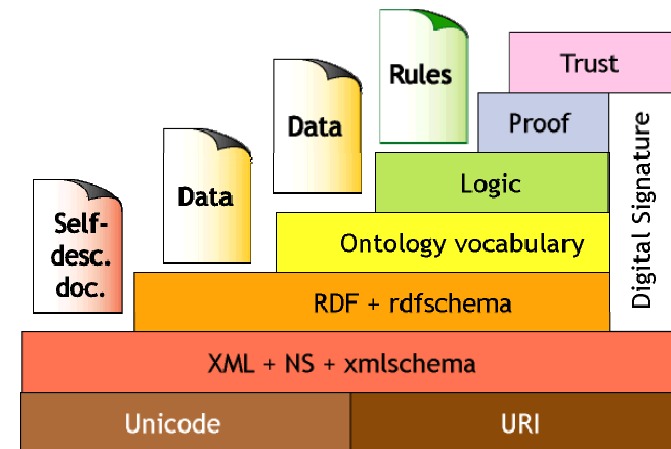


Intelligente Systeme im WWW: Semantic Web

XML – Extensible Markup Language

York Sure, Pascal Hitzler, Anupriya Ankolekar
Institut AIFB, Universität Karlsruhe

Semantic Web Schichtenmodell



Slide 2

Übersicht

- **XML – eXtensible Markup Language**
 - XML Basis-Wissen
 - Strukturierung mit DTDs und XML Schema
 - Namespaces

XML

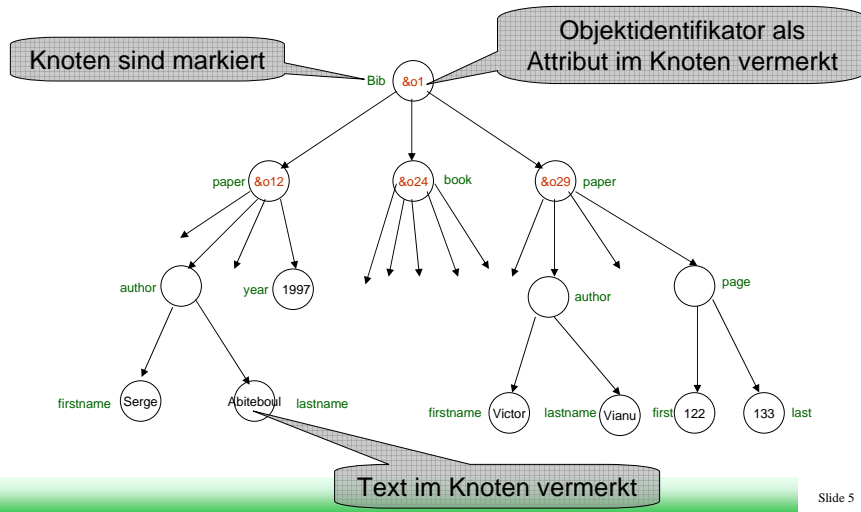
- eXtensible Markup Language
- Ursprung: strukturierter Text (HTML4.0 \in XML \subset SGML)
- Web-Standard (W3C) zum Datenaustausch:
 - Ein- und Ausgabedaten von Anwendungen können mittels XML beschrieben werden
 - Industrie muß sich nur noch auf standardisierte Beschreibung einigen
- Komplementärsprache zu HTML:
 - HTML beschreibt die Präsentation
 - XML beschreibt den Inhalt
- Datenbank-Sichtweise: XML als Datenmodell für semistrukturierte Daten

Slide 3

Slide 4

XML-Modell

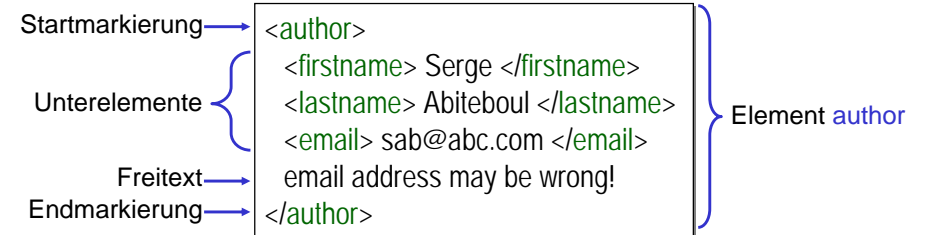
Veranschaulichung von Objekten als gerichtete Graphen



