UNIVERSITÄT SIEGEN

Vorlesung Datenbanksysteme II

Einführung in die XML

Inhalt

- Datenmodell von XML-Dateien
- Korrektheitsstufen von XML-Dateien
- Grundlegende lexikalische Eigenschaften von XML-Dateien
- Syntaktische Struktur von XML-Dateien

Hintergrund

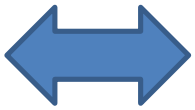
- Anwendungsgebiete mit **semistrukturierten** Daten und **baumartigen** Dokumentstrukturen.
- **Selbstbeschreibende** Datei-Formate mit oder ohne vorheriger Typdefinition, dargestellt als **lesbarer Text**.
- Häufig zunächst „unkontrollierte“ Erstellung von (**potentiell inkonsistenten**) Dokumenten in generischen Editoren oder unbekannten Applikationen.

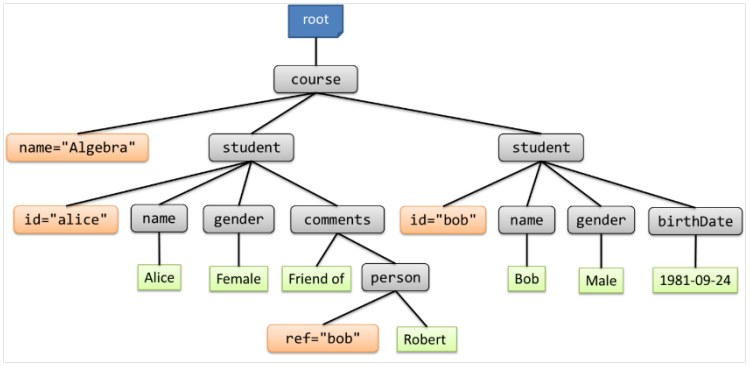
UNIVERSITÄT SIEGEN

Übersicht über das Datenmodell von XML-Dateien

Datenmodell von XML

XML-Datei



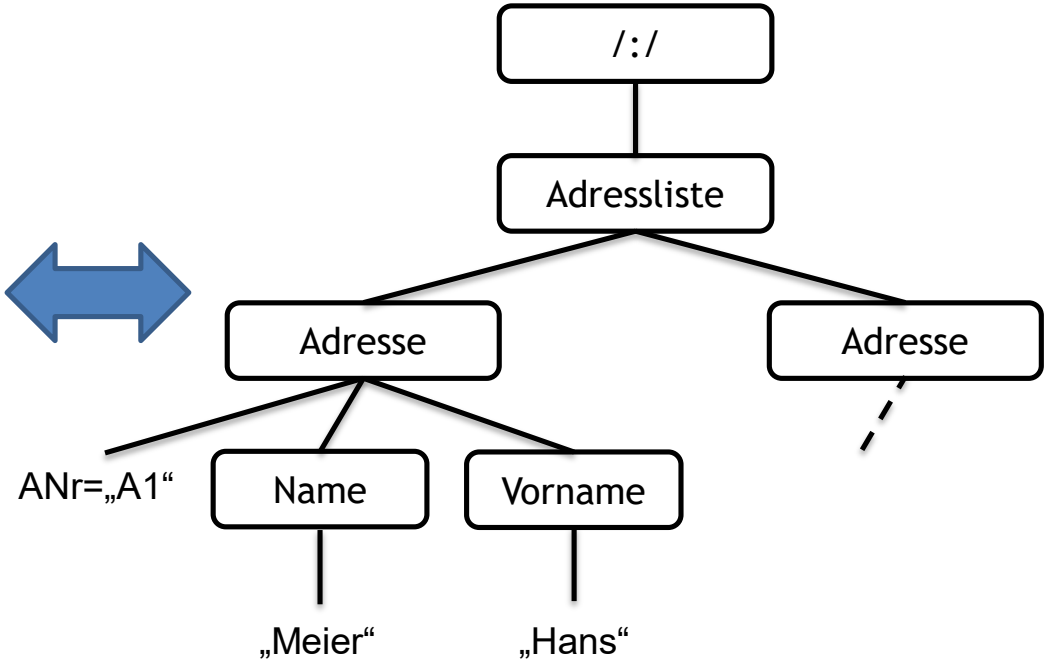


Attributierter Baum

Im Gegensatz zu (R)DBS umfasst das grundlegende Datenmodell von XML keine vordefinierten Operationen und (fast) keine Integritätsbedingungen.

