Rückblick

© K. Schild, 2006/M.Mochol, 2007

Block XML

Vorlesungen – 6 Termine	Übung – 5 Termine
XML-Grundlagen einsch. Namenräume	XML-Syntax, Namensräume
DTD & XML-Schema	DTD
XML-Schema im Detail	XML Schema
SAX & DOM Parser	
XSLT	XPath, XSLT
XML und Datenbanken	XML & Datenabanken

© K. Schild, 2006/M.Mochol, 2007

Block Web Services

Vorlesungen – 4 Termine	Übung – 2 Termine
Web Services, RPCs vs. Messaging	
SOAP im Detail	SOAP
WSDL im Detail	WSDL
Web Services in der Praxis & Ausblick	

Block Wiederholung + Klausur

Vorlesungstermin	Vorlesung
11.07.	Rückblick + (14:00-16:00) Sprechstunde vor der Klausur, Fabeckstr. 15
18.07.	Klausur

© K. Schild, 2006/M.Mochol, 2007

1. BLOCK XML

© K. Schild, 2006/M.Mochol, 2007

1.1 XML

© K. Schild, 2006/M.Mochol, 2007

Warum reicht HTML nicht aus?

immer häufiger medienneutrale Darstellung nötig:

- Vielfalt von Endgeräten und Bandbreiten macht Trennung Inhalt von Präsentation nötig
- Austausch von Daten und Dokumenten zwischen Computern
  - ⇨ z.B. Übermittlung eines Bestellformulars
  - ⇨ z.B. Web Services

HTML: *keine* layoutunabhängige Darstellung von Inhalten

XML

- Extensible Markup Language
- generische Auszeichnungssprache (*generalized markup language*)
  - *keine* Tags vorgegeben, beliebige Tags erlaubt
  - Vorteil: beliebige Metainformationen darstellbar
  - Nachteil: Bedeutung der Metainformationen (Tags) offen
  - Beispiele: SGML und XML

- Unterschiede zu HTML:
  - medienneutral
  - Tag-Namen `<name>…</name>` beliebig

© K. Schild, 2006/M.Mochol, 2007

XML ist …

- eine Methode, um strukturierte Daten in einer Textdatei darzustellen
- Text, aber nicht zum Lesen.
- eine Familie von Technologien
- lizenzfrei und plattformunabhängig
- ein offener Standard, der sich weit verbreitet hat
- ein Protokoll zur Aufnahme und Verwaltung von Informationen
- eine Philosophie für den Umgang mit Informationen
- ein Werkzeug für die Speicherung von Dokumenten
- ein konfigurierbares Medium
- neu, aber nicht so neu
- Modular

XML

- sieht ein wenig wie HTML aus
- überführt HTML in XHTML
- erlaubt Strukturieren von Inhalten

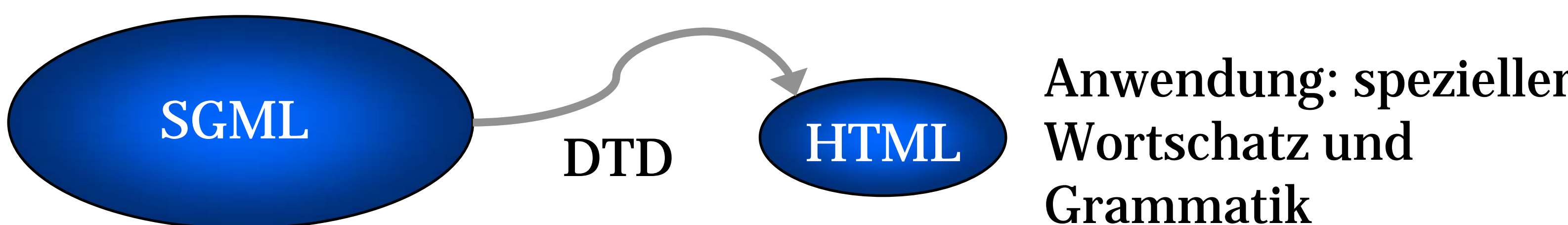
- lässt sich mit Stylesheets kombinieren um Dokumente in einer bestimmten Form darzustellen

- mit klarem und einfachem Syntax und eindeutigen Strukturen kann von Menschen und Maschinen verstanden werden

© K. Schild, 2006/M.Mochol, 2007

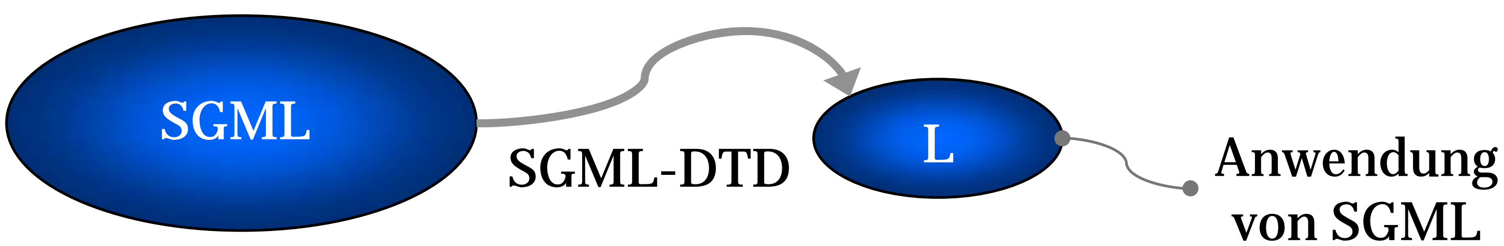
SGML

- Standard Generalized Markup Language
- *keine* vorgegebenen Tags, auch keine für das Layout von Dokumenten
- Vorgänger von XML
- Anwendungen von SGML → mit Document Type Definitions (DTDs) können spezielle Auszeichnungssprachen mit konkreten Tags definiert werden:
  - bekannteste Anwendung von SGML: HTML

