

Extensible Markup Language

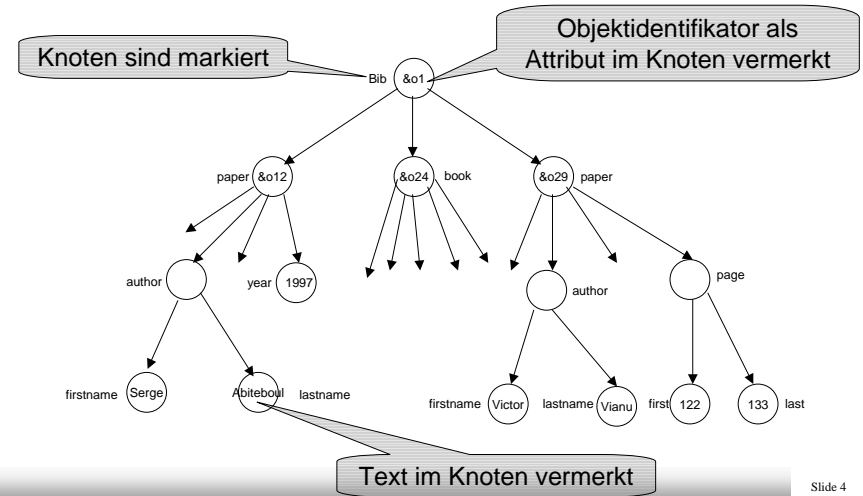


Purpose here: storing and exchanging knowledge

Non-purpose here: structuring documents

XML-Modell

Veranschaulichung von Objekten als gerichtete Graphen



Intelligent Systems on the World Wide Web

In-Depth XML

Lecture Slides
Steffen Staab & Raphael Volz
Institute for Applied Computer Science and Formal
Description Methods (AIFB)
Karlsruhe University

XML

- eXtended 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

XML-Syntax (2) – XML-Attribut

- XML-Attribut (engl. attribute):
 - Name-Zeichenkettenwert-Paar
 - 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: feste Bezeichner (tag) und Semantik (die Darstellung von Text)
- XML: freie Bezeichner zur Beschreibung von anwendungsspezifischer Syntax (Meta-Grammatik)
- XML<SGML

```

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

HTML

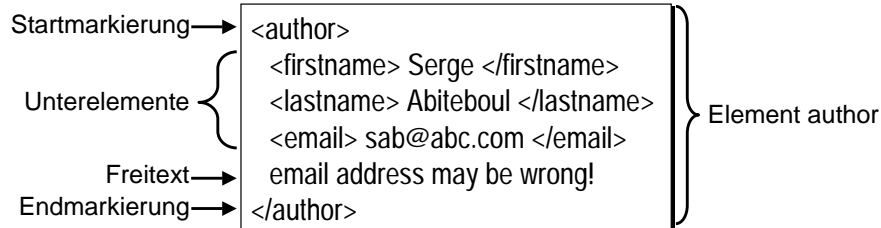
```

    <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

XML-Syntax (1) – XML-Element

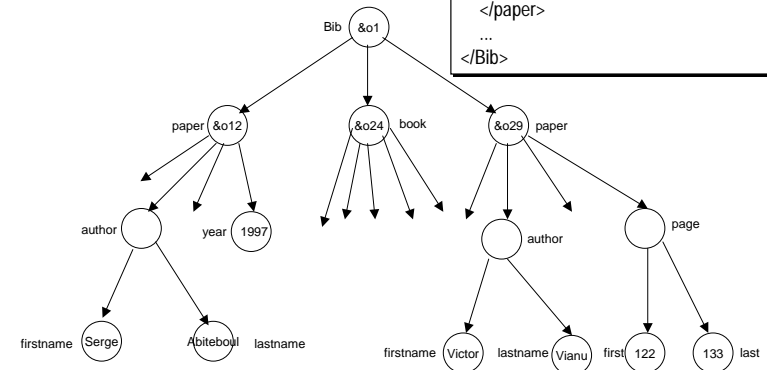
- XML-Element (engl. element):
 - Beschreibung eines Objekts, die durch passende Markierungen (tags) wie <author> und </author> geklammert ist
 - Inhalt eines Elements: Text und/oder weitere Elemente (Unterelemente)
 - Elemente können beliebig geschachtelt sein
 - Leere Elemente: <year></year> kurz: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



XML-Modell

```

    <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>
    
```



Structuring XML

Structuring

Slide 10

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

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 9

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 in Ergänzung zu XML (noch nicht verabschiedet!)
 - Abwärtskompatibel zu DTD

Slide 11

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 (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