Beispiel: Adressliste

```
<Adressliste>
  <Adresse ANr='A1'>
    <Name>Meier</Name>
    <Vorname>Hans</Vorname>
  </Adresse>
  <Adresse>
  ....
  </Adresse>
</Adressliste>
```

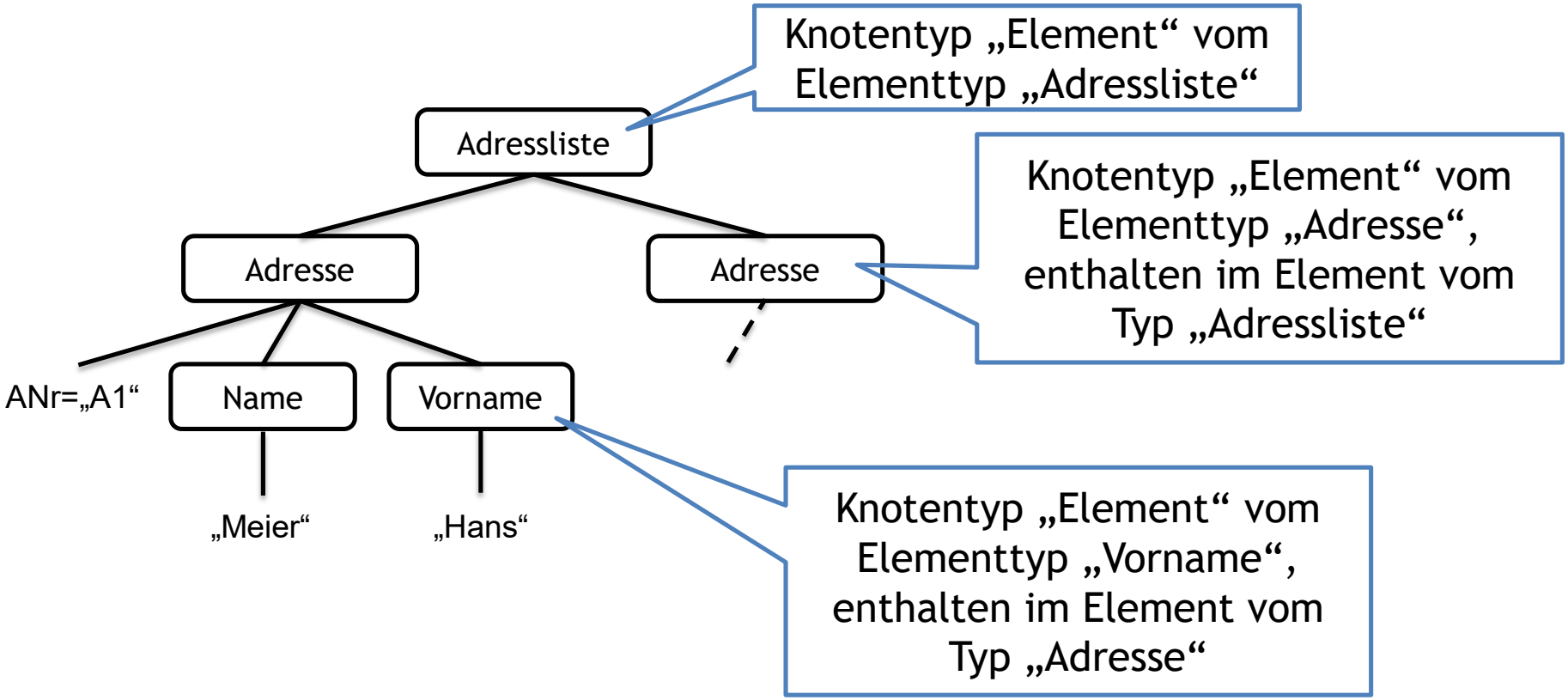


XML-Knotentypen (Auswahl)

Knotentyp „Element“

- Elemente haben Namen, die den Elementtyp (nicht den Knotentyp!) charakterisieren.
- Elemente können Attribute aufweisen und andere Knoten enthalten.
- Alle inneren Knoten des Baumes sind Elemente.