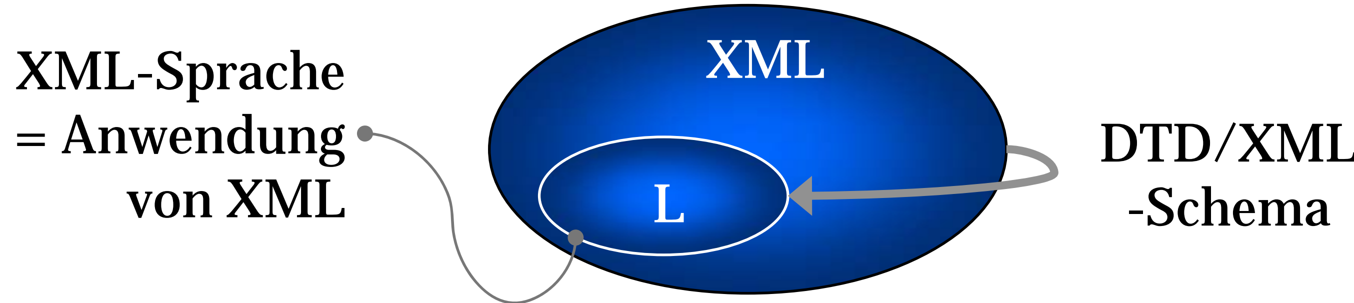


- Anwendung selbst kann keine Anwendung definieren

SGML- vs. XML-Anwendungen

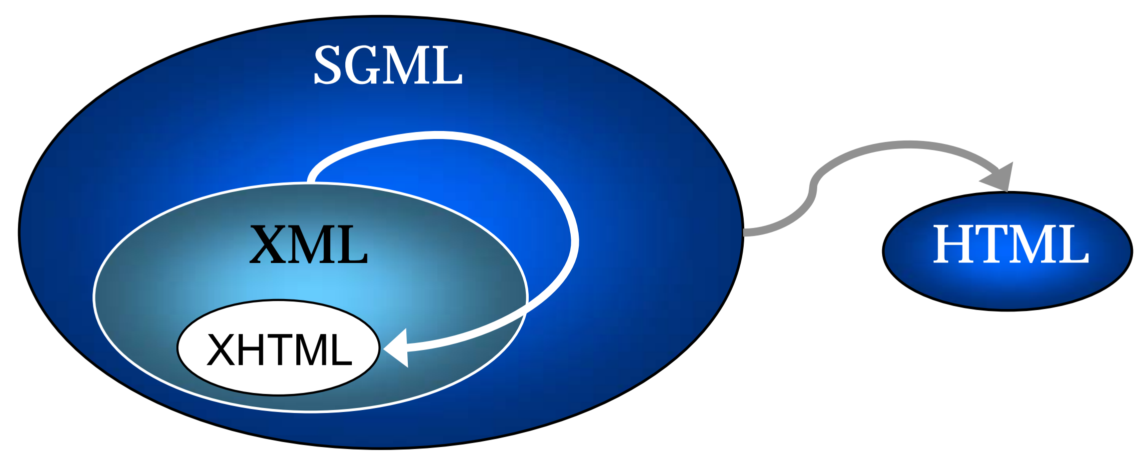


- L muss *nicht* Teilsprache von SGML sein.
- L kann *keine* neue Sprache definieren.
- <u>Beispiel</u>: HTML



- L *immer* Teilsprache von XML
- L kann *keine* neue Sprache definieren.
- <u>Beispiel</u>: XHTML

SGML, HTML, XML, XHTML?!



HTML

- Anwendung von SGML

XML

- Teilsprache von SGML

XHTML

- XML-Sprache
- = Anwendung von XML
- alle XHTML-Dokumente immer wohlgeformte XML-Dokumente

© K. Schild, 2006/M.Mochol, 2007

Grundbausteine von XML

- Elemente: strukturieren das XML-Dokument
- Attribute: Zusatzinformationen zu Elementen
- XML-Deklaration: Informationen für Parser

```
<?xml version="1.0" encoding="UTF-8"?>
<name id="1232345">
  <first>John</first>
  <middle>Fitzgerald Johansen</middle>
  <last>Doe</last>
</name>
```

- Namensräume: lösen Namenskonflikte auf und geben Elemente eine Bedeutung

© K. Schild, 2006/M.Mochol, 2007

XML-Elemente Inhalt

1. unstrukturierter Inhalt:
   einfacher Text ohne Kind-Elemente
2. strukturierter Inhalt:
   Sequenz von > Ø Kind-Elementen
3. gemischter Inhalt:
   enthält Text mit mind. einem Kind-Element
4. leerer Inhalt

© K. Schild, 2006/M.Mochol, 2007

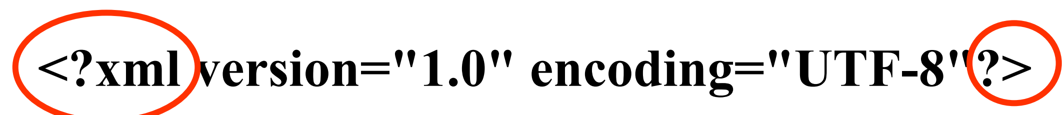
XML-Attribute

```
<name id="1232345" nickname="Shiny John">
  <first>John</first>
   <last>Doe</last>
</name>
```

- **Attribut**: Name-Wert-Paar
  - name="wert" oder name='wert' aber ~~name= "wert'~~
- **Attribut-Wert**:
  - immer PCDATA: keine Kind-Elemente, kein < und &
  - "we"rt" und 'we'rt' ebenfalls nicht erlaubt
  - Normalisierung: u.a. Zeilenumbruch ➔ #xA
- Beachte: Reihenfolge der Attribute belanglos

© K. Schild, 2006/M.Mochol, 2007

XML-Deklaration



Attribut version

- verwendete XML-Version: "1.0" oder "1.1"
- obligatorisch

Attribut encoding

- Kodierung der XML-Datei
- optional

Attribut standalone

- Gibt an, ob es eine zugehörige DTD oder ein XML-Schema gibt ("no") oder nicht ("yes").
- optional

Beachte: immer in dieser Reihenfolge!

© K. Schild, 2006/M.Mochol, 2007

Wohlgeformte XML-Dokumente

1. Jedes Anfangs-Tag muss ein zugehöriges Ende-Tag haben.
2. Elemente dürfen sich nicht überlappen.
3. XML-Dokumente haben genau ein Wurzel-Element.
4. Element-Namen müssen bestimmten Namenskonventionen entsprechen.
5. XML beachtet grundsätzlich Groß- und Kleinschreibung.
6. XML belässt White Space im Text.
7. Ein Element darf niemals zwei Attribute mit dem selben Namen haben.

© K. Schild, 2006/M.Mochol, 2007

Namensräume

{ course:course, course:title, course:abstract, course:lecturers, course:date }

{ pers:name, pers:title, pers:first, pers:last }

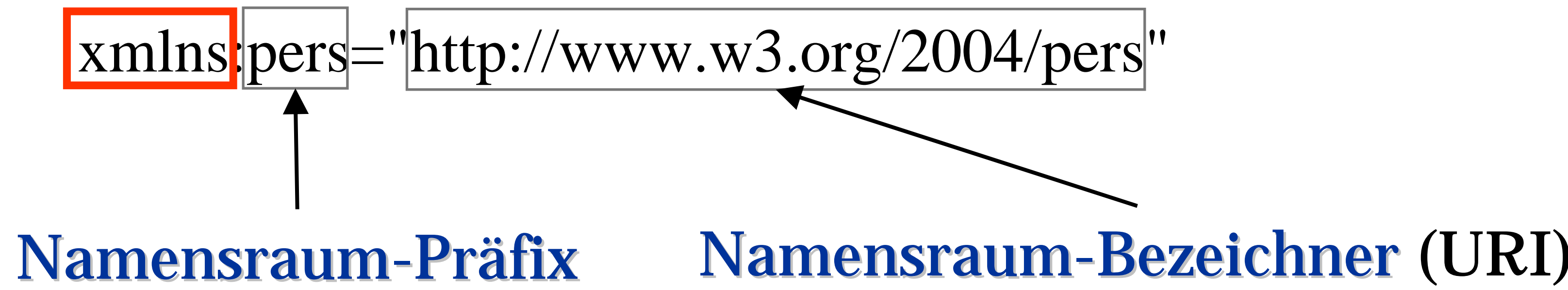
Namensraum (namespace):

- alle Bezeichner mit identischen Anwendungskontext
- Namensräume müssen eindeutig identifizierbar sein.