Slide 5

XML-Syntax (1) – XML-Element

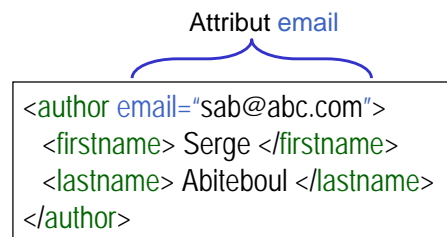
- XML-Element (engl. element):
 - Beschreibung eines Objekts, die durch passende Markierungen (tags) wie `<author>` und `</author>` geklammert ist
 - Inhalt eines Elements: Text und/oder weitere Elemente (Unterelemente)
 - Elemente können beliebig geschachtelt sein
 - Leere Elemente: `<year></year>` kurz: `<year/>`



Slide 6

XML-Syntax (2) – XML-Attribut

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

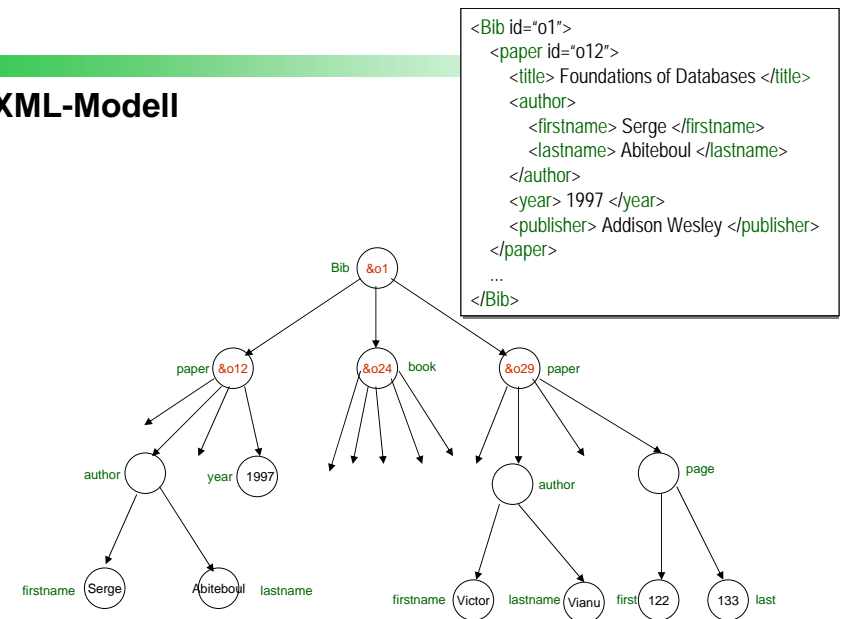


Weitere denkbare Beschreibung derselben Daten:

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

Slide 7

XML-Modell



Slide 8

XML vs HTML

- **HTML:** feste **Bezeichner** (tag) und **Semantik** (die Darstellung von Text)
- **XML:** freie **Bezeichner** zur Beschreibung von anwendungsspezifischer Syntax (Meta-Grammatik)
- XML \subset 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

Slide 9

XML-Dokument

- **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 Wurzelement
 - 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)

Slide 10

Schemata in XML (Optional !)

- **DTD – Document Type Definitions:**
 - Einfache Grammatik für ein XML-Dokument
 - **Deklaration von Elementen, Attributen, u.a.**
 - **Beschränkt die beliebige Verschachtelung von Elementen und Attributen**
 - Ist Teil des XML-Standards
 - Erbe von SGML
- **XML-Schema:**
 - Komplexere Datendefinitionssprache:
 - **Viele standardisierte Basistypen, z.B. float, double, decimal, boolean**
 - **Typen und typisierte Objektreferenzen**
 - **Klassenhierarchien / Vererbung**
 - **Konsistenzbedingungen**
 - Standard (sog. „W3C Recommendation“) in Ergänzung zu XML
 - Abwärtskompatibel zu DTD

Slide 11

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

DTD – Deklaration von Elementen (2)