Beispiel: Elementknoten

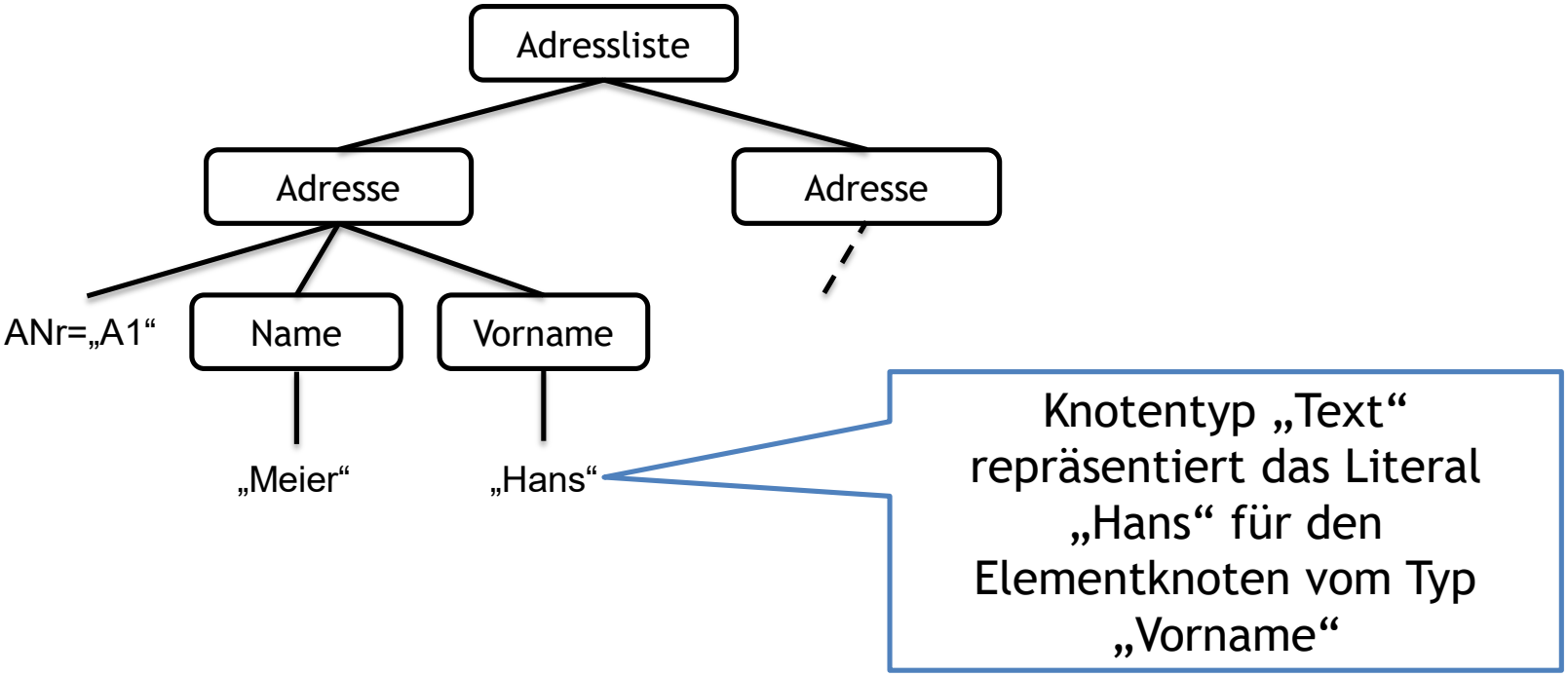


XML-Knotentypen (Auswahl)

Knotentyp „Text“ (Literal)

- Literale repräsentieren beliebigen Text.
- Literale treten nur als Blattknoten des Baums auf.

Beispiel: Elementknoten



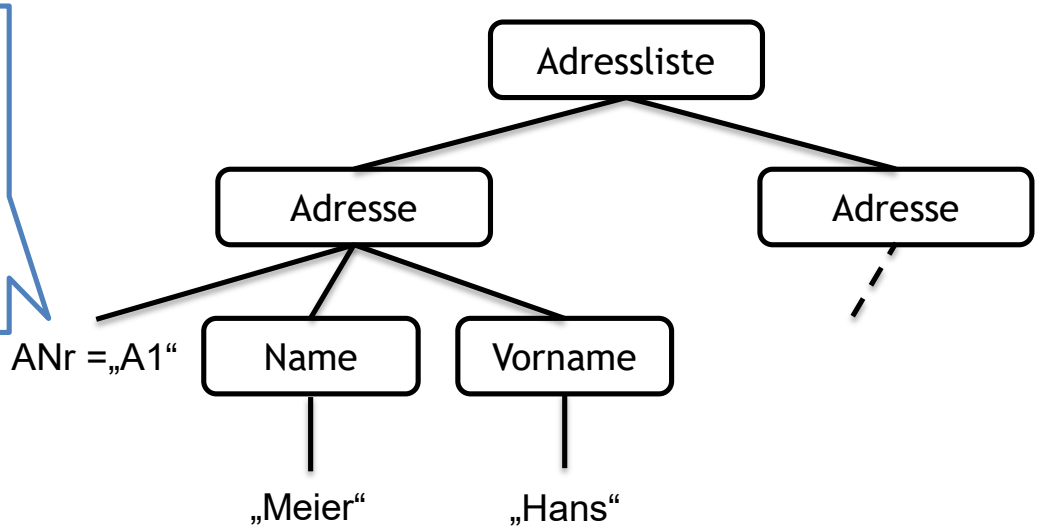
XML-Knotentypen (Auswahl)

Knotentyp „Attribut“

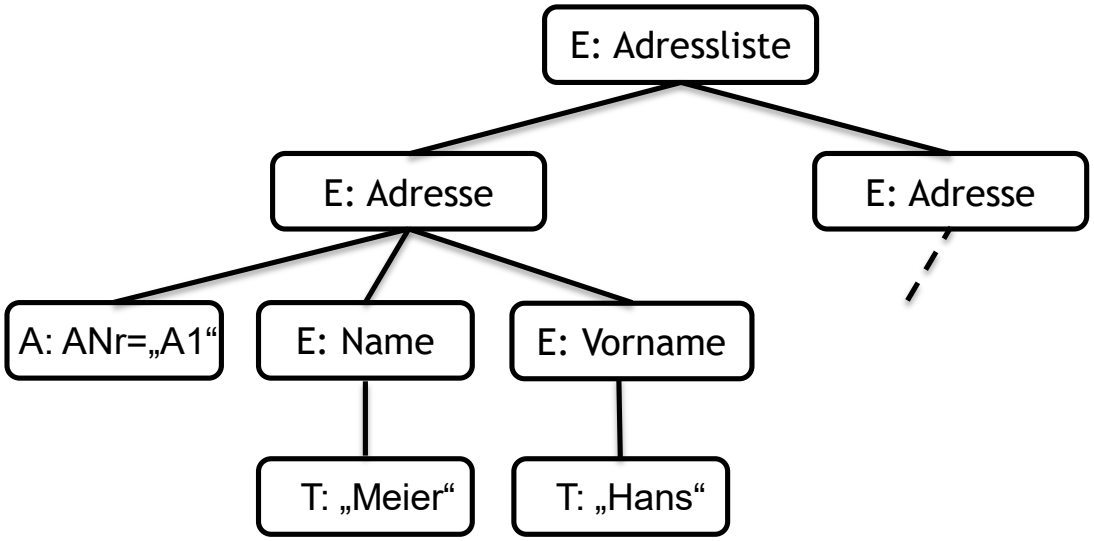
- Attribute haben einen Namen, der den Attributtyp repräsentiert, und einen zugeordneten Wert.
- Attribute sind genau einem Elementknoten zugeordnet.
- Alle Attributtypen eines Elements haben unterschiedliche Namen.