© K. Schild, 2006/M.Mochol, 2007

Namensräume in XML

- WWW: Namensräume müssen global eindeutig sein.
- In XML wird Namensraum mit URI identifiziert.
- Zuerst wird Präfix bestimmter Namensraum zugeordnet, z.B.:



- Anschließend kann das Namensraum-Präfix einem Namen vorangestellt werden: z.B. pers:title
- <u>Beachte</u>: Wahl des Präfixes egal!

© K. Schild, 2006/M.Mochol, 2007

Standard-Namensraum

- xmlns="URI" statt xmlns:prefix="URI"
- Namensraum-Präfix kann weggelassen werden.

- Standard-Namensraum gilt für das Element, wo er definiert ist.
- Kind-Elemente erben Standard-Namensraum von ihrem Eltern-Element.
- Ausnahme: Standard-Namensraum wird überschrieben
- Beachte: Standardnamensräume gelten nicht für Attribute

© K. Schild, 2006/M.Mochol, 2007

Qualified vs. Unqualified

- Element- oder Attribut-Name heißt **namensraumeingeschränkt** (**qualified**), wenn er einem Namensraum zugeordnet ist.

- Einschränkung vom Element-Namensraum:
  1. Standard-Namensraum festlegen
  2. Namensraum-Präfix voranstellen

- Einschränkung vom Attribut-Namensraum:
  1. Namensraum-Präfix voranstellen

© K. Schild, 2006/M.Mochol, 2007

XML → Lernziele

Lernziele

- Was ist eine generische Auszeichnungssprache?
- Vor- und Nachteile einer generischen Auszeichnungssprache
- Unterschiede zwischen SGML, HTML, XML und XHTML
- Grundbausteine von XML
- Funktionen von Namensräumen
- Wie werden Elementen/Attributen in XML Namensräume zugeordnet?
- Syntaxregel und Wohlgeformtheit von XML
- Was ist W3C & W3C Recommendation?

© K. Schild, 2006/M.Mochol, 2007

1.2 DTD & XML Schema

(DTD)

© K. Schild, 2006/M.Mochol, 2007

DTDs vs. XML-Schema

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.

© K. Schild, 2006/M.Mochol, 2007

Die DTD für das Beispiel-Dokument

```
<!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)>
```

<!ELEMENT *Name Content-Modell*>

Element-Deklaration

© K. Schild, 2006/M.Mochol, 2007

Datentypen für Element-Inhalte

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.

© K. Schild, 2006/M.Mochol, 2007

Operatoren

- + bezeichnet n Wiederholungen mit $n > 0$.
- * bezeichnet n Wiederholungen mit $n \geq 0$.
- | bezeichnet **Auswahl**: genau eine der beiden Alternativen
- , bezeichnet **Sequenz** von Elementen.
- ? bedeutet optional
- ( ) fassen den Kontent zusammen, auf die sich ein nachfolgender Operator bezieht

Beachte:

- Rekursive Deklaration nicht äquivalent zur iterativen Definition!
- (fast) beliebige Verschachtelung von Sequenz, Auswahl |, ?, *, + und Rekursion erlaubt

© K. Schild, 2006/M.Mochol, 2007

Deklaration von Attributen

<!ATTLIST BookStore
version CDATA #IMPLIED>

<!ATTLIST *Name*
AttrName1 AttrTyp1 Attrbeschr1
AttrName2 AttrTyp2 Attrbeschr2
>

Attribut-Deklarationen

© K. Schild, 2006/M.Mochol, 2007

Deklaration von Attributen

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

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

- Element **`BookStore`** hat Attribut **`version`**.
- CDATA: Attribut-Wert = String ohne <, & und ' bzw. "

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

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

© K. Schild, 2006/M.Mochol, 2007

Datentypen für Attribute

Zusätzlich zu CDATA (Strings) und Aufzählungstypen:

- NMTOKEN: String (Namenskonventionen von XML)
- ID: eindeutiger Bezeichner (Namenskonventionen von XML)
- IDREF: Referenz auf einen eindeutigen Bezeichner

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

- Wert von **`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.

© K. Schild, 2006/M.Mochol, 2007

Optionale/erforderliche Attribute

```
<!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

© K. Schild, 2006/M.Mochol, 2007

1.2 DTD & XML Schema
(XML Schema)

© K. Schild, 2006/M.Mochol, 2007

XML-Schema: Überblick

- eine XML basierte Alternative zu DTD
- beschreibt die Struktur eines XML Dokuments
- eine W3C Recommendation
- statt XML Schema wird oft die Abkürzung XSD (XML Schema Definition) benutzt
- definiert
  - Elemente/Attribute , die im Dokument vorkommen dürfen
  - die Reihenfolge & Anzahl der (Kinder-)Elemente
  - wie Inhalt eines Element auszusehen hat
  - Datentypen für Elemente und Attribute

© K. Schild, 2006/M.Mochol, 2007

XML-Schema vs. DTD (Auszug)

- Für jede DTD gibt es ein äquivalentes XML-Schema.
- Umgekehrt gibt es jedoch XML-Schemata, für die es keine äquivalente DTD gibt.

➔XML-Schemata ausdrucksmächtiger als DTDs

- XML-Schemata sind wohlgeformte XML-Dokumente.

- Wurzel-Element: Schema aus W3C-Namensraum http://www.w3.org/2001/XMLSchema
  - hier XML-Schema für XML-Schema hinterlegt: Schema der Schemata

© K. Schild, 2006/M.Mochol, 2007

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
- 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.
- Beachte: DTD definiert keinen Namensraum

Verschiedene Arten von Datentypen

Unterteilung bzgl. Komplexität
- einfache
- komplexe

Unterteilung bzgl. Gültigkeit
- benannte
- anonyme

Unterteilung bzgl. Ableitung
- abgeleitete
- primitive

} Datentypen

Einfache vs. komplexe Datentypen

einfache Datentypen (simple types)

- beschreiben unstrukturierten Inhalt ohne Elemente oder Attribute (PCDATA)

komplexe Datentypen (complex types)

- beschreiben strukturierten XML-Inhalt mit Elementen oder Attributen
- natürlich auch gemischten Inhalt

© K. Schild, 2006/M.Mochol, 2007

Anonyme vs. benannte Datentypen

```
<xsd:element name="BookStore">
    <xsd:complexType>
        Liste von Büchern
    </xsd:complexType>
</xsd:element>
```

- anonymer komplexer Datentyp
- lokale Deklaration

```
<xsd:complexType name="BookStoreType">
        Liste von Büchern
</xsd:complexType>
```

- benannter komplexer Datentyp
- globale Deklaration
- wiederverwendbar

© K. Schild, 2006/M.Mochol, 2007

Element-Deklaration: 1. Möglichkeit

Freie Universität Berlin

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

