheidung treffen:
  - Reine Typdefinitionen beschreiben (wiederverwendbare) Typstruktur
  - Dokumentdefinitionen beschreiben welche Elemente wie im Dokument auftauchen dürfen

XML-Schema: Atomare Typen

- XML-Schema unterstützt eine große Menge eingebauter Basistypen (>40):
  - Numerisch: byte, short, int, long, float, double, decimal, binary, …
  - Zeitangaben: time, date, month, year, timeDuration, timePeriod, …
  - Sonstige: string, boolean, uriReference, ID, …

```
<xsd:element name="year" type="xsd:year"/>
<xsd:element name="pages" type="xsd:positiveInteger"/>
<xsd:attribute name="age" type="xsd:unsignedShort"/>
```

XML-Schema

XML-Schema: Einfache Typen

- Zusätzlich können von bestehenden Typen noch weitere, sog. einfache Typen, abgeleitet werden:
  - Typdefinition:
    <xsd:simpleType name="humanAge" base="xsd:unsignedShort">
    <xsd:maxInclusive value="200"/> </xsd:simpleType>
  - Dokumentdefinition:
    <xsd:attribute name="age" type="humanAge"/>
  - Solche einfachen Typen dürfen jedoch keine verschachtelten Elemente enthalten!
- In ähnlicher Weise können Listen definiert werden:
  - Typdefinition:
    <xsd:simpleType name="authorType" base="xsd:string"
    derivedBy="list"/>
    (Name eines Autors als mit Leerzeichen getrennte Liste von Zeichenketten)
  - Dokumentdefiniton:
    <element name="author" type="authorType"/>

XML-Schema: Komplexe Typen

- Komplexe Typen dürfen im Gegensatz zu einfachen Typen eingebettete Elemente und Attribute besitzen
- Beispiel:
  - Typdefinition:

```
<xsd:complexType name="authorType">
    <xsd:sequence>
        <xsd:element name="firstname" type="xsd:string" minOccurs="0"
            maxOccurs="unbounded"/>
        <xsd:element name="lastname" type="xsd:string"/>
    </xsd:sequence>
    <xsd:attribute name="age" type="xsd:string" use="optional"/>
</xsd:complexType>
```

- Gruppierungs-Bezeichner:
  - <xsd:sequence> ... </xsd:sequence> Feste Reihenfolge (a,b)
  - <xsd:all> ... </xsd:all> Beliebige Reihenfolge (a,b oder b,a)
  - <xsd:choice> ... </xsd:choice> Auswahl (entweder a oder b)

XML-Schema: Komplexe Typen

```
<xsd:complexType name="authorType">
    <xsd:sequence>
        <xsd:element name="firstname" type="xsd:string" minOccurs="0"
                maxOccurs="unbounded"/>
        <xsd:element name="lastname" type="xsd:string"/>
    </xsd:sequence>
    <xsd:attribute name="age" type="xsd:string" use="optional"/>
</xsd:complexType>
```

… Grundlage für weitere Beispiele!

Typhierarchien

- Gesetzmäßigkeit zwischen zwei Typen
- Typdefinition durch
  - Erweiterung (engl. extension) oder
  - Restriktion (engl. restriction) einer bestehenden Typdefinition
- Alle Typen in XML-Schema sind entweder
  - Atomare Typen (z.B. string) oder
  - Erweiterung bzw. Restriktion bestehender Typen
- Alle Typen bilden eine Typhierarchie
  - Baum mit Wurzel: Typ Zeichenkette
  - Keine Mehrfachvererbung

Erweiterung von Typen

- Typen können konstruktiv um weitere Elemente oder Attribute zu neuen Typen erweitert werden
- Beispiel:

```
<xsd:complexType name="extendedAuthorType">
        <xsd:extension base="authorType">
            <xsd:sequence>
                <xsd:element name="email" type="xsd:string" minOccurs="0" maxOccurs="1"/>
            </xsd:sequence>
            <xsd:attribute name="homepage" type="xsd:string" use="optional"/>
        </xsd:extension>
</xsd:complexType>
```

- Erweitert den zuvor definierten Typ authorType um
  - Ein optionales Element email
  - Ein optionales Attribut homepage

Erweiterung von Typen (2)

Die Erweiterungen werden an die bestehenden Definitionen angehängt:

```
<xsd:complexType name="extendedAuthorType">
    <xsd:extension base="authorType">
        <xsd:sequence>
            <xsd:element name="email" type="xsd:string" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
        <attribute name="homepage" type="xsd:string" use="optional"/>
    </xsd:extension>
</xsd:complexType>
```

```
<xsd:complexType name="extendedAuthorType">
<xsd:sequence>
    <xsd:element name="firstname" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="lastname" type="xsd:string"/>
    <element name="email" type="xsd:string" minOccurs="0" maxOccurs="1"/>
</xsd:sequence>
<xsd:attribute name="age" type="xsd:string" use="optional"/>
<attribute name="homepage" type="xsd:string" use="optional"/>
</xsd:complexType>
```

Restriktion von Typen

- Typen werden durch Verschärfung von Zusatzangaben bei Typdefinitionen in ihrer Wertemenge eingeschränkt
- Beispiele für Restriktionen:
  - Bisher nicht angegebene type-, default- oder fixed-Attribute
  - Verschärfung der Kardinalitäten minOccurs, maxOccurs
- Substituierbarkeit
  - Menge der Instanzen des eingeschränkten Untertyps muss immer eine Teilmenge des Obertyps sein!
- Restriktion komplexer Typen
  - Struktur bleibt gleich: es dürfen keine Elemente oder Attribute weggelassen werden
- Restriktion einfacher Typen
  - Restriktion ist (im Gegensatz zur Erweiterung) auch bei einfachen Typen erlaubt

Restriktion von Typen (2)

- Beispiel (Komplexer Typ):
-

```
<xsd:complexType name="restrictedAuthorType">
    <xsd:restriction base="authorType">
        <xsd:sequence>
            <xsd:element name="firstname" type="xsd:string" minOccurs="0"
                maxOccurs="2"/>
            <xsd:element name="lastname" type="xsd:string"/>
        </xsd:sequence>
        <xsd:attribute name="age" type="xsd:string" use="required"/>
    </xsd:restriction>
</xsd:complexType>
```

- Gegenüber dem ursprünglichen Typ wurde die Anzahl der Vornamen (firstname) auf 2 begrenzt und das Altersattribut (age) erzwungen

Bewertung von XML-Schema

- Mit XML-Schema kann viel mehr Semantik in einem Schema eingefangen werden als mit DTDs
- XML-Schema immer weitläufiger unterstützt (Tendenz steigend) und lösen DTDs sukzessive ab
- Syntax und Ausdruckskraft von XML-Schema sind sehr umfangreich
  - Schwäche: geringe Vielfalt bei Konsistenzbedingungen (in der Vorlesung nicht behandelt)
- Mehr zu XML-Schema im Web:
  - http://www.w3.org/TR/xmlschema-0/ Einführung
  - http://www.w3.org/TR/xmlschema-1/ Teil I: Strukturen
  - http://www.w3.org/TR/xmlschema-2/ Teil II: Datentypen