Beispiel: Attributknoten

Knoten vom Typ „Attributknoten“ für das Attribut vom Typ „ANr„ des Elementtyps „Adresse“ mit dem zugeordneten Wert „A1“



Beispiel: Alternative Darstellung als getypter Baum



Annotierte Knotentypen: Element (E), Attribut (A), Text (T)

UNIVERSITÄT SIEGEN

Korrektheitsstufen von XML-Dateien

Wohlgeformte XML-Dokumente

Ein XML-Dokument ist (syntaktisch) **wohlgeformt**, wenn es als Darstellung eines (getypten) Baumes interpretiert werden kann.

Elementare syntaktische Korrektheit (Auswahl)

- Spitze Klammern der tags syntaktisch korrekt.
- Paarweise öffnende und schließende tags vorhanden.
- Korrekte Schachtelung der Elemente.
- Attribute nur innerhalb öffnender tags.
- Attributschreibweise a=’x’ oder a="x".

```
<Adressliste>
  <Adresse ANr=’A1’>
    <Name>Meier</Name>
    <Vorname>Hans</Vorname>
  </Adresse>
  <Adresse>
  ....
  </Adresse>
</Adressliste>
```

Gültige XML-Dokumente

Ein XML-Dokument ist (semantisch) **gültig**, wenn alle Baumknoten sinnvoll angeordnet sind.

Gültige XML-Dokumente

```
<Adressliste>
  <Adresse ANr='A1'>
    <Name>Meier</Name>
    <Vorname>Hans</Vorname>
  </Adresse>
  <Adresse>
  ....
  </Adresse>
</Adressliste>
```

- XML-Dokumente können beliebig wohlgeformte, aber trotzdem unsinnige Anordnungen von Knoten enthalten.
- Beispiel: weiterer Adressliste-Knoten unterhalb eines Vornamen-Knotens.
- Gültigkeit kann nur hinsichtlich einer **zusätzlichen Spezifikation** geprüft werden.

Spezifikation von Korrektheitsbedingungen

Methoden (Sprachen) zur Spezifikation von Korrektheitsbedingungen:

- **Dokumenttyp-Definition (DTD)**: Teil des grundlegenden XML-Standards; eingeschränkte Ausdruckskraft.
- **XML Schema:** Späterer, zusätzlicher Standard; weitaus umfangreicher, hohe Ausdruckskraft.

Beispiel: DTD für Adresslisten

```
<!DOCTYPE Adressliste [
<!ELEMENT Adressliste (Adresse)* >
<!ELEMENT Adresse (Name, Vorname) >
<!ELEMENT Name (#PCDATA) >
<!ELEMENT Vorname (#PCDATA) >
<!ATTLIST Adresse
   ANr CDATA >
]>
```

- BNF-Notation: „*“ beliebig oft, „+“ mindestens einmal, „?“ gar nicht oder einmal, „-“ abzüglich.
- Weitere Details folgen später.

Fazit: Gültigkeit von XML-Dokumenten

Ein XML-Dokument ist **gültig**, wenn

1. eine zugehörige DTD vorhanden ist und
2. die darin spezifizierten Korrektheits-bedingungen durch die Nutzdaten erfüllt sind.

UNIVERSITÄT SIEGEN

Grundlegende lexikalische Eigenschaften von XML-Dateien

Zeichen und Sonderzeichen

- Alle erlaubten Zeichen gemäß ISO 10646 (UTF-16/32); stehen in der Regel für sich selbst.
- Ausnahme: Die Sonderzeichen „-“, „<“, „>“ werden als Beginn und Ende eines tags interpretiert.
- Ausnahme: „&*es*;“ bewirkt die Expansion eines (vor-)definierten Entity-Symbols „*es*“

  Beispiele: *< > &*

Uninterpretierte Textabschnitte

Die Interpretation von Sonderzeichen in Textabschnitten kann durch den CDATA-Abschnitt verhindert werden und ist überall dort erlaubt, wo Texte stehen können:

<![CDATA[.... uninterpretierter Text ....]]>

UNIVERSITÄT SIEGEN

Syntax uninterpretierter Textabschnitte

CDSect ::= CDStart CData CDEnd

CDStart ::= '<![CDATA['

CData ::= (Char* - (Char* ']]>' Char*))

CDEnd ::= ']]>'

Namen (Bezeichner)

- Regeln für Namen (von Elementtypen, Attributen usw.) wie in Programmiersprachen üblich.
- Dürfen nicht mit dem Präfix ’xml’ in Klein- oder Großbuchstaben beginnen (also nicht: (’X’ | ’x’) (’M’ | ’m’) (’L’ | ’l’)).
- Doppelpunkte innerhalb von Namen sind erlaubt, haben aber eine besondere Bedeutung (siehe Thema Namensräume).

