Freie Universität Berlin

DTDs und XML-Schemata

Heutige Vorlesung

Freie Universität Berlin

letzte Woche

☑ XML-Syntax

☑ Namensräume

heute

- Definition von XML-Sprachen
- DTDs und XML-Schema anhand eines einheitlichen Beispiels

nächste Woche

- XML-Schema im Detail

Motivation

Freie Universität Berlin

```
<book>
   <title>My Life and Times</title>
   <authors>
     <author>
       <first>Paul</first>
       <last>McCartney</last>
     </author>
   </authors>
   <date>
     <year>1998</year>
     <month>July</month>
   </date>
   <isbn>94303-12021-43892</isbn>
   <publisher>McMillin Publishing</publisher>
</book>
```

oder so?

so?

```
<Book>
   <Title>My Life and Times</Title>
   <Author>Paul McCartney</Author>
   <Date>July, 1998</Date>
   <ISBN>94303-12021-43892</ISBN>
   <Publisher>McMillin
   Publishing</Publisher>
</Book>
```

- ⇨ einheitliches Format nötig
- ⇨ Format sollte durch XML-Prozessor validierbar sein

Spezifische Formate

- prinzipieller Aufbau von Dokumenten: Welche Elemente/Attribute dürfen wo verwendet werden?
- Datentypen der Inhalte: Welche Inhalte sind erlaubt?

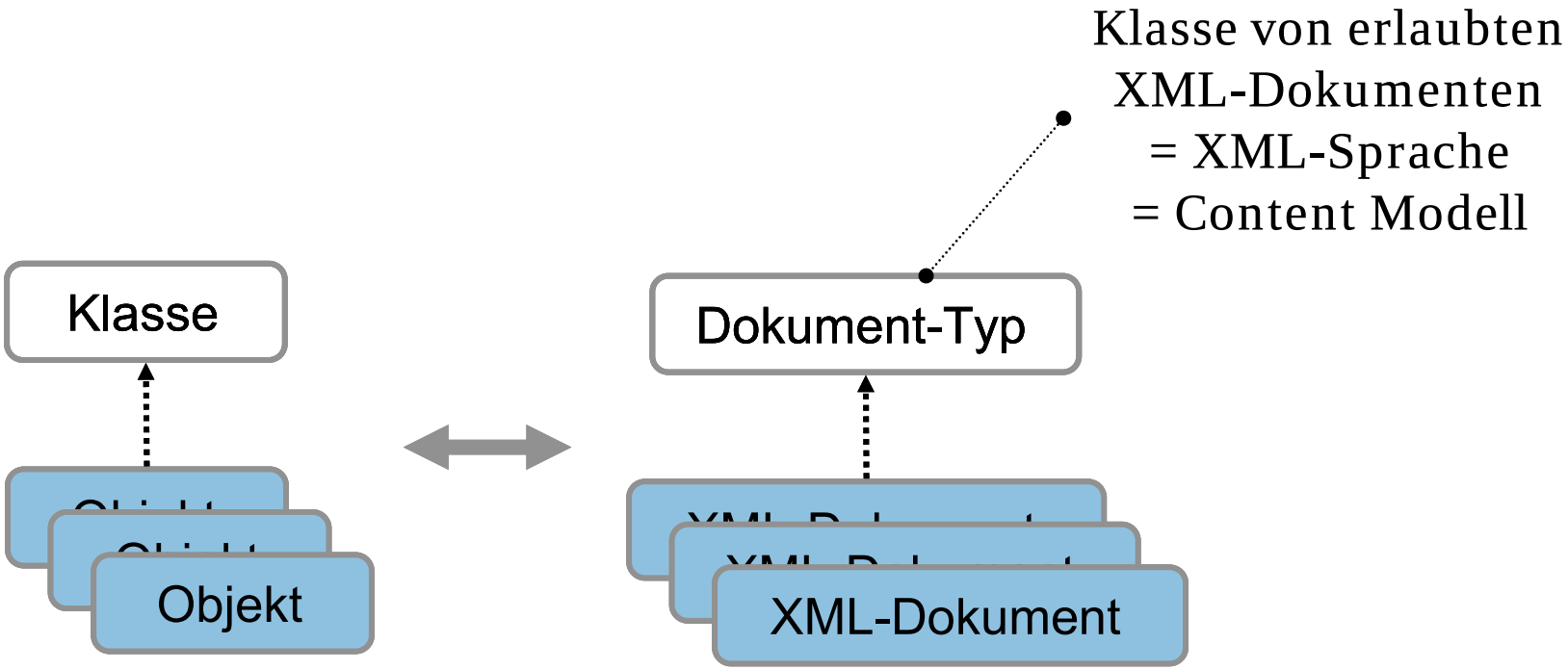
```
<Book>
   <Title> PCDATA </Title>
   <Author> PCDATA </Author>
   <Date> PCDATA </Date>
   <ISBN> PCDATA </ISBN>
   <Publisher> PCDATA
   </Publisher>
</Book>
```

- konkrete Inhalte werden nicht beschrieben

⇨ Klasse von erlaubten XML-Dokumenten

- auch Dokument-Typ, XML-Sprache, Anwendung von XML oder Content Modelle genannt

Dokument-Typ



- Dokument-Typ kann mit einer DTD, einem XML-Schema oder ähnlichen Formalismen definiert werden.

DTDs vs. XML-Schema

Freie Universität Berlin

DTD‘s	XML-Schema
vereinfachte SGML-DTD, Teil von XML 1.0/1.1	eigener W3C-Standard
eigene Syntax	XML-Schemta = XML-Sprache
kompakter und lesbarer	ausdrucksstärker
zur Beschreibung von Text-Dokumenten ausreichend	zur Beschreibung von Daten besser geeignet.

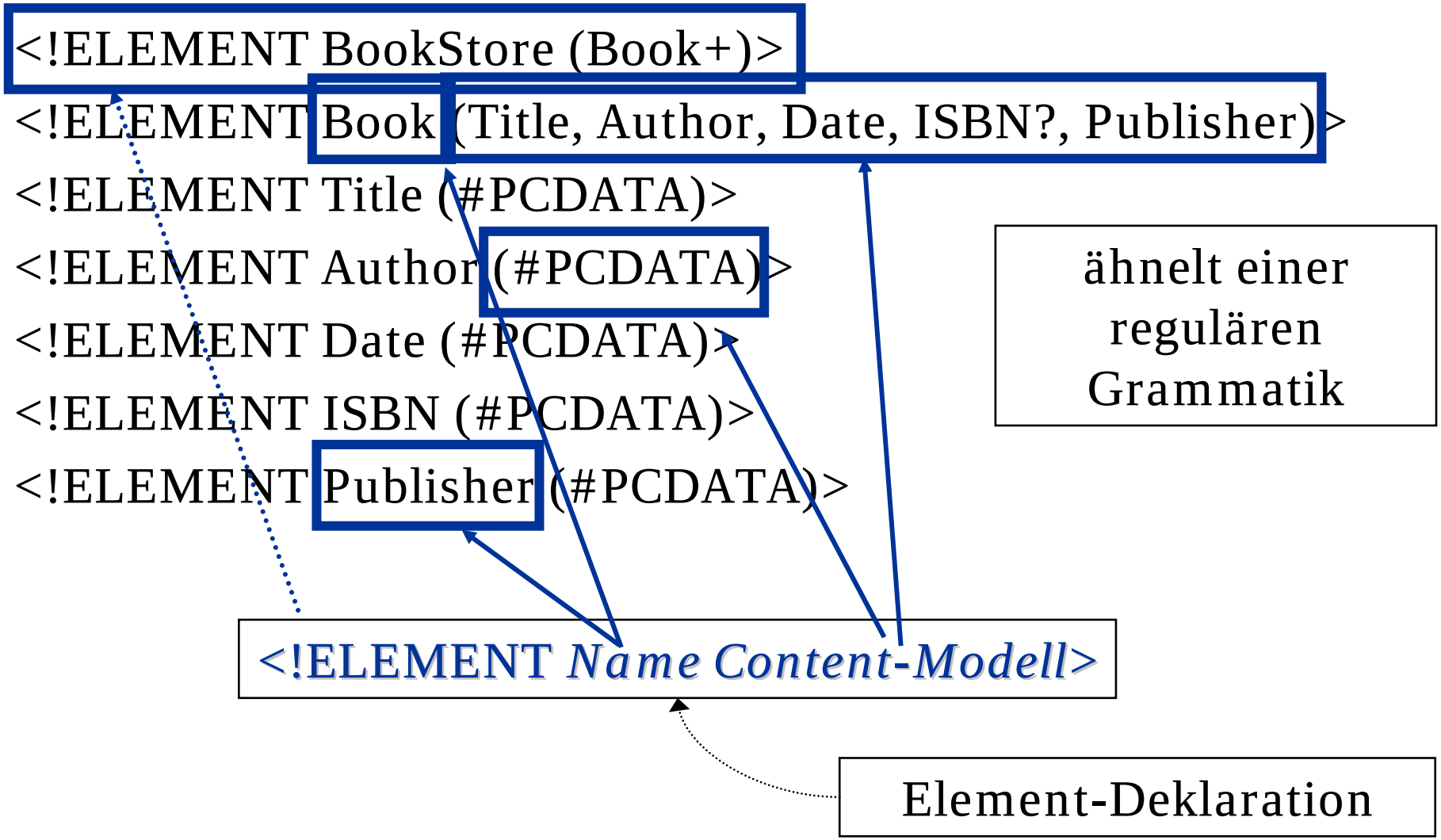
Document Type Definitions (DTDs)

Wie sieht eine DTD hierfür aus?

```
<BookStore>
  <Book>
      <Title>My Life and Times</Title>
      <Author>Paul McCartney</Author>
      <Date>July, 1998</Date>
      <ISBN>94303-12021-43892</ISBN>
      <Publisher>McMillin Publishing</Publisher>
  </Book>
</BookStore>
```

- BookStore soll mindestens ein Buch enthalten.
- ISBN optional
- alle anderen Kind-Elemente obligatorisch

Die DTD für das Beispiel-Dokument



DTDs:

Element-Deklaration

Datentypen für Element-Inhalte

Freie Universität Berlin

verschiedene Datentypen:

1. Element: Element mit speziellen Symbolen + * | ?
2. #PCDATA: unstrukturierter Inhalt ohne reservierte Symbole < und &.

   <!ELEMENT Title (#PCDATA)>

2. EMPTY: leerer Inhalt, Element kann aber Attribute haben

   <!ELEMENT br EMPTY> ➔

3. ANY: beliebiger Inhalt (strukturiert, unstrukturiert, gemischt oder leer)

   <!ELEMENT title ANY>

Datentypen wie INTEGER oder FLOAT stehen nicht zur Verfügung.

Deklaration von BookStore

<!ELEMENT BookStore (Book+)>

```
<BookStore>
   <Book>...</Book>
   <Book>...</Book>
</BookStore>
```

- + bezeichnet n Wiederholungen mit n > 0.
- * bezeichnet n Wiederholungen mit n ≥ 0.