- 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 (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 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 (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 – 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 (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

Bewertung von DTDs

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

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

DTD

XML-Schema: Elemente

- Syntax:


```
<element name="Name"/>
```
- Optionale Zusatzattribute:
 - Typ
 - type = "Typ" Typname** **atomarer, einfacher oder komplexer**
 - 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"/>
```

## 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**

## 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: 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: 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: Attribute

- Syntax:
 

```
<attribute name="Name"/>
```
- Optionale Zusatzattribute:
  - Typ:
    - **type = "Typ"**
  - Existenz:
 

• <b>use = "optional"</b>	<b>Kardinalität [0,1]</b>
• <b>use = "required"</b>	<b>Kardinalität [1,1]</b>
  - Vorgabewerte:
 

• <b>use = "default" value = "v"</b>	<b>veränderliche Vorgabe</b>
• <b>use = "fixed" value = "u"</b>	<b>unveränderliche Vorgabe</b>
- 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: 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: 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>
```

Slide 30

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

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

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: 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

Polymorphe Konsistenzbedingungen

- Elemente und Attribute können zusätzlich mit Konsistenzbedingungen belegt werden
- Eindeutigkeit (Schlüsselkandidat):


```
<unique name="eindeutigerAutorenName">
  <field xpath="bib/paper/author/@firstname"/>
  <field xpath="bib/paper/author/@lastname"/>
</unique>
```

 Die Kombinationen von Vor- und Nachnamen bei den Autoren sind immer eindeutig
 - Mittels field werden die entsprechenden Elemente oder Attribute identifiziert
 - Mittels xpath wird der genaue Pfadausdruck angegeben, unter dem das entsprechende field innerhalb der Dokumenthierarchie gefunden werden kann
 - XPath (xpath): XML-Standard für Pfadausdrücke

Slide 36

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 (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>
```

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

Slide 35

Beispiel-Schema

```
<!-- 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 38

Beispiel-Schema (3)

```
<!-- Konsistenzbedingungen -->

<unique name="eindeutigerAutorenName">
  <field xpath="bib/paper/author/@firstname"/>
  <field xpath="bib/paper/author/@lastname"/>
</unique>

<key name="papierSchlüssel">
  <field xpath="bib/paper/@id"/>
</key>

<keyref name="papierFremdschlüssel" refer="papierSchlüssel">
  <field xpath="bib/paper/@references"/>
</keyref>

</schema>
```

Slide 40

Polymorphe Konsistenzbedingungen (2)

- Schlüsselbedingung:

```
<key name="papierSchlüssel">
  <field xpath="bib/paper/@id"/>
</key>
```

Das Attribut id in paper dient als Schlüssel

- Schlüssel sind eindeutig und können referenziert werden

- Fremdschlüsselbedingung:

```
<keyref name="papierFremdschlüssel" refer="papierSchlüssel">
  <field xpath="bib/paper/@references"/>
</keyref>
```

Ergänzend zum bisherigen Schema: references ist eine Liste von Papieren, die vom Papier aus referenziert werden, d.h. in dessen Literaturverzeichnis auftauchen.

- Erst jetzt macht name ein Sinn: mit refer bezieht man sich auf das name-Attribut einer Schlüsselbedingung, nicht auf das Schlüsselfeld!
- Die Werte in references müssen also immer unter den Schlüsseln zu den Papieren zu finden sein

Slide 37

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>
```

XML Namespaces

Namespaces

Slide 42

Namespace Bindings

- Prefixes are bound to namespace URIs by attaching an `xmlns:prefix` attribute to the prefixed element or one of its ancestors, `prefix:name1, ..., prefix:namen`
- The value of the `xmlns:prefix` attribute is a URI, which may or (unlike for DTDs!) may not point to a description of the namespace's syntax
- An element can use bindings for multiple name-spaces via attributes `xmlns:prefix1, ..., xmlns:prefixm`

Slide 44

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 wird jedoch noch nicht weitläufig unterstützt, während DTDs bereits zum Kernstandard von XML gehören
- Syntax und Ausdruckskraft von XML-Schema sind sehr umfangreich
 - Einzige Schwäche: geringe Vielfalt bei Konsistenzbedingungen
- 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

Slide 41

XML Namespaces and Programming-Language Modules

- XML namespaces are akin to namespaces, packages, and modules in programming languages
- Disambiguation of tag names from different XML applications ("spaces") through different prefixes
- A *prefix* is separated from the local *name* by a ":", obtaining `prefix:name` tags
- Namespaces constitute a layer on top of XML 1.0, since `prefix:name` is again a valid tag name and namespace bindings are ignored by some tools

Slide 43