Syntax von Namen (Bezeichnern)

Name ::= (Letter | ’_’ | ’:’) (NameChar)*

NameChar ::= Letter | Digit | ’.’ | ’-’ | ’_’ | ’:’ |
CombiningChar | Extender

Attributinstanzen

- Syntax für Attribute eines Elements:

  attrname="wert"

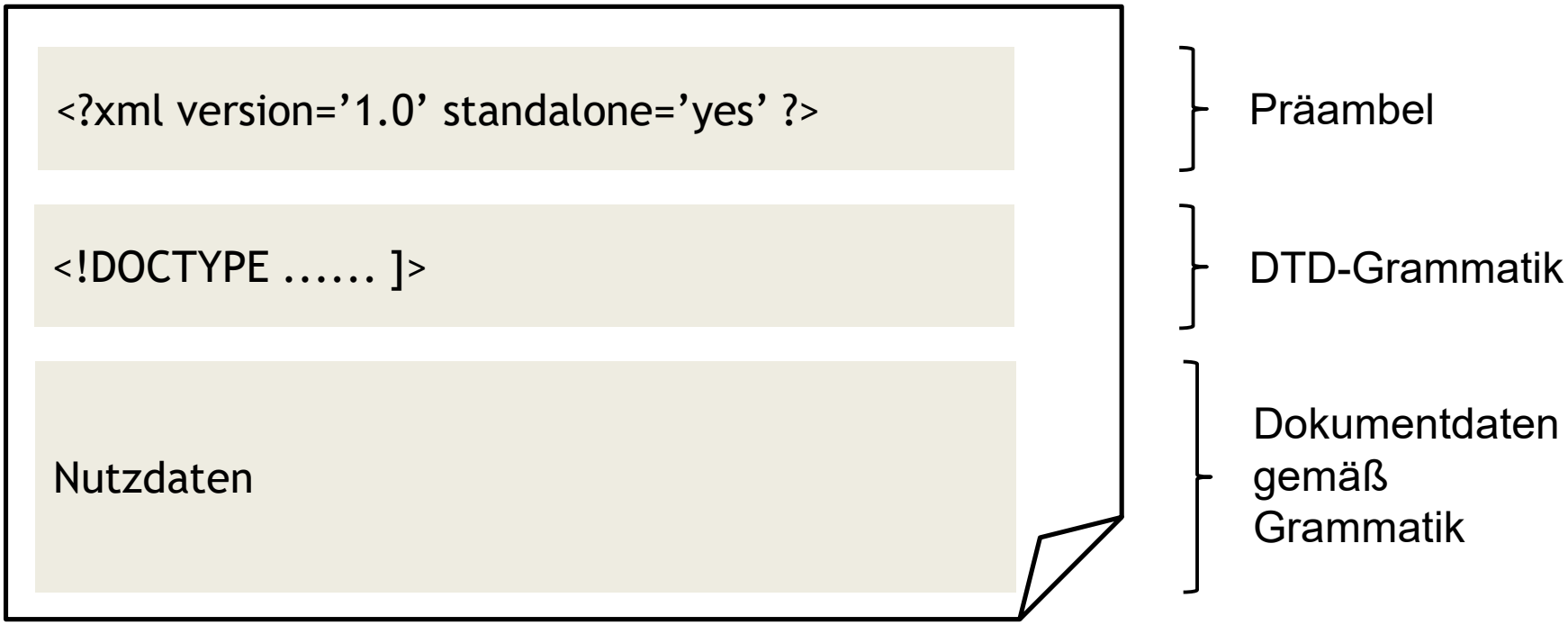
- Anführungszeichen oder Apostrophe sind obligatorisch!

  Beispiel: x="3" oder x=’3’

UNIVERSITÄT SIEGEN

Syntaktische Struktur von XML-Dateien

Aufbau von XML-Dateien



Syntax des Aufbaus von XML-Dateien

document ::= prolog element Misc*

- „document“ ist Startsymbol der XML-Grammatik.
- „prolog“ umfasst die erste Zeile (Präambel) und die (optionale) DTD.
- „element“ umfasst die Nutzdaten.
- „Misc“ für sonstige Angaben.

Prolog

Der Prolog-Abschnitt enthält:

- In der ersten Zeile: Angaben zu XML-Version, Zeichensatzcodierung und eine Angabe darüber, ob alle nachfolgend verwendeten Deklarationen enthalten sind (wichtig u.a. für Vorgabewerte von Attributen).
- die (optionale) DTD.

Syntax für den Prolog

```
prolog      ::= XMLDecl? Misc* (doctypedecl
                Misc*)?
XMLDecl     ::= '<?xml' VersionInfo EncodingDecl?
                SDDecl? S? '?>'
```

- Achtung: kein Leerraum vor „XMLDecl“ erlaubt.
- „EncodingDecl“ gibt die verwendete Zeichensatzcodierung an.