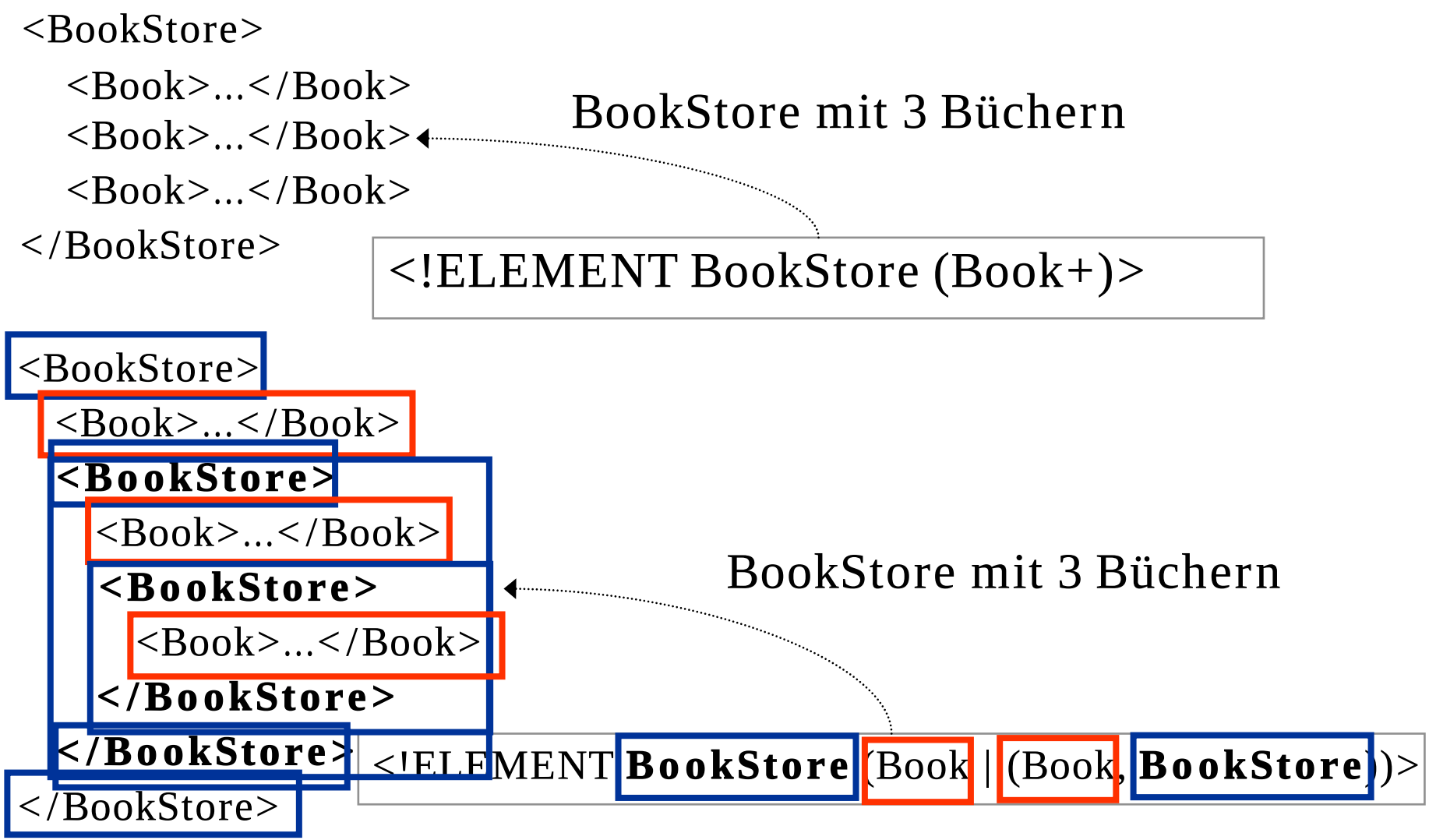
- BookStore hat mindestens ein Kind-Element Book.
- Außer Book darf BookStore keine anderen Kind-Elemente haben.

Rekursive Deklarationen

<!ELEMENT BookStore (Book | (Book, BookStore))>

- Bookstore besteht aus genau einer der folg. Alternativen:
  - genau ein Kind-Element Book
  - zwei Kind-Elemente: Book und BookStore
- | bezeichnet Auswahl: genau eine der beiden Alternativen
- , bezeichnet Sequenz von Elementen.
- Beachte: Rekursive Deklaration nicht äquivalent zur vorherigen, iterativen Definition!

Rekursive vs. iterative Deklaration



Die DTD für das Beispiel-Dokument

Freie Universität Berlin

```
<!ELEMENT BookStore (Book+)>
<!ELEMENT Book (Title, Author, Date, ISBN?, Publisher)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>
```

Deklaration von Book

<!ELEMENT Book (Title, Author, Date, ISBN?, Publisher)>

- Title, Author, Date, ISBN und Publisher (in dieser Reihenfolge) Kind-Elemente von Book.
- außer diesen keine anderen Kind-Elemente

- ? bedeutet optional

```
<Book>
   <Title>...</Title>
   <Author>...</Author>
   <Date>...</Date>
   <ISBN>...</ISBN>
   <Publisher>...</Publisher>
</Book>
```

Die DTD für das Beispiel-Dokument

Freie Universität Berlin

```
<!ELEMENT BookStore (Book+)>
<!ELEMENT Book (Title, Author, Date, ISBN?, Publisher)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>
```

Deklaration von Title etc.

<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>

<Title>My Life and Times</Title>
<Author>Paul McCartney</Author>
<Date>July, 1998</Date>
<ISBN>94303-12021-43892</ISBN>
<Publisher>McMillin Publishing</Publisher>

Verschachtelungen

- (fast) beliebige Verschachtelung von Sequenz, Auswahl |, ?, *, + und Rekursion erlaubt
- Beispiel:

```
<!ELEMENT Chap (Title, (Para | Chap)+)>
<!ELEMENT Para ANY>
<!ELEMENT Title (#PCDATA)>
```

```
<Chap>
  <Title>Kap1</Title>
  <Para>Ein Absatz</Para>
  <Chap>
    <Title>Kap1.1</Title>
    <Para>...</Para>
  </Chap>
  <Para>...</Para>
  <Chap>
    <Title>Kap1.2</Title>
    <Para>...</Para>
  </Chap>
</Chap>
```

Einschränkung der Verschachtelung

- Beispiel: ((b, c) | (b, d)) ist nicht-deterministisch
- Grund: Wenn erstes Element = b, dann kann XML-Prozessor keine der beiden Alternativen ausschließen.

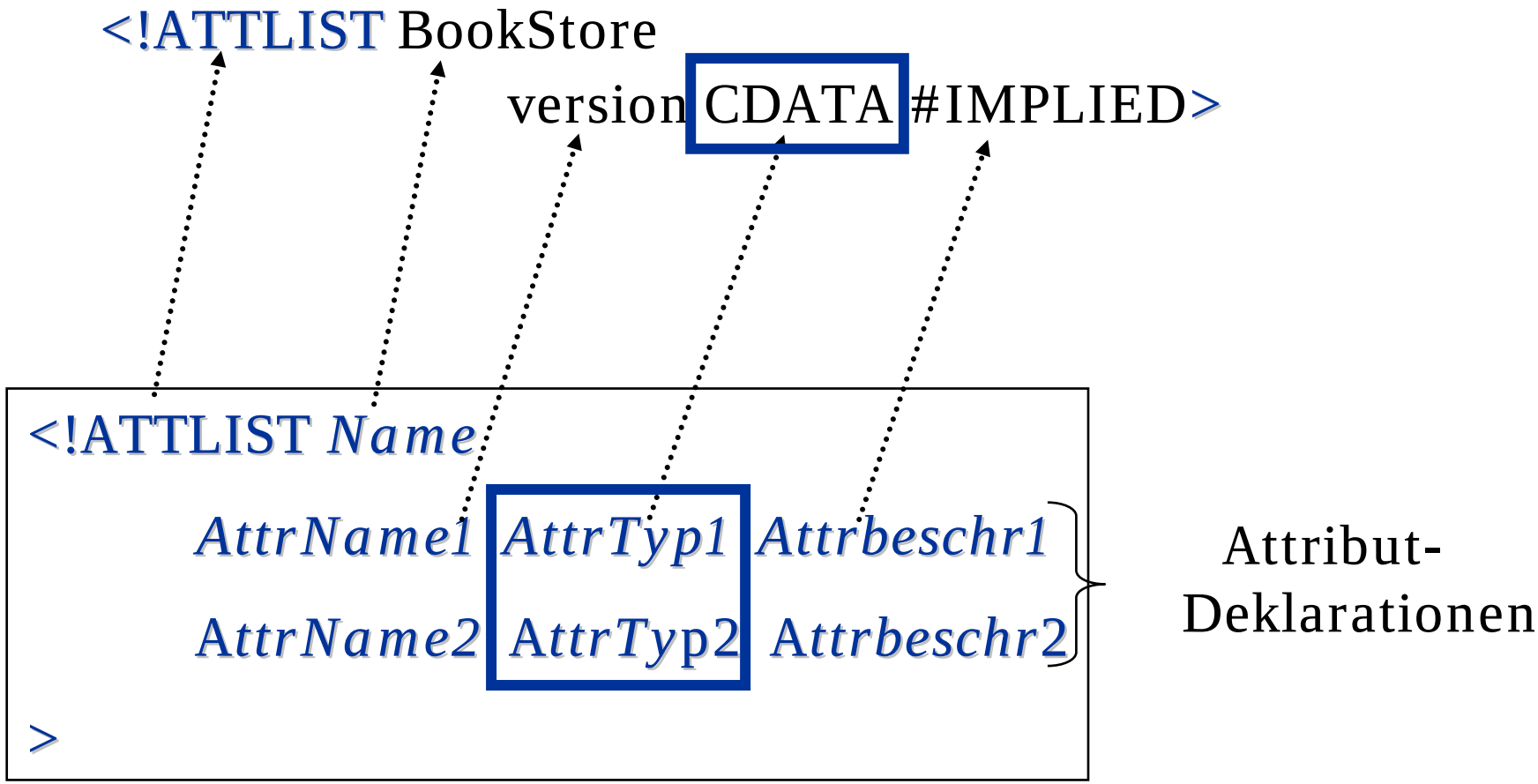
- XML erlaubt nur deterministische Content Modelle
- jedes nicht-deterministische Content Modell kann in ein äquivalentes deterministisches umgeformt werden

- Beispiel: ((b, c) | (b, d)) = (b, (c | d))

DTDs:

Attribut-Deklaration

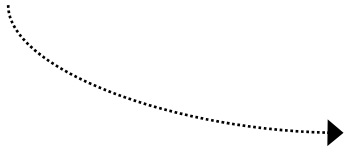
Deklaration von Attributen



Attribut-Deklarationen

Deklaration von Attributen

```
<!ATTLIST BookStore
          version CDATA #IMPLIED>
```



```
<BookStore version="1.0">
  ...
</BookStore>
```

- Element BookStore hat Attribut version.
- Außer version hat BookStore keine weiteren Attribute.
- CDATA: Attribut-Wert = String ohne <, & und ' bzw. "
- Beachte: nicht verwechseln mit <![CDATA[…]]>
- daher Entity References für <, & und ' bzw. " verwenden

Aufzählungstypen

```
<!ATTLIST Author
          gender (male | female) "female">
```

- hier statt CDATA Aufzählungstyp:
- Attribut gender hat entweder Wert male oder female.
- "female" ist Standard-Wert von gender.