```
<xsd:element name="name" type="type“ minOccurs="int" maxOccurs="int"/>
```

- **name**: Name des deklarierten Elementes
- **type**: Datentyp (benannt oder vordefiniert)
- **minOccurs**: so oft erscheint das Element mindestens (nicht-negative Zahl)
- **maxOccurs**: so oft darf das Element höchstens erscheinen (nicht-negative Zahl oder unbounded).
- Default-Werte von minOccurs und maxOccurs jeweils 1
- Beachte: abhängig vom Kontext gibt es Einschränkungen von minOccurs und maxOccurs

© K. Schild, 2006/M.Mochol, 2007

Element-Deklaration: 2. Möglichkeit

anonymer Datentyp kann entweder

- komplex oder

```
<xsd:element name="name" minOccurs="int" maxOccurs="int">
    <xsd:complexType>
        ...
    </xsd:complexType>
</xsd:element>
```

- einfach sein

```
<xsd:element name="name" minOccurs="int" maxOccurs="int">
    <xsd:simpleType>
        ...
    </xsd:simpleType>
</xsd:element>
```

© K. Schild, 2006/M.Mochol, 2007

Attributen -Deklaration

- ähnlich wie Elemente
- aber nur **einfache Datentypen** erlaubt
- Deklaration mit **benanntem Datentyp**:

```
<xsd:attribute name= "name" type= "type" />
```

globaler Typ

- oder mit **anonymem Datentyp**:

```
<xsd:attribute name= "name">
    <xsd:simpleType>
     ...
    </xsd:simpleType>
</xsd:attribute>
```

lokaler Typ

global: Deklaration Kind von xsd:schema

lokal: Deklaration kein direktes Kind von xsd:schema

© K. Schild, 2006/M.Mochol, 2007

Attributen-Deklaration

<xsd:attribute name= "*name*" type= "*type*" use="*use*"
default= "*value*" />

- use="optional" Attribut optional
- use="required" Attribut obligatorisch
- use="prohibited" Attribut unzulässig
- Beachte: Wenn nichts anderes angegeben, ist das Attribut optional!
- default: Standard-Wert für das Attribut
- fixed: Standard-Fix-Wert für das Attribut

© K. Schild, 2006/M.Mochol, 2007

Einfache Datentypen

einfache Datentypen (simple types)

- beschreiben unstrukturierten Inhalt ohne Elemente oder Attribute (PCDATA)
- Schema der Schemata definiert 44 einfache Datentypen
- eigene einfache Datentypen können definiert werden

© K. Schild, 2006/M.Mochol, 2007

Primitive vs. abgeleitete einfache Datentypen

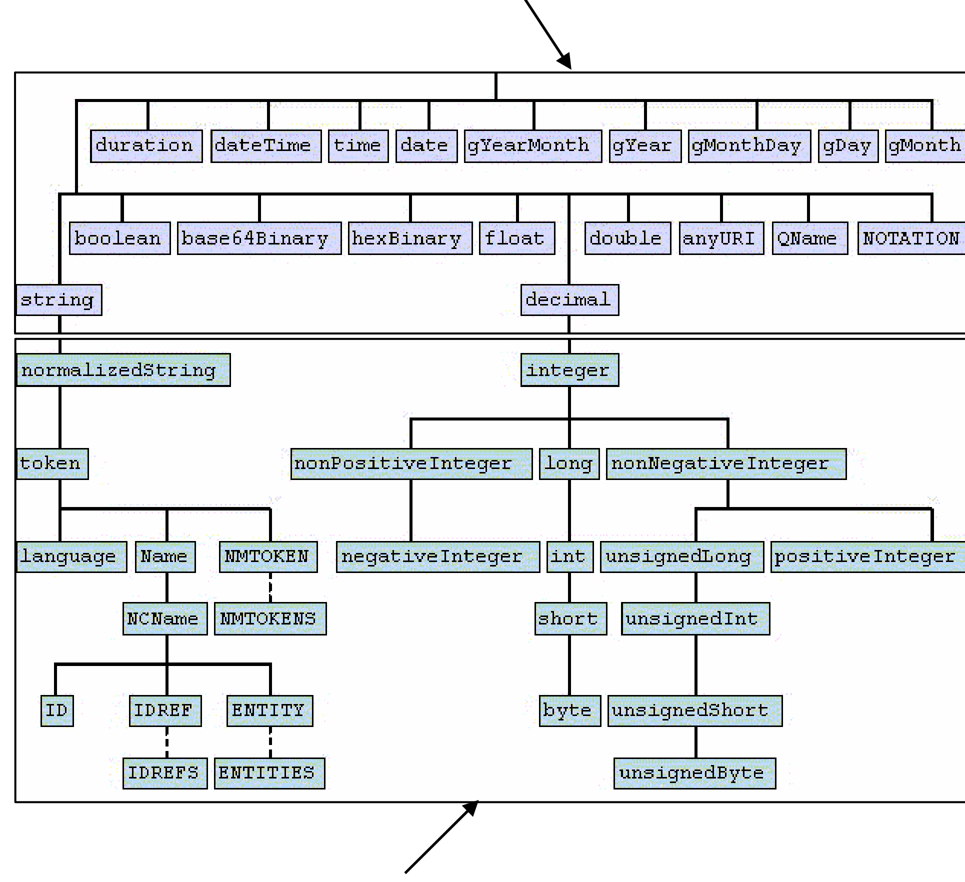
primitive Datentypen
(primitive types)

- nicht von anderen Datentypen abgleitet

abgeleitete Datentypen
(derived types)

- auf Basis von anderen Datentypen definiert, z.B. durch Einschränkung oder Erweiterung

Primitive einfache Datentypen



Abgeleitete einfache Datentypen

© K. Schild, 2006/M.Mochol, 2007

Abgeleitete einfache Datentypen

Einschränkung (Teilmenge)
Einschränkung des Wertebereiches eines einfachen Datentyps

```
<xsd:simpleType name="longitudeType">
 <xsd:restriction base="xsd:integer">
  <xsd:minInclusive value="-180"/>
  <xsd:maxInclusive value="180"/>
 </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:simpleType name="MyInteger">
 <xsd:union>
  <xsd:simpleType>
   <xsd:restriction base="xsd:integer"/>
  </xsd:simpleType>

  <xsd:simpleType>
   <xsd:restriction base="xsd:string">
    <xsd:enumeration value="unknown"/>
   </xsd:restriction>
  </xsd:simpleType>
</xsd:union>
</xsd:simpleType>
```

Vereinigung
Vereinigung der Wertebereiche mehrerer einfacher Datentypen