Syntax für sonstige Angaben

```
Misc      ::= Comment | PI | S
S         ::= (#x20 | #x9 | #xD | #xA)+
Comment   ::= '<!--' ((Char - '-') | ('-' (Char - '-')))* '-->'
```

Syntax für sonstige Angaben

- „S“ (*empty Space*): Leerräume (Leerzeichen, Tabulatoren, …).
- „#x“: übliche darstellbare Zeichen und Zeichen gemäß ISO/IEC 10646.
- „PI“: Verarbeitungsanweisung (weitere Knotenart).
- „Comment“ XML-Kommentar (weitere Knotenart).

Syntax für den Prolog

Beispiele für die „XMLDecl“-Klausel:

<?xml version=’1.0’ standalone=’yes’ ?>

<?xml version="1.0" encoding=’ISO-8859-1’ ?>

Syntax für den Prolog

```
VersionInfo     ::= S 'version' Eq ("'" VersionNum
                    "'"| '"' VersionNum '"')
Eq              ::= S? '=' S?
VersionNum      ::= ([a-zA-Z0-9_.:] | '-')+
```

- „VersionNum“: Derzeit 1.0 oder 1.1 möglich.
- Änderungen in Version 1.1 betreffen vor allem Zeichensätze und in Namen zulässige Zeichen.

Syntax für den Prolog

```
SDDecl      ::= S 'standalone' Eq
                  (("'" ('yes' | 'no') "'") |
                   ('"' ('yes' | 'no') '"'))
```

- 'yes' : Datei enthält die vollständige Grammatik.
- 'no' oder SDDecl fehlt ganz: Grammatik fehlt (teilweise) und die (korrekte) Verarbeitung hängt von externen Ressourcen ab.

Dokumenttypdeklarationen

Nochmal unser Beispiel:

```
<!DOCTYPE Adressliste [
<!ELEMENT Adressliste (Adresse)* >
<!ELEMENT Adresse (Name, Vorname) >
<!ELEMENT Name (#PCDATA) >
<!ELEMENT Vorname (#PCDATA) >
<!ATTLIST Adresse
  ANr CDATA >
]>
```

Syntax von Dokumenttypdeklarationen

```
doctypedecl    ::= '<!DOCTYPE' S Name
                   (S ExternalID)? S?
                   ('[' (markupdecl | DeclSep)*
                   ']' S?)? '>'
ExternalID     ::= 'SYSTEM' S SystemLiteral |
                   'PUBLIC' S PubidLiteral S
                   SystemLiteral
```

Syntax von Dokumenttypdeklarationen

- Interne Quelle: der nach „DOCTYPE“ angegebene „Name“ ist Name des Wurzel-Elementtyps der DTD.
- Externe Quelle, z.B.: <!DOCTYPE Adressliste SYSTEM "adressliste.dtd">.
- Externe *und* interne Dokumenttypdeklarationen können gemischt werden (bei Namenskonflikten hat die interne Dokumenttypdeklarationen Priorität).

Markupdeklarationen

markupdecl ::= elementdecl | AttlistDecl |
EntityDecl | NotationDecl |
PI | Comment

- Alle tags von Markupdeklarationen beginnen mit der Zeichenfolge „<!“.

Elementtypdeklarationen

Elementtypdeklarationen sind zweigeteilt:

1. Inhalt (zwischen den tags): Angabe in der „elementdecl“-Klausel.

2. Attribute (im öffnenden tag): Angabe der Attribute in separaten „AttlistDecl“-Klauseln.

Elementtypdeklarationen

elementdecl ::= ’<!ELEMENT’ S Name S contentspec S? ’>’

contentspec ::= ’EMPTY’ | ’ANY’ | Mixed | children

Beispiel: <!ELEMENT plz ANY>

Elementtypdeklarationen

- ’EMPTY’: leerer Inhalt.
- ’ANY’: beliebiger Inhalt, d.h. normaler Text und wohlgeformte tags, beliebig gemischt.
- mixed: normaler Text und zusätzlich darin eingestreute Elemente mit vorgegebenen Typen.
- children: nur Elemente.

Kindknoten in XML-Dokumenten

- Der Inhalt einer XML-Datei repräsentiert einen Syntaxbaum.
- Im Syntaxbaum wird **jedes Element und jedes Attribut** durch einen Knoten repräsentiert.
- Ein Elementknoten kann weitere Elementknoten, Attributknoten, Kommentarknoten usw. “unter sich” haben.
- Achtung: Kindknoten in der XML-Diktion sind **alle Knoten außer den Attributknoten!**

Mixed Elemente

1. Nur normaler Text.

Beispiel: <!ELEMENT e (#PCDATA)>

- „PCDATA“ steht für „parsed character data“.
- Das Element darf keine tags enthalten (hat keine Kindelemente).

Mixed Elemente

2. Text mit zulässigen eingestreuten Elementen.
Beispiel: <!ELEMENT e (#PCDATA | a | b | c)* >

- „#PCDATA“ muss an erster Stelle stehen.
- Es folgt eine Liste der erlaubten Elementtypen.
- Am Ende der Aufzählung muss ein ’*’ stehen, dabei kein Leerzeichen zwischen „)“ und „*“.