Datentypen für Attribute

Freie Universität Berlin

Zusätzlich zu CDATA (Strings) und Aufzählungstypen:

- NMTOKEN: String, der Namenskonventionen von XML entspricht
- ID: eindeutiger Bezeichner, der Namenskonventionen von XML entspricht
- IDREF: Referenz auf einen eindeutigen Bezeichner

ID/IDREF

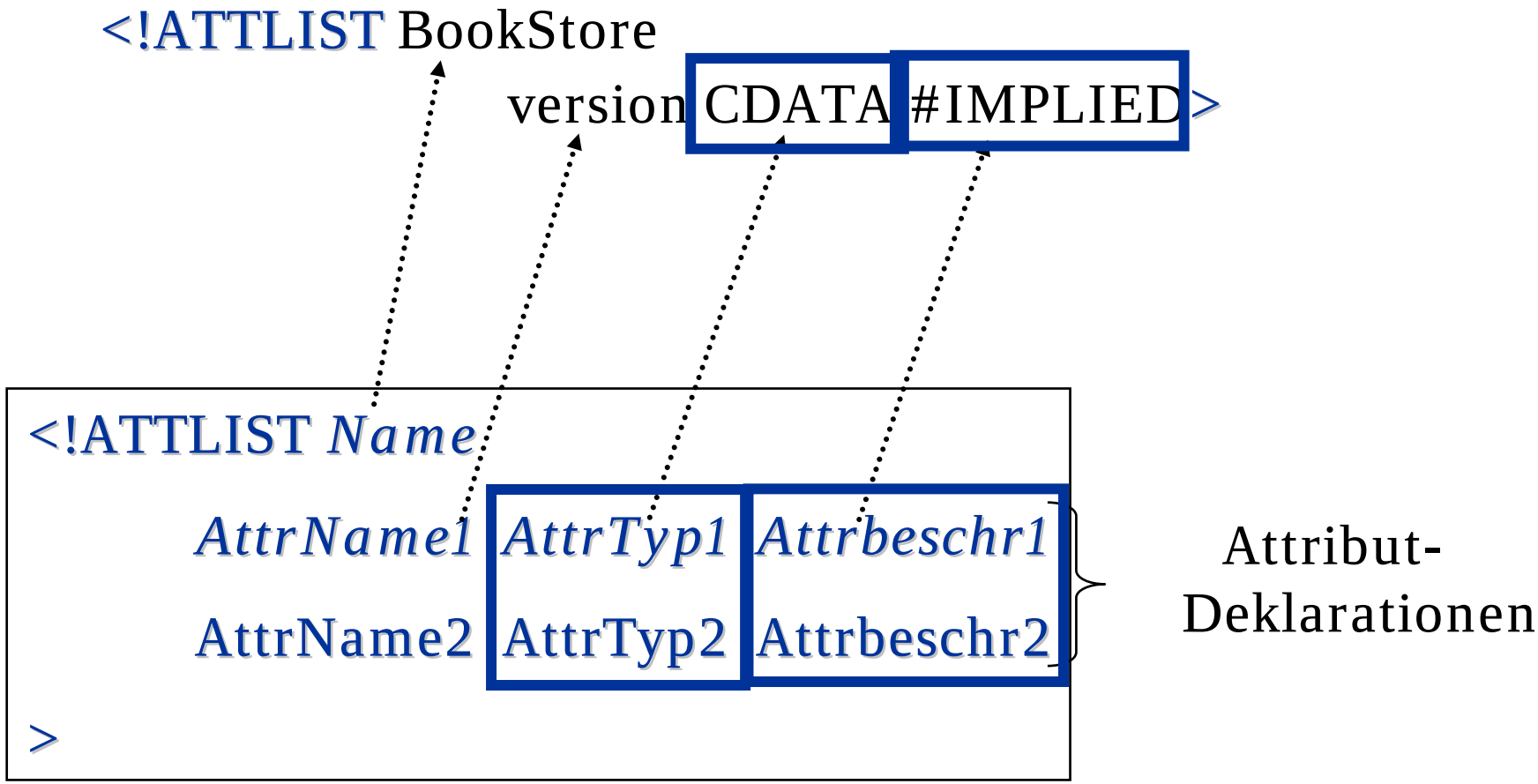
```
<!ATTLIST Author
          key ID #IMPLIED
          keyref IDREF #IMPLIED>
```

- Wert des Attributes key muss eindeutig sein:

  Zwei Attribute vom Typ ID dürfen niemals gleichen Wert haben.
- Wert des Attributes keyref muss gültige Referenz sein:

  Wert von keyref muss Wert eines Attributes vom Typ ID sein.

Deklaration von Attributen



Optionale/erforderliche Attribute

Freie Universität Berlin

```
<!ATTLIST BookStore
                    version CDATA #FIXED "1.0">
```

- #FIXED: Attribut hat immer den gleichen Wert.
- #IMPLIED: Attribut optional
- #REQUIRED: Attribut obligatorisch

- "1.0": Standard-Wert des Attributes

Beispiel

```
<BookStore>
  <Book>
      <Title>Text</Title>
      <Author key="k1">Text</Author>
      <Date>Text</Date>
      <Publisher pkey="p1">Text</Publisher>
  </Book>
  <Book>
      <Title>Text</Title>
      <Author keyref="k1"/>
      <Date>Text</Date>
      <Publisher pkey="p1">Text</Publisher>
  </Book>
</BookStore>
```

Wert k1 muss eindeutig sein: kein anderes Attribut vom Typ ID darf diesen Wert haben.

Referenz k1 muss existieren: ein Attribut vom Typ ID muss den Wert k1 haben.

Festlegung des Dokument-Typs

- direkt nach XML-Deklaration einfügen:

  - vollständige DTD oder
  - Verweis auf externe DTD

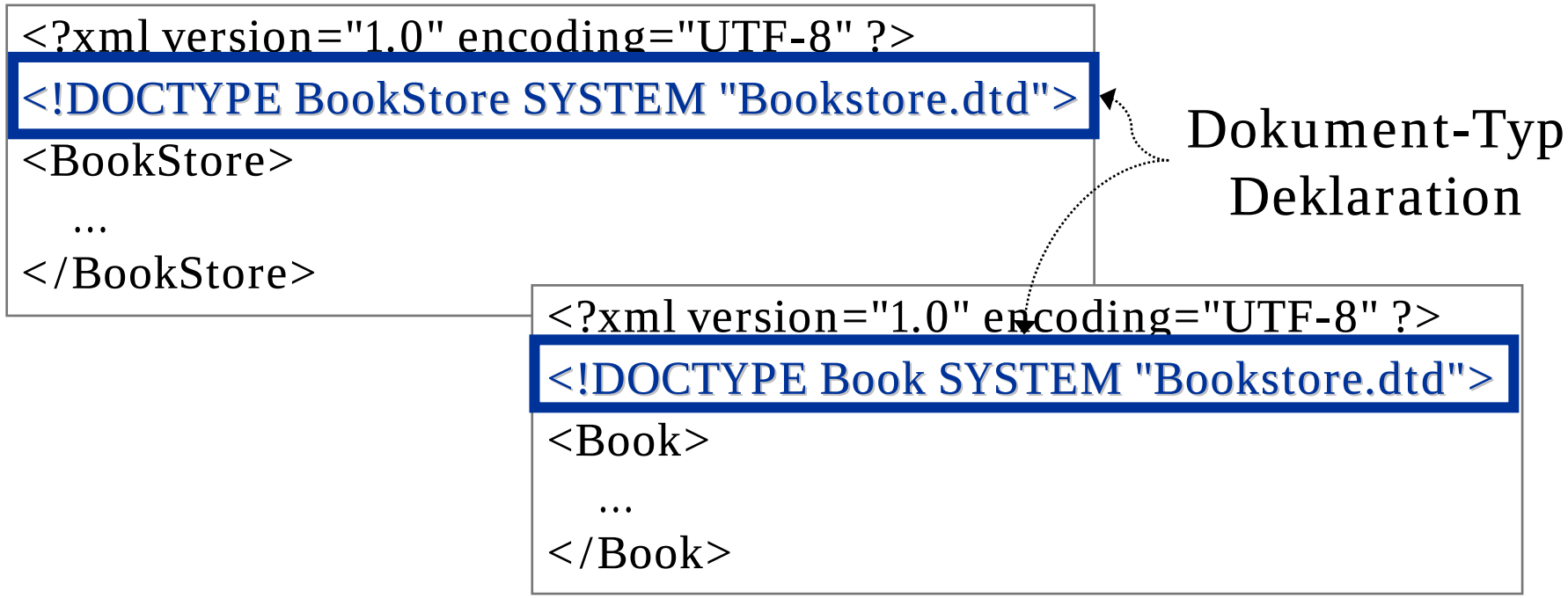
    <!DOCTYPE *Wurzel-Element* SYSTEM "*DTD*">