- 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

Einleitung und Festlegung des Wurzelements 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

DTD – Deklaration von Elementen (3)

- 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

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

DTD – Deklaration von Elementen (4)

- 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

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

DTD – Deklaration von Elementen (5)

- 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

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

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

DTD – Deklaration von Attributen (2)

- 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

DTD – Deklaration von Attributen (3)

- 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

DTD – OIDs und Referenzen

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

```
<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

- Zur Erinnerung: DTDs definieren kontextfreie **Grammatiken**
 - Rekursive Definitionen sind möglich

```
<!DOCTYPE paper [
  <!ELEMENT paper (section*)>
  <!ELEMENT section ((title, section*)|text)>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT text (#PCDATA)>
]>
```

DTD

- 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

```
<schema>
  <element name="bib">
    <complexType>
      <element name="paper" minOccurs="0" maxOccurs="unbounded">
        <complexType>
          <attribute name="id" type="ID" use="required"/>
          <sequence>
            <element name="author" type="authorType"
              maxOccurs="unbounded"/>
            <element name="year" type="string"/>
            <element name="publisher" type="string" minOccurs="0"/>
          </sequence>
        </complexType>
      </element>
    </complexType>
  </element>
</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: <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"** **y ∈ { 1, n, unbounded }**
 - Wertvorgaben (schließen sich gegenseitig aus!):
 - **default = "v"** **veränderliche Vorgabe**
 - **fixed = "u"** **unveränderliche Vorgabe**
- Beispiele:
 - <element name="bib"/>
 - <element name="paper" minOccurs="0" maxOccurs="unbounded"/>
 - <element name="publisher" type="string" minOccurs="0"/>

XML-Schema: Attribute

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

Slide 25

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

Slide 26

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`, ...
- Beispiele:
 - `<element name="year" type="year"/>`
 - `<element name="pages" type="positiveInteger"/>`
 - `<attribute name="age" type="unsignedShort"/>`

Slide 27

XML-Schema: Einfache Typen

- Zusätzlich können von bestehenden Typen noch weitere, sog. einfache Typen, abgeleitet werden:
 - Typdefinition:


```
<simpleType name="humanAge" base="unsignedShort">
  <maxInclusive value="200"/> </simpleType>
```
 - Dokumentdefinition:


```
<attribute name="age" type="humanAge"/>
```
- Solche einfachen Typen dürfen jedoch keine verschachtelten Elemente enthalten!
- In ähnlicher Weise können Listen definiert werden:
 - Typdefinition:


```
<simpleType name="authorType" base="string"
  derivedBy="list"/>
```

 (Name eines Autors als mit Leerzeichen getrennte Liste von Zeichenketten)
 - Dokumentdefinition:


```
<element name="author" type="authorType"/>
```

Slide 28

XML-Schema: Komplexe Typen

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


```
<complexType name="authorType">
  <sequence>
    <element name="firstname" type="string" minOccurs="0"
      maxOccurs="unbounded"/>
    <element name="lastname" type="string"/>
  </sequence>
  <attribute name="age" type="string" use="optional"/>
</complexType>
```
- Gruppierungs-Bezeichner:
 - <sequence> ... </sequence> Feste Reihenfolge (a,b)
 - <all> ... </all> Beliebige Reihenfolge (a,b oder b,a)
 - <choice> ... </choice> Auswahl (entweder a oder b)

Slide 29

XML-Schema: Komplexe Typen

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

... Grundlage für weitere Beispiele!

Slide 30

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
- Typen sind entlang der Typhierarchie **abwärtskompatibel:**
 - Für Typinstanzen gilt das Substituierbarkeitsprinzip
 - Elemente eines bestimmten Typs akzeptieren auch Daten einer Erweiterung oder Restriktion des geforderten Typs

Slide 31

Typhierarchien: Erweiterung von Typen

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


```
<complexType name="extendedAuthorType">
  <extension base="authorType">
    <sequence>
      <element name="email" type="string" minOccurs="0"
        maxOccurs="1"/>
    </sequence>
    <attribute name="homepage" type="string" use="optional"/>
  </extension>
</complexType>
```
- Erweitert den zuvor definierten Typ authorType um
 - Ein optionales Element **email**
 - Ein optionales Attribut **homepage**