Syntax der Mixed Elemente

Mixed ::= '(' S? '#PCDATA' (S? '|' S? Name)* S? ')*' | '(' S? '#PCDATA' S? ')'

Children Elemente

Beispiele:

```
<!ELEMENT Adressliste ( Adresse )* >
<!ELEMENT Adresse ( Famname, Vorname+, Titel?,
        ( Inlandsanschrift | Auslandsanschrift ) >
```

UNIVERSITÄT SIEGEN

Syntax der Children Element

```
children    ::= (choice | seq) ('?' | '*' | '+')?
choice      ::= '(' S? cp ( S? '|' S? cp )+ S? ')'
seq         ::= '(' S? cp ( S? ',' S? cp )* S? ')'
cp          ::= (Name | choice | seq)
                ('?' | '*' | '+')?
```

Syntax der Children Elemente

- „choice“: Auswahl (mindestens 2 Elemente). Beispiel: „(a | b | c)“ bedeutet, dass entweder „a“ oder „b“ oder „c“ an dieser Stelle (in den Nutzdaten) steht.
- „seq“: Sequenz (mindestens 1 Element). Beispiel: „(a, b, c)“ bedeutet, dass an dieser Stelle „a“ gefolgt von „b“ und „c“ steht.

Attributlistendeklaration

Attributlistendeklarationen stehen in separaten Klauseln für die Attribute eines Elementtyps.
Beispiel:

```
<!ATTLIST Adresse
            Name CDATA #REQUIRED
            Vorname CDATA #REQUIRED
            Strasse CDATA #IMPLIED
>
```

Attributdeklarationen

Inhalt einer Attributdefinition:

- „Name“: Name des Attributs.
- „AttType“: Typ des Attributs.
- „DefaultDecl“ Vorgabewert / Optionalitätsangabe.

Syntax von Attributdeklarationen

AttlistDecl ::= ’<!ATTLIST’ S Name AttDef* S? ’>’

AttDef ::= S Name S AttType S DefaultDecl

- Mehrere Attributlistendeklarationen für gleichen Elementtyp erlaubt.
- Falls gleiches Attribut mehrfach definiert ist, gilt die erste beim Einlesen verarbeitete Definition.

Formatierungsrichtlinien für Attributdeklarationen

Es wird folgende Formatierung empfohlen:

- Erste Zeile: nur „<!ATTLIST“ und der Name des Elementtyps.
- Pro Attribut eine Zeile, eingerückt.
- Schließendes „>“ am besten in eigene abschließende Zeile, genau unter dem „<„.

Syntax von Attributwertangaben

DefaultDecl ::= ’#REQUIRED’ | ’#IMPLIED’ |
((’#FIXED’ S)? AttValue)

- „#REQUIRED“: Attributwert obligatorisch.
- „#IMPLIED“: Attributwert optional.
- „AttValue“: Vorgabewert.
- „#FIXED“: muss gleich Vorgabewert sein.

Vorgabewerte

- Achtung: Vorgabewerte sind keine Konsistenz-/ Korrektheitskriterien.
- Vorgabewerte definieren das „Verhalten“ standardkonformer Software.
- Wenn keine explizite Wertangabe vorhanden ist, liefert die Software diesen Vorgabewert.

Konstante Attribute

Beispiel: Übertragung nichtpersistenter Daten.

```
<!ATTLIST FormularX
          Uebertragungsmethode CDATA #FIXED
          "POST"
>
```

Attributtypen

- „StringType“ (CDATA): beliebiger Text ohne „<“. Vorverarbeitung: Tabulatoren und Zeilen-vorschübe werden durch Leerzeichen ersetzt.
- „TokenizedType“: Wortlisten. Vorverarbeitung: wie CDATA , zusätzlich wird Leerraum auf ein Leerzeichen reduziert und Leerraum vorne und hinten entfernt.
- „EnumeratedType“: Aufzählungstypen.

Syntax von Attributtypen

AttType ::= StringType | TokenizedType | EnumeratedType

StringType ::= 'CDATA'

TokenizedType ::= 'ID' | 'IDREF' | 'IDREFS' | 'ENTITY' | 'ENTITIES' | 'NMTOKEN' | 'NMTOKENS'

EnumeratedType ::= NotationType | Enumeration

Eigenschaften von Attributtypen

„ID“: Name gemäß Syntax von „Name“.

- Der Name muss im **gesamten Dokument eindeutig** sein.
- Muss #IMPLIED oder #REQUIRED sein.
- Nur ein ID-Attribut pro Element erlaubt!

Eigenschaften von Attributtypen

„IDREF“: Name gemäß Syntax von „Name“.

- Referenz auf ein anderes Element.
- Dokument ungültig, falls kein solches Element existiert, dessen ID-Attribut diesen Datenwert enthält.

„IDREFS“: entsprechend für mehrere Referenzen.

Eigenschaften von Attributtypen

„NMTOKEN“: Genau ein Namenstoken gemäß der Syntax von „Nmtoken“.