Interne DTD

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE BookStore [
 <!ELEMENT BookStore (Book+)>
 <!ELEMENT Book (Title, Author, Date, ISBN?, Publisher)>
 <!ELEMENT Title (#PCDATA)>
 <!ELEMENT Author (#PCDATA)>
 <!ELEMENT Date (#PCDATA)>
 <!ELEMENT ISBN (#PCDATA)>
 <!ELEMENT Publisher (#PCDATA)>
]>
 <BookStore>
   ...
 </BookStore>
```

Externe DTD

<!DOCTYPE *Wurzel-Element* SYSTEM "*DTD*">



DTDs:

Wohlgeformt & zulässig

Wohlgeformheit vs. Zulässigkeit

Freie Universität Berlin

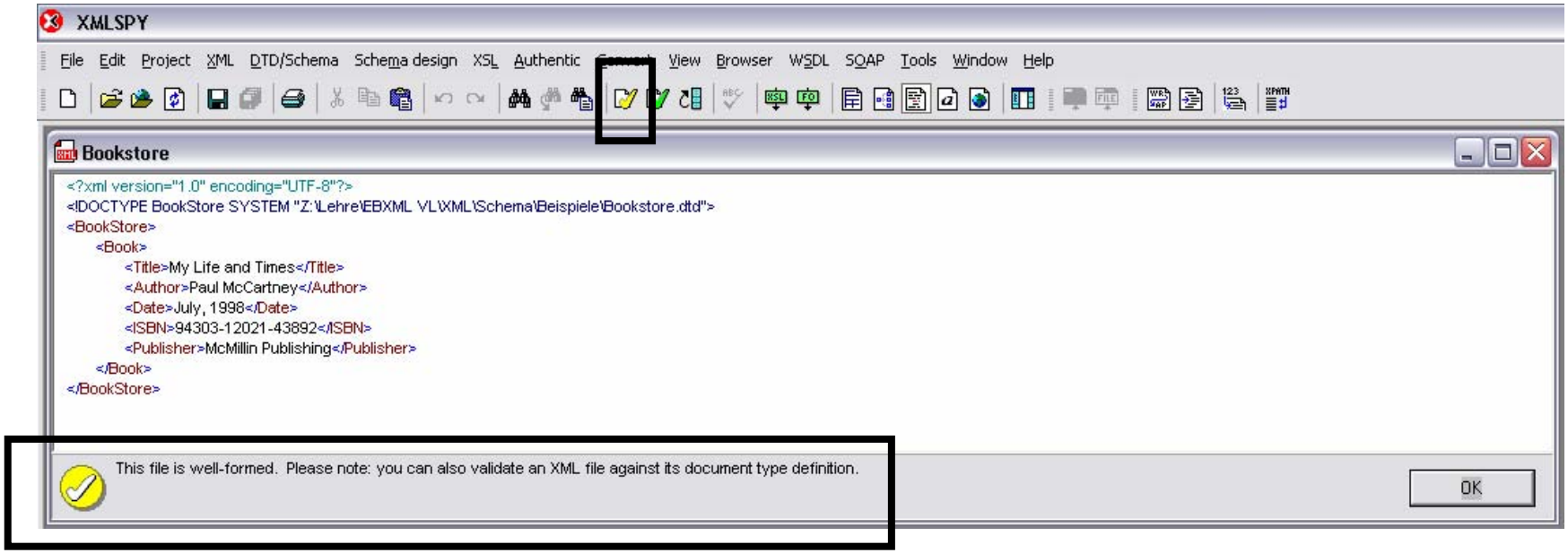
wohlgeformt (well formed)

- XML-Dokument entspricht Syntaxregeln von XML

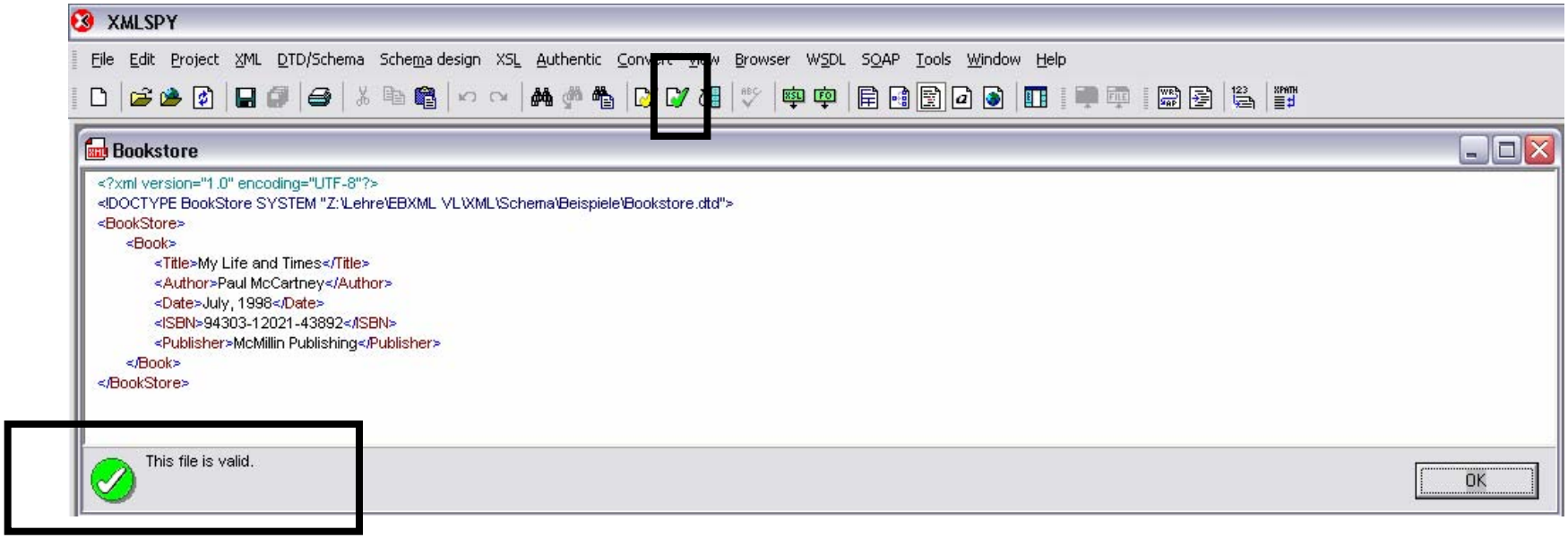
zulässig (valid) bzgl. einer DTD

1. Wurzel-Element des XML-Dokumentes ist in der DTD deklariert und
2. Wurzel-Element hat genau die Struktur, wie sie in der DTD festgelegt ist.

Prüfung der Wohlgeformtheit



Prüfung der Zulässigkeit



Nachteile von DTDs

Freie Universität Berlin

- keine XML-Syntax, eigener Parser nötig
- nur sehr wenige Datentypen, insbesondere für Element-Inhalte
- keine eigenen Datentypen definierbar
- keine Namensräume:

  DTDs können nur dann kombiniert werden, wenn es keine Namenskonflikte gibt!
- keine Vererbungshierarchien

Und noch ein Nachteil

Freie Universität Berlin

- Sequenzen einfach zu definieren:

  <!ELEMENT Book (Title, Author)>

⇨ starre Struktur in XML-Dokumenten

- soll Reihenfolge der Kind-Elemente egal sein, müssen alle Permutationen explizit aufgezählt werden:

  <!ELEMENT Book ((Title, Length) | (Length, Title))>

- nicht praktikabel: bei n Kind-Elementen n! Permutationen

DTDs Zusammenfassung

Freie Universität Berlin

DTD deklariert das erlaubte Vokabular

DTD definiert für jedes Element ein Content-Modell

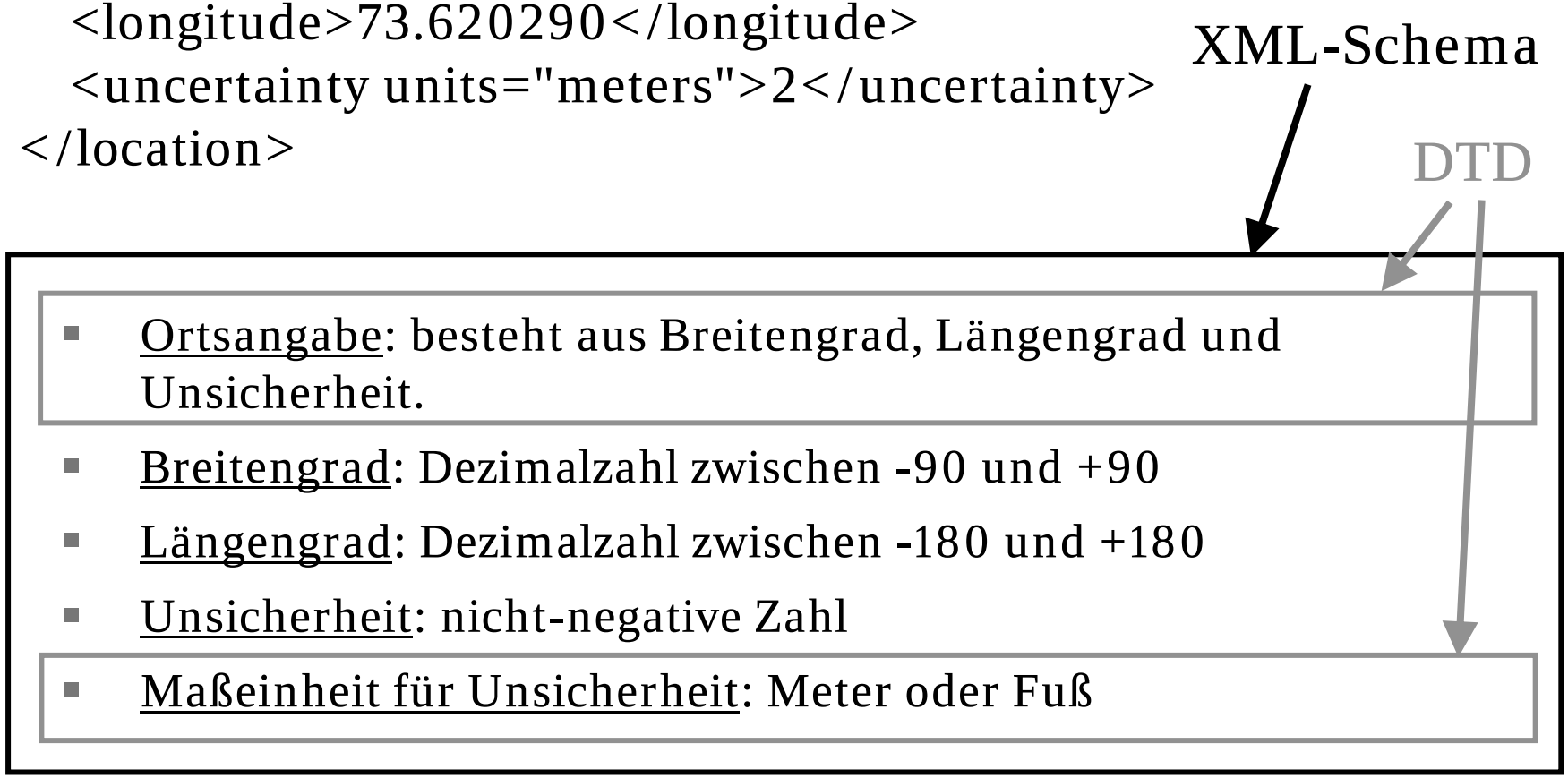
- Content-Modell legt fest:
  - Elemente oder Daten
  - Reihenfolge & Anzahl von Elementen/Daten innerhalb eines Elements
  - Pflicht oder optionale Elemente

DTD deklariert für jedes Element eine Menge von erlaubten Attributen

XML-Schema

Warum XML-Schema?