```
<xsd:simpleType name="IntegerList">
 <xsd:list itemType="xsd:integer"/>
</xsd:simpleType>
```

Listen
Liste als String (PCDATA): einzelne Elemente durch White Spaces getrennt

© K. Schild, 2006/ M.Mochol, 2007

Komplexe Datentypen

komplexe Datentypen (complex types)

- beschreiben strukturierten XML-Inhalt mit Elementen oder Attributen
- natürlich auch gemischten Inhalt

Elemente mit komplexen Typen können andere Elemente und/oder Attribute enthalten.

© K. Schild, 2006/M.Mochol, 2007

Komplexen Datentypen BILDEN

1. Sequenz <xsd:sequence>...< /xsd:sequence>
   - **Reihenfolge vorgegeben**
   - Elemente erscheinen so oft, wie mit minOccurs/maxOccurs festgelegt.
2. Menge <xsd:all>...</xsd:all>
   - **Reihenfolge der Elemente beliebig**
   - Jedes Element erscheint hier genau einmal.
3. Auswahl <xsd:chioce>...< /xsd:choice>
   - Inhalt besteht aus **genau einem** der aufgezählten Alternativen.

Beachte: Alle Operatoren können minOccurs und maxOccurs selbst spezifizieren

© K. Schild, 2006/M.Mochol, 2007

Komplexe Datentypen ABLEITEN

1. **Erweiterung**

- Datentyp wird durch zusätzliche Attribute und Elemente erweitert.
- Ergebnis: immer komplexer Datentyp

Basis-Datentyp (einfach oder komplex)	+	zusätzliche Attribute oder Elemente	=	erweiterter Datentyp (immer komplex)

2. **Teilmenge**

- Einschränkung des Wertebereiches eines Datentyps
- Resultierender Datentyp darf nur gültige Werte des ursprünglichen Datentyps enthalten (echte Teilmenge).
- hier wäre z.B. xsd:string statt xsd:unsignedShort nicht erlaubt: xsd:string keine Teilmenge von xsd:nonNegativeInteger

© K. Schild, 2006/M.Mochol, 2007

Typsubstitution

- Voraussetzung: XML-Schema S leitet Datentyp t' von Datentyp t ab:

  entweder mit xsd:extension oder xsd:restriction
- Betrachten wir eine Instanz von S.

Typsubstitution

- An jeder Stelle in der Instanz, wo S den Datentyp t verlangt, kann auch t' verwendet werden.
- Verwendete Datentyp t' muss mit xsi:type explizit angegeben werden.

© K. Schild, 2006/M.Mochol, 2007

Mögliche Probleme

t' Teilmenge (restriction) von t

- Laut Schema S müssen Anwendungen sowieso mit allen gültigen Werten von t umgehen, also auch mit t'.

⇨ unproblematisch

t' Erweiterung (extension) von t

- Laut Schema S müssen Anwendungen mit allen gültigen Werten von t umgehen, nicht aber mit zusätzlichen Attributen und Elementen

⇨ evtl. problematisch

⇨ Typsubstitution für Erweiterungen evtl. unterdrücken:

```
<xsd:element name="name" type="NameType" block="extension">
```

© K. Schild, 2006/M.Mochol, 2007

DTD & XML Schema → Lernziele

DTDs und XML-Schemata

Lernziele

- DTDs und XML-Schemata lesen und vestehen können
- Vorteile von XML-Schemata gegenüber DTDs
- Was ist ein Ziel-Namensraum?
- Verschiedene Arten von Datentypen in XML Schema
- Wie kann man Datentypen in XML Schema bilden/ableiten?
- Was bedeutet Typsubstitution in Schema-Instanzen und welche Probleme ergeben sich hierdurch?

© K. Schild, 2006/M.Mochol, 2007

1.3 XML Parser

Kategorien von Parser

Validierender vs. nicht-validierender Parser

- Wird die Validität des Dokumentes untersucht?

Pull- vs. Push-Parser

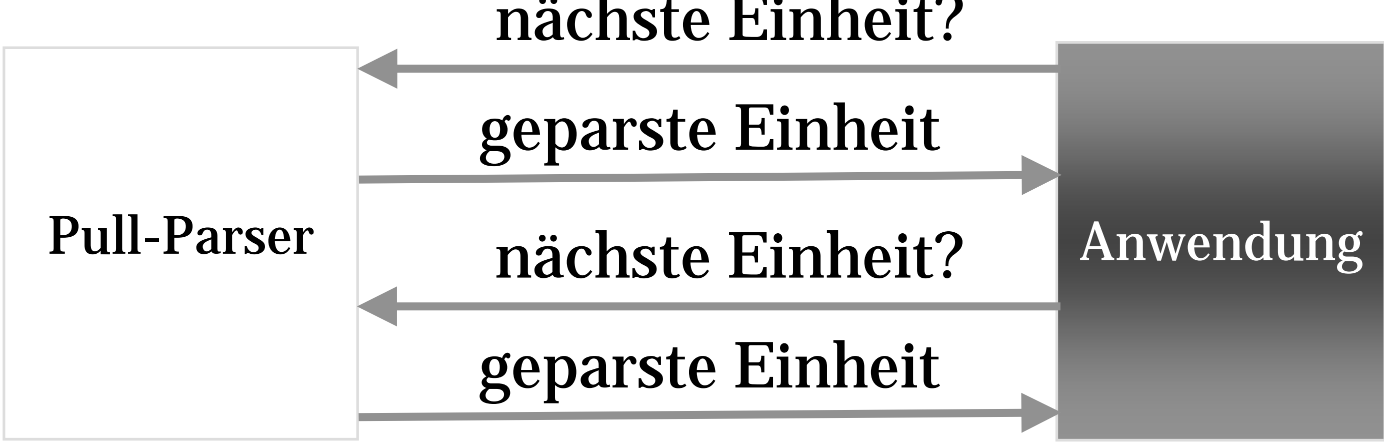
- Wer hat Kontrolle über das Parsen: die Anwendung oder der Parser?

Einschritt- vs. Mehrschritt-Parser

- Wird das XML-Dokument in einem Schritt vollständig geparst oder Schritt für Schritt?

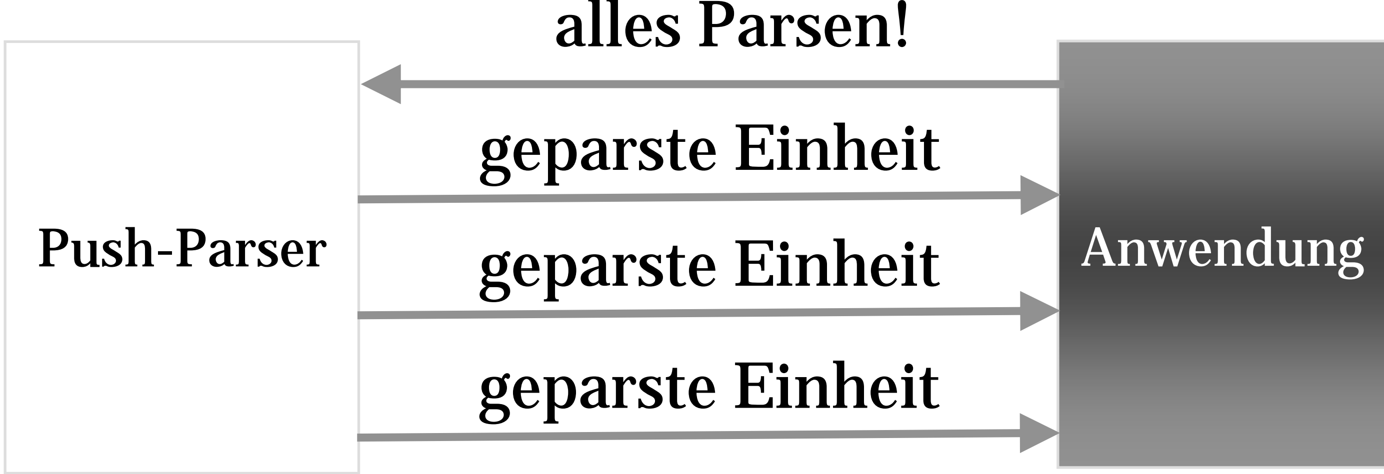
- Beachte: Kategorien unabhängig voneinander, können kombiniert werden

Pull-Parser



- Anwendung hat Kontrolle über das Parsen.
- Analyse der nächsten syntaktischen Einheit muss aktiv angefordert werden.
- Beachte: „Pull" aus Perspektive der Anwendung.

Push-Parser

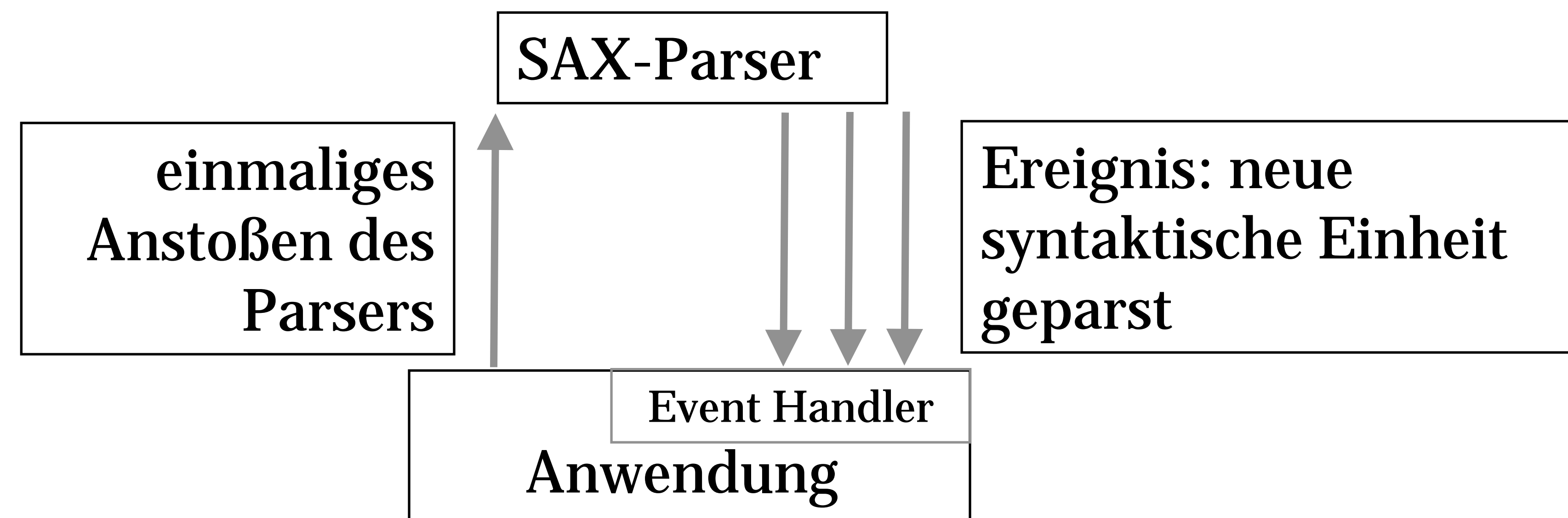


- Parser hat Kontrolle über das Parsen.
- Sobald der Parser eine syntaktische Einheit analysiert hat, übergibt er die entsprechende Analyse.
- Beachte: „Push” aus Perspektive der Anwendung.

© K. Schild, 2006/M.Mochol, 2007

SAX: Simple API for XML

- Mehrschritt-Push-Parser für XML
- kein W3C-Standard, sondern *de facto* Standard
- standardisiertes API
- ursprünglich nur Java-API, inzwischen auch: C, C++, VB, Pascal, Perl



Callback-Methoden

- Methoden des Event-Handlers (also der Anwendung), die vom Parser aufgerufen werden
- für jede syntaktische Einheit eigene Callback-Methode, u.a.:
  - startDocument **und** endDocument
  - startElement **und** endElement
  - characters
  - processingInstruction

DefaultHandler

- Standard-Implementierung der Callback-Methoden: tun jeweils nichts!
- können natürlich überschrieben werden

Beispiel

```
<priceList>
   <coffee>
        <name>
              Mocha Java
        </name>
        <price>
              11.95
        </price>
   </coffee>
</priceList>
```

Parser ruft startElement(...,priceList,...) auf.
Parser ruft startElement(...,coffee,...) auf.
Parser ruft startElement(...,name,...) auf.
Parser ruft characters("Mocha Java",...) auf.
Parser ruft endElement(...,name,..) auf.
Parser ruft startElement(...,price,...) auf.
Parser ruft characters("11.95",...) auf.
Parser ruft endElement(...,price,...) auf.
Parser ruft endElement(...,coffee,...) auf.
Parser ruft endElement(...,priceList,...) auf.

- Ereignisfluss: Sobald Einheit geparst wurde, wird Anwendung benachrichtigt.
- Beachte: Es wird kein Parse-Baum aufgebaut!

© K. Schild, 2006/M.Mochol, 2007

Logik der Callback-Methoden