- Name muss (anders als ID) nicht eindeutig sein.
- Beispiel:

```
<!ATTLIST bestellung
            datum NMTOKEN #REQUIRED
>
```

(„NMTOKENS“: entsprechend mehrere, durch Leerraum getrennte, Namenstoken.)

Syntax weiterer Atributtypen

```
Nmtoken          ::= (NameChar)+
Nmtokens         ::= Nmtoken (S Nmtoken)*
EnumeratedType   ::= NotationType | Enumeration
NotationType     ::= 'NOTATION' S '(' S? Name (S?
                     '|' S? Name)* S? ')'
Enumeration      ::= '(' S? Nmtoken (S? '|' S?
                     Nmtoken)* S? ')'
```

Aufzählungstypen

Beispiel:

```
<!ATTLIST mitarbeiter
          steuerklasse (I|II|III|IV|V|VI)
          #REQUIRED
>
```

Diskussion: Schlüsselattribute

ID-Attribute vs. UNIQUE-Attribute in relationalen Datenbanken:

- Werte eines UNIQUE-Attributs nur **innerhalb einer Tabelle eindeutig**, nicht über alle Tabellen hinweg.
- ID-Attribute sind dagegen eher mit Surrogaten in objektorientierten Datenbankmodellen vergleichbar.

Diskussion: Fremdschlüsselattribute

IDREF-Attribute vs. Fremdschlüsselattribute:

- Fremdschlüsselattribute beziehen sich auf **genau eine** andere Tabelle.
- Ziele der Referenzen in einem IDREF-Attribut können dagegen **verschiedene** Elementtypen sein.
- Eventuell gewünschte Einschränkungen der Zielelementtypen können in DTD nicht ausgedrückt werden (aber in XML-Schema).

Diskussion: Konsistenzkriterien

DBS vs. XML-Dateien:

- Datenbanken: Fremdschlüsseldeklaration **verhindert** Änderungen, die zu Inkonsistenzen führen (DB bleibt immer korrekt).
- XML-Dateien: kein proaktiver Schutz, nur **Gültigkeitstest beim Laden** einer XML-Datei.
- Problem: Test erfordert DTD, DTD ist aber oft nicht vorhanden.

Abschließendes Beispiel

XML-Dateien zur Studienplanung:

- 1. Datei: Persönliche Studienplanung.
- 2. Datei: Moduldaten verfügbarer Lehrangebote.
- 3. Datei: “Hauptprogramm” zur Integration der Datenquellen und deren Transformation zur Generierung einer HTML-Darstellung.

Beispiel: Datei mit „Hauptprogramm“

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

<!DOCTYPE anzeige_studienplanung [
   <!ELEMENT anzeige_studienplanung ANY >
   <!ENTITY moduldaten
      SYSTEM „myurl/moduldaten.xml" >
   <!ENTITY studienplanung
      SYSTEM "MEINESTUDIENPLANUNG.xml" >
]>
<?xml-stylesheet type="text/xsl"
  href=„myurl/gen_planung_bs_inf.xslt" ?>

<anzeige_studienplanung>
&moduldaten;
&studienplanung;
</anzeige_studienplanung>
                          <!-- Ende anzeige_studienplanung.xml -->
```

Beispiel: Datei mit Studienplanung (1/3)

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

<studienverlauf
    name="NN"
    matikelnummer="XYZ"
    email="xyz@student.uni-siegen.de"
    stand="2015-xx-xx"
    studiengang="BS Informatik mit Vertiefung Softwaretechnik (PO 2012)"
    studiengangskrzl="bs_inf"
    mentor=„Dijkstra"
>
.....

<studienbereich name="Hauptfach Informatik" >
```

Beispiel: Datei mit Studienplanung (2/3)

```
    <modulgruppe name="Pflichtmodule 1. + 2. Semester" LP="60" >
        <modul krzl="DMI" semester="2014w" />
        <modul krzl="DuR" semester="2014w" />
        <modul krzl="AuD" semester="2014w" />
    ....
    </modulgruppe>

    <modulgruppe name="weitere Pflichtmodule" LP="35" > ....
    <modulgruppe name="Wahlpflichtblock Vertiefungspraktikum“ LP="5" > ....
    <modulgruppe name="Wahlpflichtblock Kernfächer" LP="35" > ....
</studienbereich>
```

Beispiel: Datei mit Studienplanung (3/3)

```
<studienbereich name="Vertiefung Softwaretechnik" >
  <modulgruppe name="Pflichtmodule Vertiefung Softwaretechnik"
                          LP="20…>
    <modul krzl="STP-1" semester="???" />
    <modul krzl="STP-2" semester="???" />
    <modul krzl="SP_I" semester="???" />
  </modulgruppe>

  <modulgruppe name="Wahlpflichtmodule Vertiefung Softwaretechnik„
                          LP=…“>
    <modul krzl="CSP" semester="???" />
    <modul krzl="ST_II" semester="???" />
    ....
  </modulgruppe>
</studienbereich>
</studienverlauf>
```

Prüfungsstoff

- Grundlegende Konzepte der XML kennen, verstehen und anwenden können.
- Aufbau einer XML-Datei kennen (Syntax).
- XML-Dateien verstehen, erklären und selbst erstellen können.
- Vergleich XML-Datei vs. DBS diskutieren können.