```
<location>
   <latitude>32.904237</latitude>
   <longitude>73.620290</longitude>
   <uncertainty units="meters">2</uncertainty>
</location>
```



Vorteile von XML-Schemata

Freie Universität Berlin

- Vielzahl von vordefinierten Datentypen
- eigene Datentypen definierbar
- keine eigene Syntax, sondern selbst XML-Sprache
- Vererbungshierarchien
- unterstützen Namensräume
- Reihenfolgeunabhängige Strukturen einfach zu definieren

Die Beispiel-DTD

```
<!ELEMENT BookStore (Book+)>
<!ELEMENT Book (Title, Author, Date, ISBN, Publisher)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>
```

Äquivalentes XML-Schema

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                targetNamespace="http://www.books.org">
   <xsd:element name="BookStore">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="Book"  maxOccurs="unbounded">
               <xsd:complexType>
                  <xsd:sequence>
                     <xsd:element name="Ti
                     <xsd:element name="A
                     <xsd:element name="D
                     <xsd:element name="IS
                     <xsd:element name="Pu
                  </xsd:sequence>
               </xsd:complexType>
            </xsd:element>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

- Für jede DTD gibt es ein äquivalentes XML-Schema.
- Übersetzung z.B. mit XMLSpy
- Umgekehrt gibt es jedoch XML-Schemata, für die es keine äquivalente DTD gibt.

➔ XML-Schemata ausdrucksmächtiger als DTDs

XML-Schemata

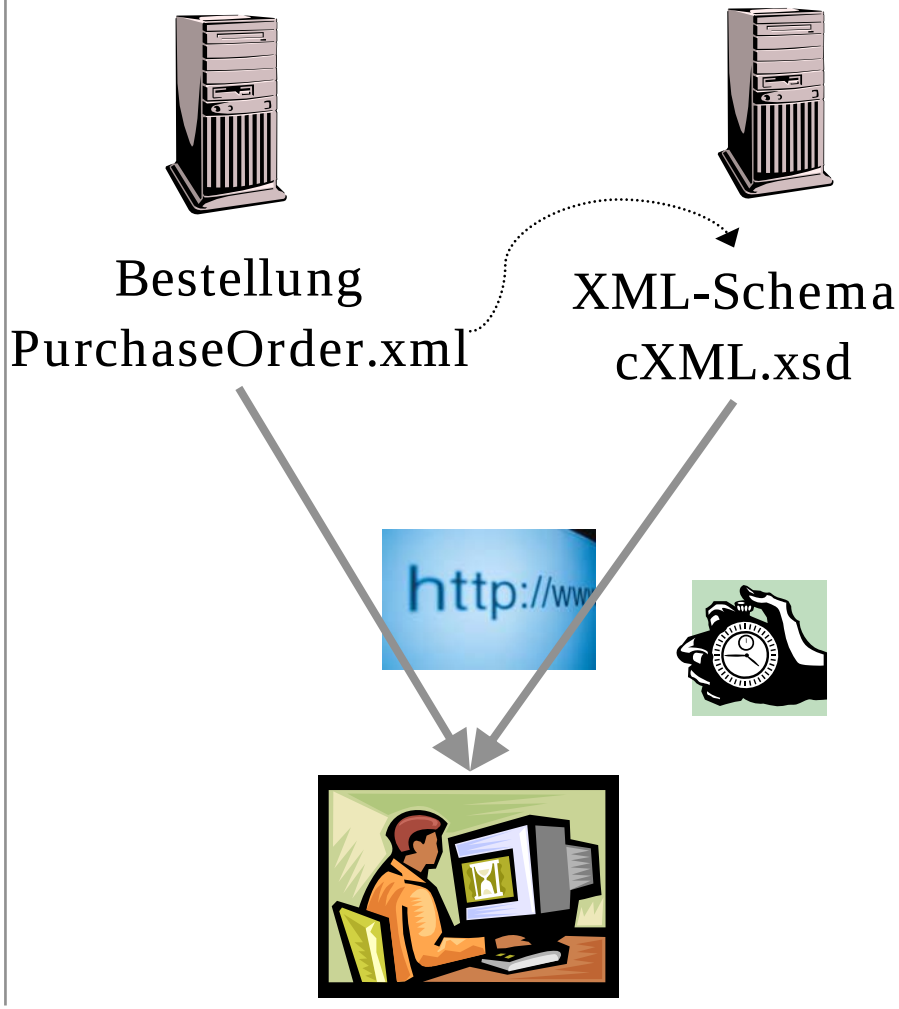
```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" ..>
        ...
</xsd:schema>
```

Namensraum für
das schema Element

- XML-Schemata sind wohlgeformte XML-Dokumente.
- Vorteil: kein eigener Parser nötig
- Nachteil: geschwätzig
- Wurzel-Element: schema aus W3C-Namensraum http://www.w3.org/2001/XMLSchema
  - hier XML-Schema für XML-Schema hinterlegt: Schema der Schemata

Geschwätzigkeit von XML-Schemata

- PurchaseOrder.xml verweist auf cXML-Schema, das Client von anderen Server laden muss.
- cXML
  - als Schema: 50KB
  - als DTD: 15KB
- XML-Schemata aber besser zu komprimieren:
  - cXML-Schema: 6KB
  - cXML-DTD: 4KB

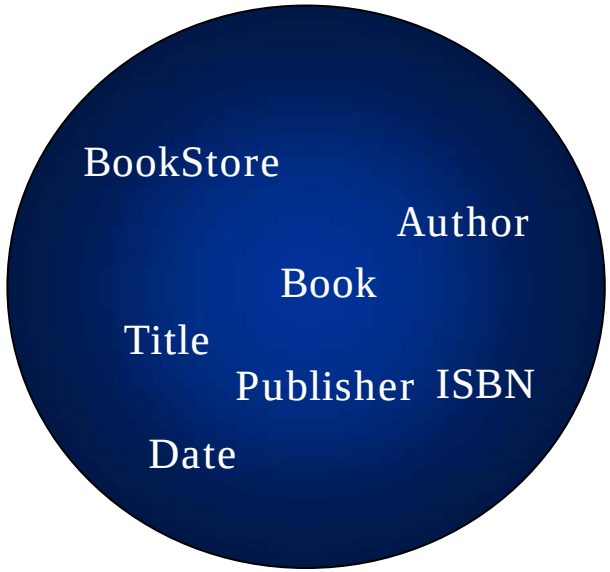


Ziel-Namensraum

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.books.org">
...
</xsd:schema>
```

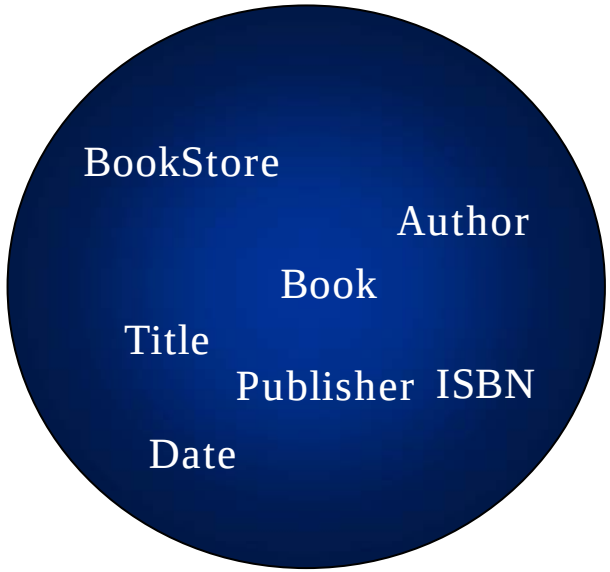
- jedes XML-Schema definiert bestimmtes Vokabular (Elemente und Attribute).
- Dieses Vokabular wird einem Namensraum zugeordnet: Ziel-Namensraum (target namespace).
- Ziel-Namensraum wird wie jeder Namensraum mit URI identifiziert
- Definiertes Vokabular kann über URI identifiziert werden.

Ziel-Namensraum



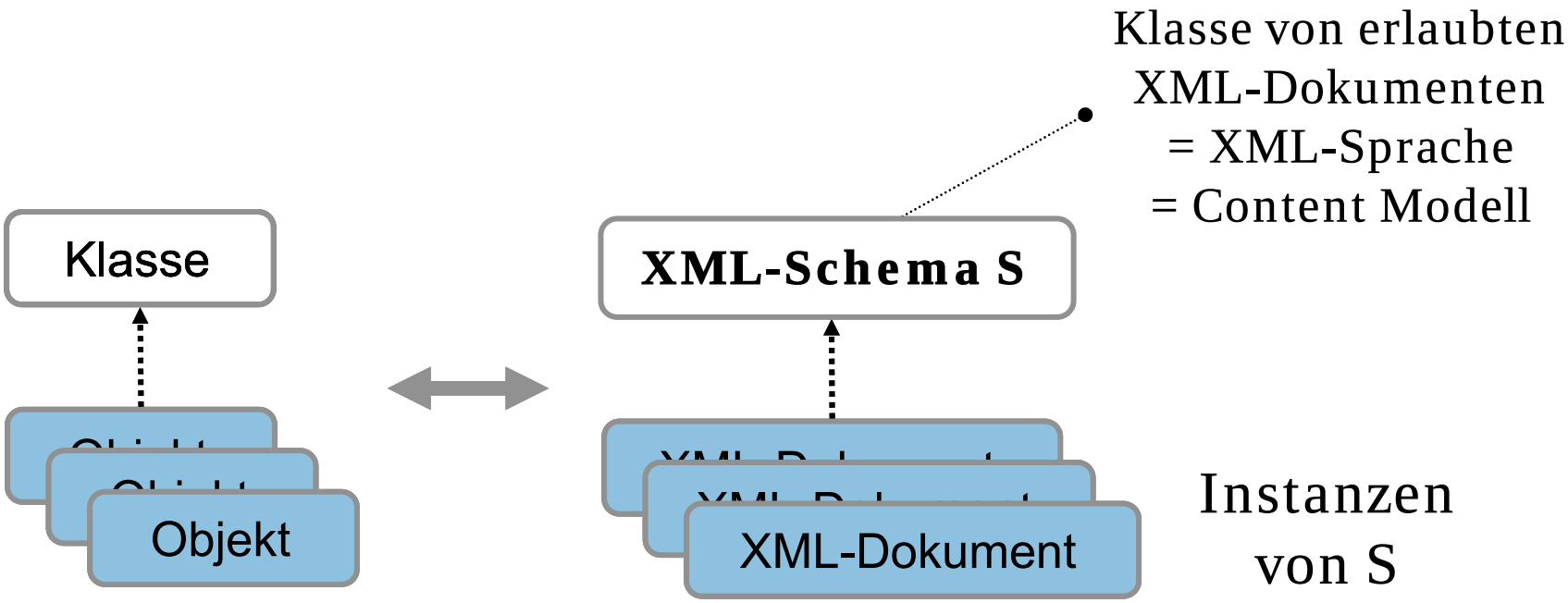
DTD definiert keinen Namensraum.

http://www.books.org



XML-Schema definiert eigenen Namensraum, den Ziel-Namensraum.

Instanz eines XML-Schemas



Instanz eines XML-Schemas S ist ein XML-Dokument, das

1. dem Ziel-Namensraum von S zugeordnet ist und
2. gültig (valid) bzgl. S ist.

Instanz eines XML-Schemas

```
<?xml version="1.0"?>
<BookStore>
    <Book>
        <Title>My Life and Times</Title>
        <Author>Paul McCartney</Author>
        <Date>July, 1998</Date>
        <ISBN>94303-12021-43892</ISBN>
        <Publisher>McMillin Publishing</Publisher>
    </Book>
    ...
</BookStore>
```

Wie wird aus diesem XML-Dokument eine Instanz eines XML-Schemas?

1. Schritt

```
<?xml version="1.0"?>
<BookStore xmlns="http://www.books.org"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.books.org
                               BookStore.xsd">
    ...
</BookStore>
```

Ziel-Namensraum des XML-Schemas

- Wurzel-Element und Namensraum legen zusammen Dokument-Typ fest.
- Wurzel-Element muss in XML-Schema global deklariert sein.
- Für bekannte Namensräume wie http://www.w3.org/1999/xhtml keine weiteren Schritte nötig.