```
<priceList>
  <coffee>
      <name>
            Mocha Java
      </name>
      <price>
            11.95
      </price>
  </coffee>
</priceList>
```


- Zustände als boolesche Variablen

© K. Schild, 2006/M.Mochol, 2007

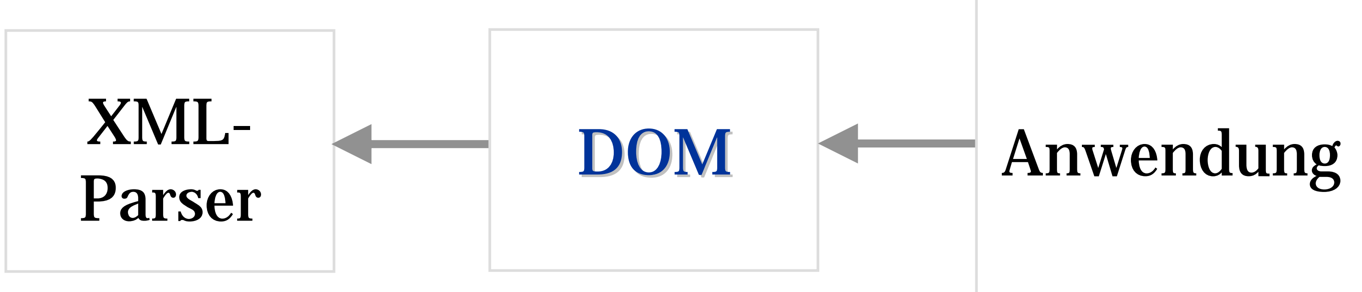
Vor- und Nachteile von SAX

+ sehr effizient und schnell, auch bei großen XML-Dokumenten
+ relative einfach

- Kontext (Parse-Baum) muss von Anwendung selbst verwaltet werden.
- abstrahiert nicht von XML-Syntax
- nur Parsen möglich, keine Modifikation oder Erstellung von XML-Dokumenten

© K. Schild, 2006/M.Mochol, 2007

Document Object Model (DOM)

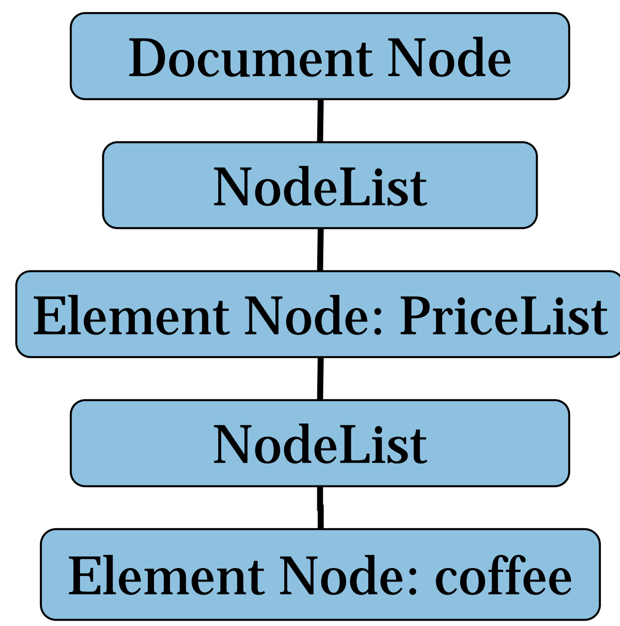


- streng genommen kein Parser, sondern abstrakte Schnittstelle zum Zugreifen, Modifizieren und Erstellen von Parse-Bäumen
- W3C-Standard
- unabhängig von Programmiersprachen
- nicht nur für XML-, sondern auch für HTML-Dokumente
- im Ergebnis aber Einschritt-Pull-Parser

© K. Schild, 2006/M.Mochol, 2007

DOM-Parse-Bäume

```
<?xml version="1.0" ?>
<priceList>
 <coffee>
  <name>Mocha Java</name>
  <price>11.95</price>
 </coffee>
</priceList>
```



- Beachte: Dokument-Wurzel (Document Node) ≠ priceList
- Document Node: virtuelle Dokument-Wurzel, um z.B. version="1.0" zu repräsentieren
- Document Node und Element Node immer NodeList als Kind

© K. Schild, 2006/M.Mochol, 2007

DOM-Parse-Bäume

```
<?xml version="1.0" ?>
<priceList>
 <coffee>
   <name>Mocha Java</name>
   <price>11.95</price>
 </coffee>
</priceList>
```

- Beachte: PCDATA wird als eigener Knoten dargestellt.

© K. Schild, 2006/M.Mochol, 2007

Vor- und Nachteile von DOM

- \+ Kontext (Parse-Baum) muss nicht von Anwendung verwaltet werden.
- \+ einfache Navigation im Parse-Baum
- \+ direkter Zugriff auf Elemente über ihre Namen
- \+ nicht nur Parsen, sondern auch Modifikation und Erstellung von XML-Dokumenten

- – speicherintensiv
- – abstrahiert nicht von XML-Syntax

© K. Schild, 2006/M.Mochol, 2007

SAX oder DOM?

SAX	DOM
ereignis-orientierter Ansatz	**modell-orientierter** Ansatz
	vollständige Umsetzung in eine **Baumstruktur**
parsen	mehrere Verarbeitungsmöglichkeiten
XML-Dokument als **Eingabestrom** (Streaming-Verfahren)	XML-Dokument **vollständig im Speicher** (Baummodell des Dokuments)
schnelle Verarbeitung von großen XML-Dokumenten	**langsame** Verarbeitung von großen XML-Dokumenten
wenig Hauptspeicher benötigt	**mehr** Hauptspeicher benötigt
	nach dem Einlesen kann auf alle Teilstrukturen des XML-Dokuments zugegriffen werden

© K. Schild, 2006/M.Mochol, 2007

XML Parser → Lernziele

© K. Schild, 2006/M.Mochol, 2007

XML-Parser

Lernziele

- Kategorien von Parser
- Wie arbeitet ein SAX-Parser?
- Was ist ein DOM-Parser?
- Vor- und Nachteile von SAX- und DOM-Parser
- Welchen Parser für welche Anwendung?

© K. Schild, 2006/M.Mochol, 2007

1.4 XSLT

© K. Schild, 2006/M.Mochol, 2007

XPath: absolute und relative Pfade

- Standard zum Zugreifen beliebiger Teile eines XML-Dokumentes

absolute Pfade

- beginnen mit "/"
- z.B. /order/item

  ➔ lesen: Folge dem Pfad von der Dokument-Wurzel zu einem Kind-Element order und von dort aus zu einem Kind-Elementen item!

relative Pfade

- beginnen mit einem Element oder Attribut
- z.B. order/item

  🡐 lesen: item-Elemente, die Kind eines Elementes order sind
- Element order an beliebiger Stelle des XML-Dokumentes

Filter: Randbedingungen für Pfade

- order/item[@item-id = 'E16-25A']

  item-**Elemente, die Kind von** order **sind und Attribut** item-id **mit Wert** 'E16-25A' **haben**

- **können an beliebiger Stelle in einem Pfad vorkommen:**

  order[@order-id = '4711']/item

Warum XML transformieren?

Trennung Inhalt und Präsentation

- XML trennt Inhalt von Präsentation (Layout)
- Für eine entsprechende Darstellung müssen XML-Inhalte transformiert werden:
  - XML-Inhalt ➔ Layout

Inhaltliche Transformationen

- Daten mit XML repräsentiert
- unterschiedliche Sichten (Views) auf XML-Inhalte erfordern Transformationen:
  - XML-Inhalt ➔ XML-Inhalt

© K. Schild, 2006/M.Mochol, 2007

Was ist XSLT?

- Programmiersprache zur Transformation von XML-Dokumenten
- erlaubt XML-Dokumente in beliebige Textformate zu Transformieren:

  XML ➔ XML/HTML/XHTML/WML/RTF/ASCII ...
- XSLT-Programme (stylesheets) haben XML-Syntax

  ➔ plattformunabhängig
- W3C-Standard seit 1999

© K. Schild, 2006/M.Mochol, 2007

Ursprungs- und Ergebnisdokument

```
<?xml version="1.0"?>
<order>
  <salesperson>John Doe</salesperson>
  <item>Production-Class Widget</item>
  <quantity>16</quantity>
  <date>...</date>
  <customer>Sally Finkelstein</customer>
</order>
```

Ursprungsdokument → Ursprungsbaum (source document → source tree)

```
<xsl:template match="order/item">
   <p><xsl:value-of select="."/></p>
</xsl:template>
```

Template

Transformation