Slide 32

Typhierarchien: Erweiterung von Typen (2)

- Die Erweiterungen werden an die bestehenden Definitionen angehängt:

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

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

Slide 33

Typhierarchien: 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

Slide 34

Typhierarchien: Restriktion von Typen (2)

- Beispiel (Komplexer Typ):

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

Vorher: maxOccurs="unbounded"

Vorher: use="optional"

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

Slide 35

Beispiel-Schema (Hausaufgabe)

```
<!-- XMLSchema für eine Literaturdatenbank -->
<schema>

  <!-- Globales Wurzelement bib -->
  <element name="bib">
    <complexType>
      <element name="paper" minOccurs="0" maxOccurs="unbounded">
        <complexType>
          <attribute name="id" type="ID" use="required"/>
          <sequence>
            <element name="author" type="authorType" maxOccurs="unbounded"/>
            <element name="year" type="string"/>
            <element name="publisher" type="string" minOccurs="0"/>
            <element name="references" type="listOfPapers" minOccurs="0"/>
          </sequence>
        </complexType>
      </element>
    </complexType>
  </element>

  <!-- Liste von Verweisen auf Papiere (siehe bib/paper/@references) -->
  <simpleType name="listOfPapers" base="ID" derivedBy="list"/>
```

Slide 36

Beispiel-Schema (2)

```
<!-- Reine Typdefinitionen -->
<complexType name="authorType">
  <sequence>
    <element name="firstname" type="string" minOccurs="0" maxOccurs="unbounded"/>
    <element name="lastname" type="string"/>
  </sequence>
  <attribute name="age" type="humanAge" use="optional"/>
</complexType>

<simpleType name="humanAge" base="unsignedShort">
  <maxInclusive value="200"/> </simpleType>

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

<simpleType name="simpleURLType" base="string">
  <pattern value="http://([^\?#]*)?([^\?#]*)\?([^\?#]*)?([^\?#]*)?"/> </simpleType>
```

Bewertung von XML-Schema

- Mit XML-Schema können Datenbasis-Schemata viel einfacher als mit DTDs spezifiziert werden
 - Es 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

XML Namespaces: 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 Namespaces

- 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
- Namespaces sind eigentlich eine Schicht on Top von XML 1.0 (derzeit ex. bereits Arbeiten an XML 1.1), da *prefix:name* ist wiederum ein gültiger Tag ist
- *Namespace Bindings* werden von manchen Werkzeugen ignoriert, sog. "flache Namespaces"

Namespace Bindings

- 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 Namespaces

```
<address>
  <name>Xaver M. Linde</name>
  <street>Wikingerufer 7</street>
  <town>10555 Berlin</town>
  <bill>12.50</bill>
  <phone>030/1234567</phone>
  <phone>030/1234568</phone>
  <fax>030/1234569</fax>
  <bill>76.20</bill>
</ address>
```

*bill ist mehrdeutig
verwendeter
Tagname*

Beispiel: Zwei verschiedene Namespaces

```
<mail:address xmlns:mail="http://www.deutschepost.de/"
  xmlns:tele="http://www.telekom.de/">
  <mail:name>Xaver M. Linde</mail:name>
  <mail:street>Wikingerufer 7</mail:street>
  <mail:town>10555 Berlin</mail:town>
  <mail:bill>12.50</mail:bill>
  <tele:phone>030/1234567</tele:phone>
  <tele:phone>030/1234568</tele:phone>
  <tele:fax>030/1234569</tele:fax>
  <tele:bill>76.20</tele:bill>
</ mail:address>
```

*bill wurde
disambiguiert
durch Verwendung
der Prefixe mail
und tele*

- Das Wurzel-Element `mail:address` sowie die Kind-Elemente `mail:name`, `mail:street`, `mail:town`, und `mail:bill` benutzen das Prefix `mail` welches an die URI deutschepost gebunden ist
- Die Kind-Knoten `tele:phone`, `tele:fax`, und `tele:bill` nutzen das Prefix `tele` welches an die URI telekom gebunden ist

Semantic Web Schichtenmodell