2. Schritt

```
<?xml version="1.0"?>
<BookStore xmlns="http://www.books.org"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.books.org
                               http://www.books.org/BookStore.xsd">
    ...
</BookStore>
```

kann auch lokale Datei wie z.B. BookStore.xsd sein

- Attribut schemaLocation gibt Hinweis, wo das XML-Schema zu finden ist.
- Beachte: XML-Prozessor darf diese Information ignorieren und anderes XML-Schema berücksichtigen!

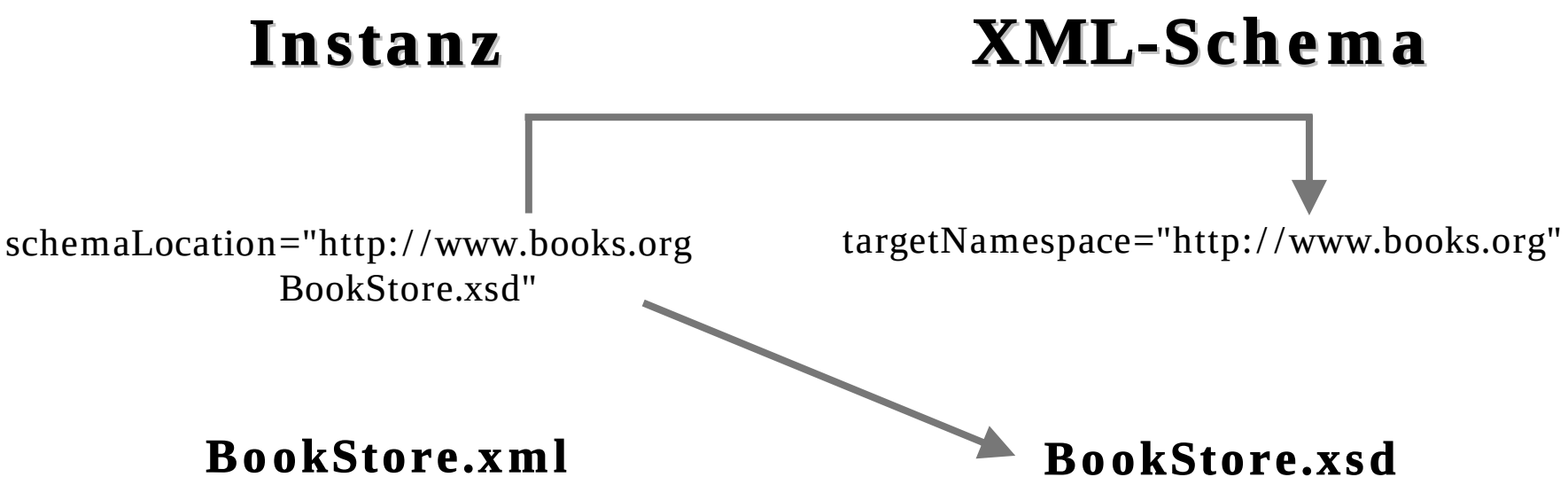
3. Schritt

```
<?xml version="1.0"?>
<BookStore xmlns="http://www.books.org"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.books.org
                               http://www.books.org/BookStore.xsd">
    ...
</BookStore>
```

- Attribut schemaLocation stammt aus dem W3C-Namensraum für Schema-Instanzen.

- weiteres Beispiel für Erweiterbarkeit von XML!
- XML durch Attribut schemaLocation erweitert

Instanz und Schema



benutzt Namensraum
http://www.books.org

definiert Namensraum
http://www.books.org

Software zum Validieren von Instanzen

- xerces by Apache (API)
  - ➔ http://xml.apache.org/
- MSXML (API)
  - ➔ http://www.microsoft.com
- XMLSpy (GUI)
  - ➔ http://www.altova.com/
- weitere: http://www.w3.org/XML/Schema#Tools

Validierung auf mehreren Ebenen

Instanz
= XML-Dokument

XML-Schema

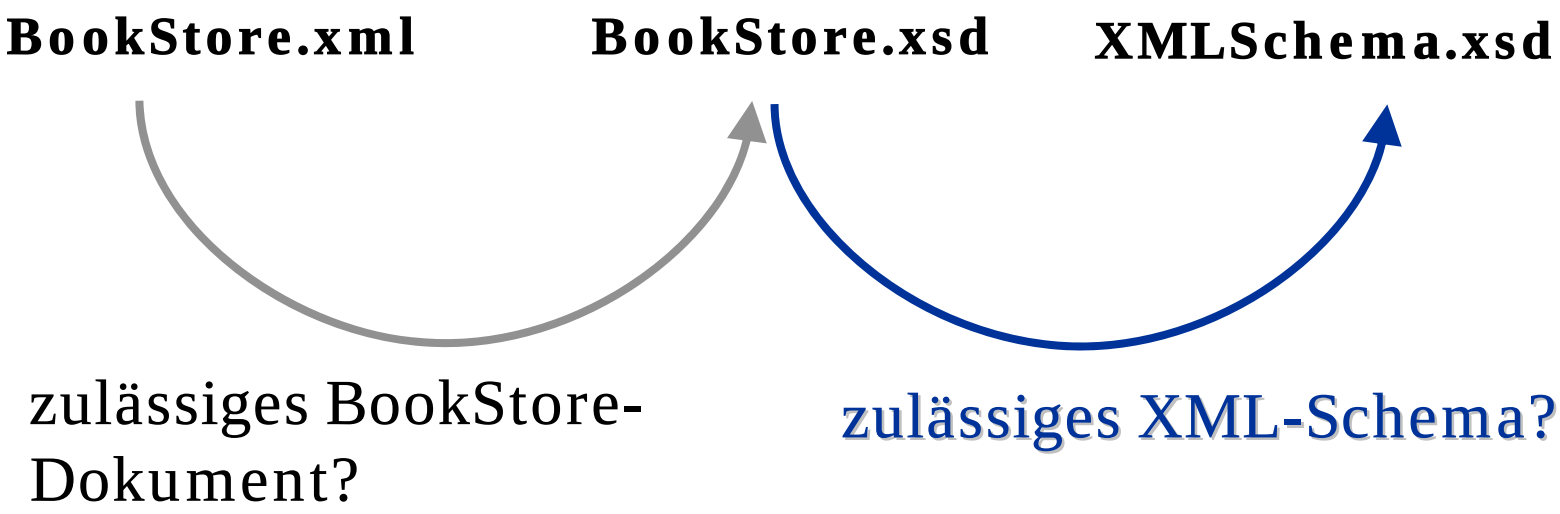
BookStore.xml **BookStore.xsd**

zulässiges BookStore-Dokument?

Validierung auf mehreren Ebenen

Freie Universität Berlin

Instanz = XML-Dokument	XML-Schema = XML-Dokument	Schema der Schemata



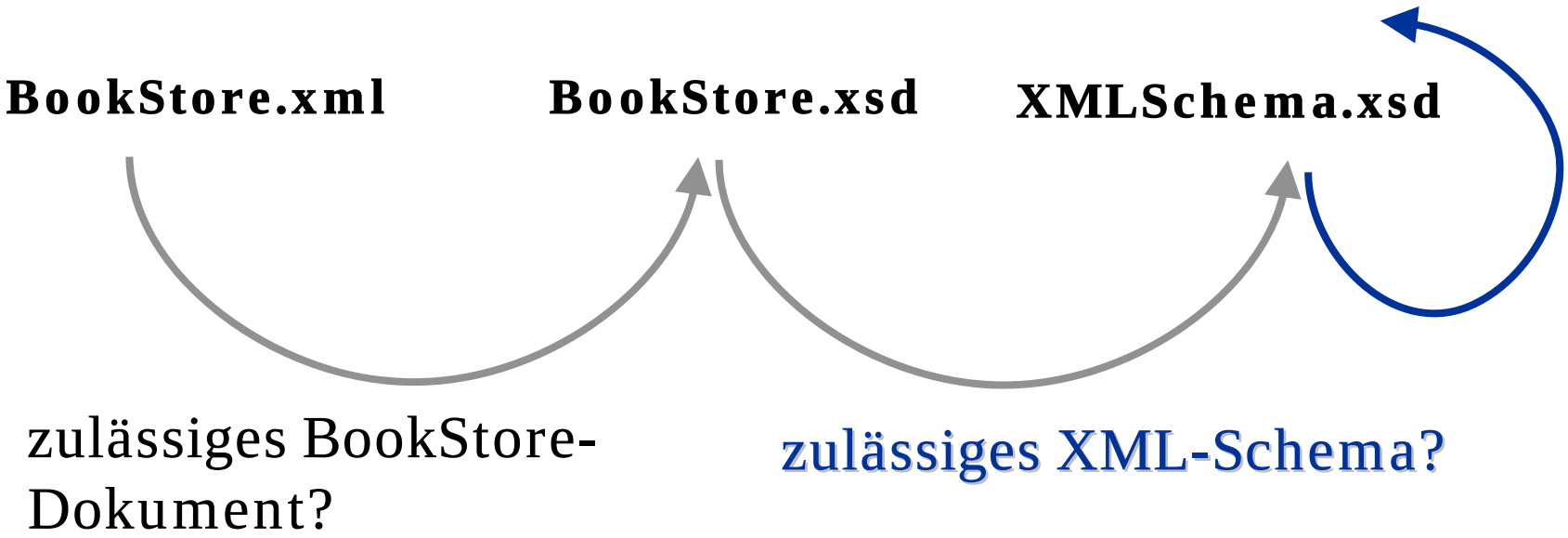
Validierung auf mehreren Ebenen

Freie Universität Berlin

Instanz
= XML-Dokument

Schema
= XML-Dokument

Schema der Schemata = XML-Dokument



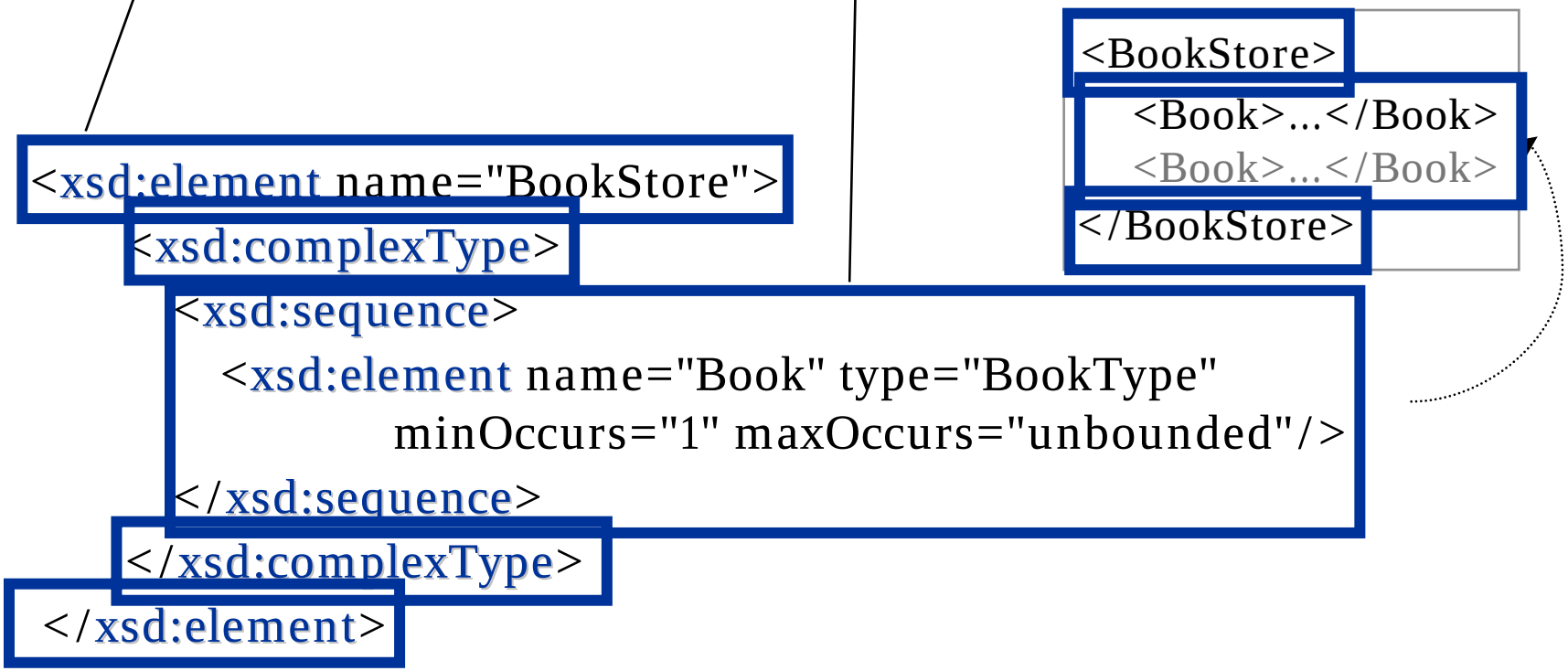
Deklaration der Element-Struktur

Freie Universität Berlin

```
<!ELEMENT BookStore (Book+)>
<!ELEMENT Book (Title, Author, Date, ISBN, Publisher)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>
```

- Wie werden diese Element-Deklarationen mit einem XML-Schema ausgedrückt?

<!ELEMENT BookStore (Book+)>



- xsd:element: Element wird deklariert
- xsd:complexType: strukturierter Inhalt
- xsd:sequence: Sequenz von Elementen

<!ELEMENT BookStore (Book+)>

```
<xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Book" type="BookType"
                     minOccurs="1" maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

```
<BookStore>
   <Book>...</Book>
   <Book>...</Book>
</BookStore>
```

- minOccurs: minimale Anzahl der Wiederholungen
- maxOccurs: maximale Anzahl der Wiederholungen
- Beachte: Standard-Werte für minOccurs und maxOccurs jeweils 1

<!ELEMENT Book (Title, Author, Date, ISBN, Publisher)>

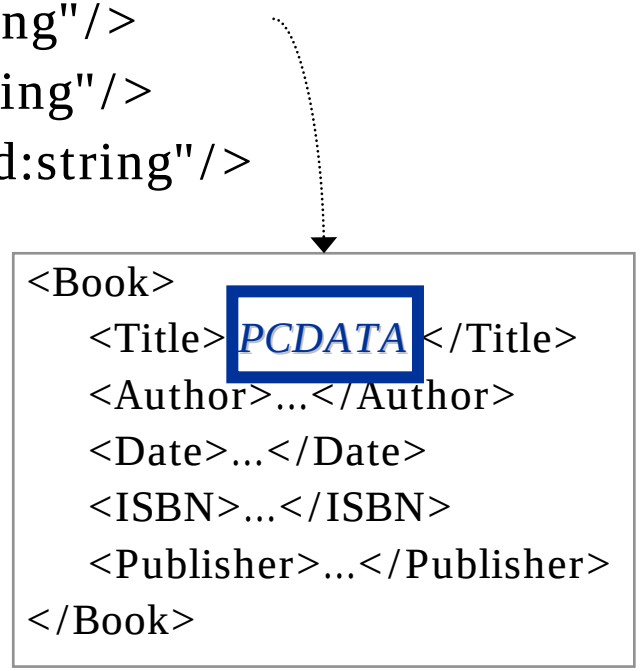
```
<xsd:complexType name="BookType">
    <xsd:sequence>
      <xsd:element name="Title" type="xsd:string"/>
      <xsd:element name="Author" type="xsd:string"/>
      <xsd:element name="Date" type="xsd:string"/>
      <xsd:element name="ISBN" type="xsd:string"/>
      <xsd:element name="Publisher" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>
```

```
<Book>
    <Title>...</Title>
    <Author>...</Author>
    <Date>...</Date>
    <ISBN>...</ISBN>
    <Publisher>...</Publisher>
</Book>
```

- Kind-Elemente: Title, Author, Date, ISBN und Publisher
- wegen xsd:sequence: feste Reihenfolge
- jeweils genau einmal

<!ELEMENT Book (Title, Author, Date, ISBN, Publisher)>

```
<xsd:complexType name="BookType">
    <xsd:sequence>
      <xsd:element name="Title" type="xsd:string"/>
      <xsd:element name="Author" type="xsd:string"/>
      <xsd:element name="Date" type="xsd:string"/>
      <xsd:element name="ISBN" type="xsd:string"/>
      <xsd:element name="Publisher" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>
```



- xsd:string: vordefinierter Datentyp
- 43 weitere vordefinierte Datentypen

<!ELEMENT Book (Title, Author, Date, ISBN, Publisher)>

```
<xsd:complexType name="BookType">
    <xsd:sequence>
      <xsd:element name="Title" type="xsd:string"/>
      <xsd:element name="Author" type="xsd:string"/>
      <xsd:element name="Date" type="xsd:date"/>
      <xsd:element name="ISBN" type="xsd:string"/>
      <xsd:element name="Publisher" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>
```

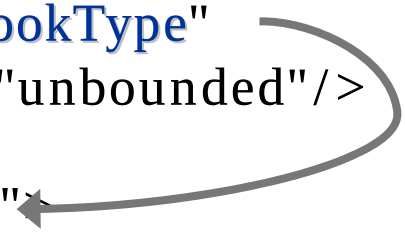
- xsd:date: vordefinierter Datentyp

```
<Book>
    <Title> PCDATA </Title>
    <Author> ... </Author>
    <Date> Kalenderdatum </Date>
    <ISBN>...</ISBN>
    <Publisher>...</Publisher>
</Book>
```

Benannte Datentypen

```
<xsd:element name="Book" type="BookType"
        minOccurs="1" maxOccurs="unbounded"/>

<xsd:complexType name="BookType">
    <xsd:sequence>
      <xsd:element name="Title" type="xsd:string"/>
      <xsd:element name="Author" type="xsd:string"/>
      <xsd:element name="Date" type="xsd:string"/>
      <xsd:element name="ISBN" type="xsd:string"/>
      <xsd:element name="Publisher" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>
```



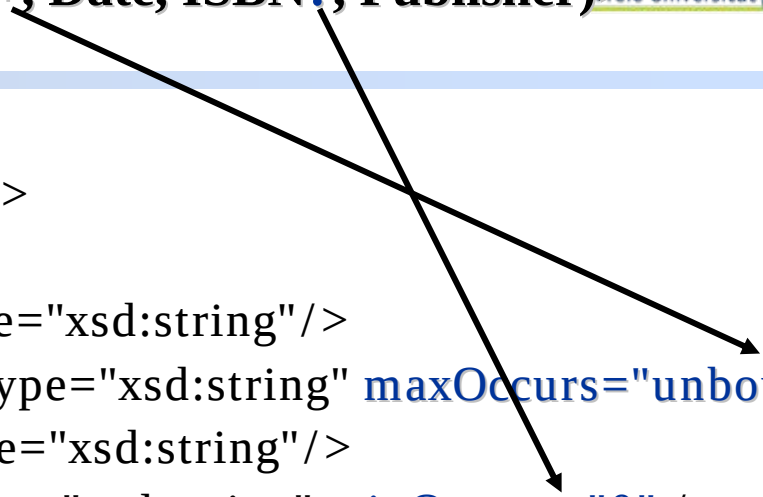
- BookType hier benannter Datentyp (named type)
- auch globale Typ-Definition genannt

Alternative: anonyme Datentypen

```
<xsd:element name="Book"  maxOccurs="unbounded">
        <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Title" type="xsd:string"/>
              <xsd:element name="Author" type="xsd:string"/>
              <xsd:element name="Date" type="xsd:string"/>
              <xsd:element name="ISBN" type="xsd:string"/>
              <xsd:element name="Publisher" type="xsd:string"/>
            </xsd:sequence>
        </xsd:complexType>
</xsd:element>
```

- äquivalente Formulierung mit anonymen Datentyp
- auch als lokale Typ-Definition bezeichnet
- Nachteil: kann an anderer Stelle nicht wieder verwendet werden

<!ELEMENT Book (Title, Author+, Date, ISBN?, Publisher)>



```
<xsd:complexType name="BookType">
    <xsd:sequence>
      <xsd:element name="Title" type="xsd:string"/>
      <xsd:element name="Author" type="xsd:string" maxOccurs="unbounded" />
      <xsd:element name="Date" type="xsd:string"/>
      <xsd:element name="ISBN" type="xsd:string" minOccurs="0" />
      <xsd:element name="Publisher" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>
```

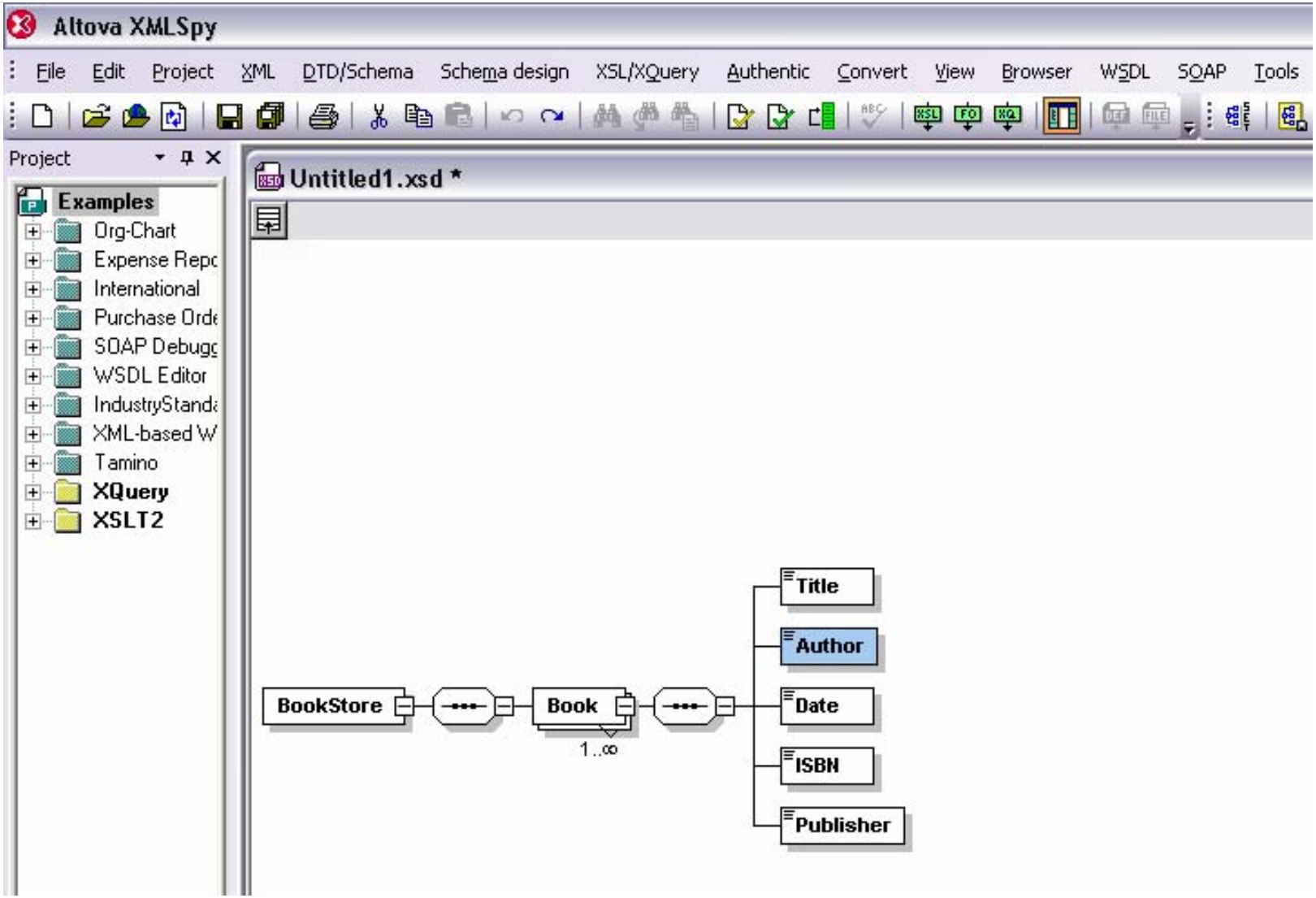
- Jedes Elemente erscheint so häufig, wie mit minOccurs und maxOccurs festgelegt.

Das vollständige XML-Schema

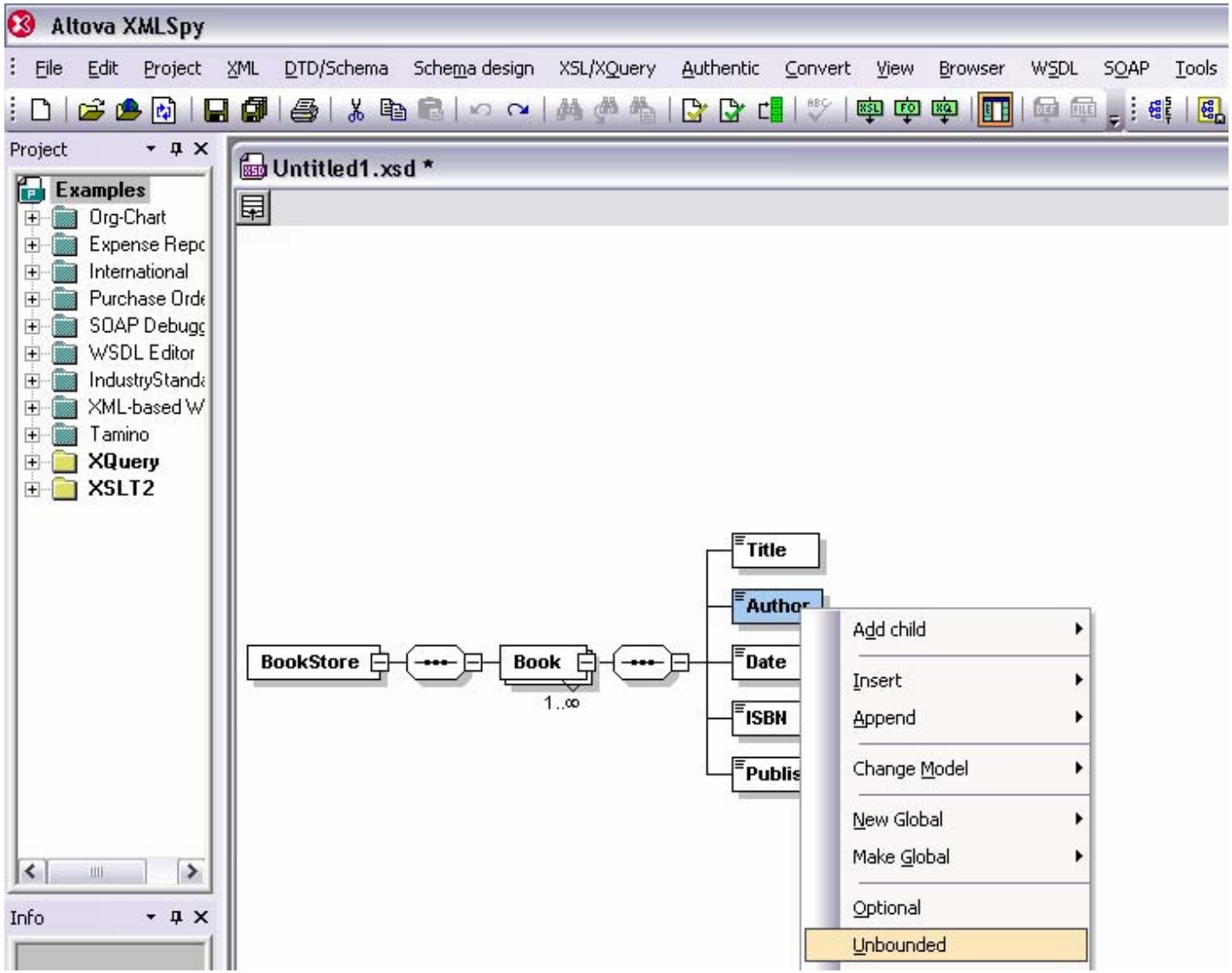
```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                targetNamespace="http://www.books.org">
   <xsd:element name="BookStore">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="Book"  maxOccurs="unbounded">
               <xsd:complexType>
                  <xsd:sequence>
                     <xsd:element name="Title" type="xsd:string"/>
                     <xsd:element name="Author" type="xsd:string"/>
                     <xsd:element name="Date" type="xsd:string"/>
                     <xsd:element name="ISBN" type="xsd:string"/>
                     <xsd:element name="Publisher" type="xsd:string"/>
                  </xsd:sequence>
               </xsd:complexType>
            </xsd:element>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Lernziel: Dieses Schema und die entsprechende DTD verstehen!

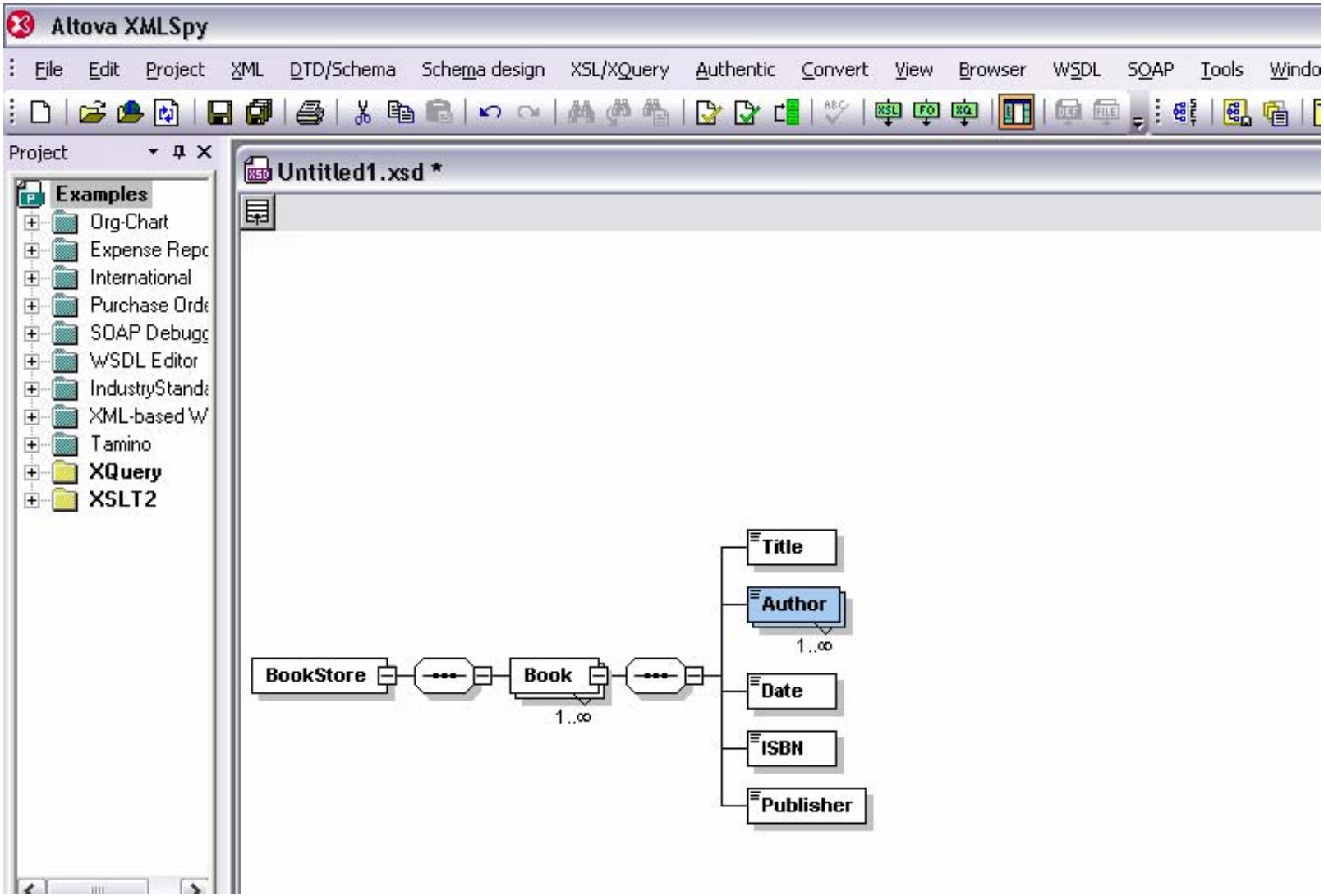
XML Spy: Visualisierung



XML Spy: Editieren



XML Spy: Editieren



Wie geht es weiter?

Freie Universität Berlin

heute

- ☑ Definition von XML-Sprachen
- ☑ DTDs und XML-Schema anhand eines einheitlichen Beispiels

nächste Woche

- XML-Schema im Detail