```
<p>Production-Class Widget</p>
```

Ergebnisbaum → Ergebnisdokument (result tree → result document)

1. Neue Inhalte erzeugen

- statt üblicher XML-Syntax

```
<xsl:template match="...">
  <p style="color:red">neuer Text</p>
</xsl:template>
```

- auch möglich:

```
<xsl:template match="...">
  <xsl:element name="p">
    <xsl:attribute name="style">color:red</xsl:attribute>
    <xsl:text>neuer Text</xsl:text>
  </xsl:element>
</xsl:template>
```

- nötig, wenn z.B. Name = Variable oder PCDATA = " "

© K. Schild, 2006/M.Mochol, 2007

2. Inhalte übertragen

<xsl:copy-of select="."> Element

- Kopiert aktuellen Teilbaum
- **aktueller Teilbaum**: Baum, der vom aktuellen Knoten aufgespannt wird, einschließlich aller Attribute und PCDATA

<xsl:copy> Element

- Kopiert aktuellen Knoten ohne Kind-Elemente, Attribute und PCDATA

⇨ Kopiert nur Wurzel-Element des aktuellen Teilbaums

<xsl:value-of select="."> Element

- Extrahiert PCDATA, das im aktuellen Teilbaum vorkommt

Beispiel

Stylesheet

```
<xsl:template match="A">
  <xsl:value-of select="@id"/>
</xsl:template>

<xsl:template match="B">
   <xsl:value-of select="@id"/>
 </xsl:template>

 <xsl:template match="C">
   <xsl:value-of select="@id"/>
 </xsl:template>

<xsl:template match="D">
  <xsl:value-of select="@id"/>
 </xsl:template>
```

```
<source>
 <A id="a1">
    <B id="b1"/>
    <B id="b2"/>
 </A>
 <A id="a2">
    <B id="b3"/>
    <B id="b4"/>
    <C id="c1">
       <D id="d1"/>
    </C>
    <B id="b5">
       <C id="c2"/>
    </B>
 </A>
</source>
```

kein Template anwendbar

Template "A" wird angewandt

Template "B" wäre anwendbar, es werden aber keine Templates aufgerufen!

→ a1 a2

© K. Schild, 2006/M.Mochol, 2007

Rekursiver Aufruf aller Templates

<xsl:apply-templates/>

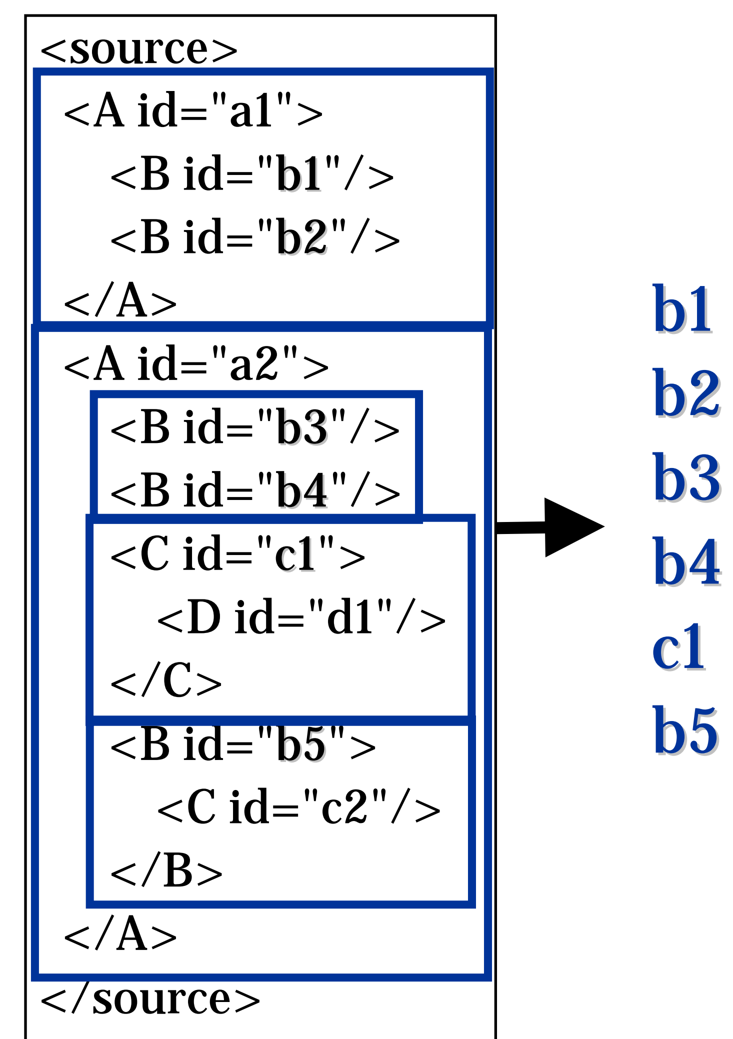
- versucht Templates auf Kinder des aktuellen Knotens anzuwenden
- Kind bedeutet hier: Kind-Element, Text-Knoten oder Attribut-Knoten
- Mit <xsl:apply-templates select = "..."/> auch rekursiver Aufruf an beliebiger Stelle möglich.
- Vorsicht: Terminierung nicht automatisch sichergestellt!
- Beispiel:

```
<xsl:template match="A">
  <xsl:value-of select="@id"/>
  <xsl:apply-templates select="/"/>
</xsl:template>
```

© K. Schild, 2006/M.Mochol, 2007

Iteration statt Rekursion

```
<xsl:template match="A">
  <xsl:for-each select="*">
    <xsl:value-of select="@id"/>
  </xsl:for-each>
</xsl:template>
```



- xsl:value-of **wird auf alle** select-**Pfade der** for-each-**Schleife angewandt.**
- **Beachte**: select-**Pfad von** xsl:for-each **relativ zum Kontext-Knoten des Templates, hier also** "A/*".

© K. Schild, 2006/M.Mochol, 2007

Zwei Vordefinierte Templates

1. **vordefiniertes Template**

- realisiert rekursiven Aufruf des Prozessors, wenn kein Template anwendbar ist

2. **vordefiniertes Template**

- kopiert PCDATA und Attribut-Werte des aktuellen Knotens in das Ergebnisdokument

Leeres Stylesheet

- traversiert gesamtes Ursprungsdokument und extrahiert dabei PCDATA und Attribut-Werte

Überschreiben

- Vordefinierte Templates können durch speziellere Templates überschrieben werden.

© K. Schild, 2006/M.Mochol, 2007

Vor- und Nachteile von XSLT

+

- + plattformunabhäng
- + relativ weit verbreitet
- + Verarbeitung in Web-Browsern
- + Standard-Transformationen (wie XML ➔ HTML) einfach zu realisieren.
- + Nicht nur HTML, sondern beliebige andere Sprachen können erzeugt werden.
- + extrem mächtig

-

- – Entwickler müssen speziell für die Transformation von XML-Dokumenten neue Programmiersprache lernen.
- – Anbindung von Datenbanken umständlich
- – manche komplexe Transformationen nur umständlich zu realisieren

XSLT → Lernziele

XSLT

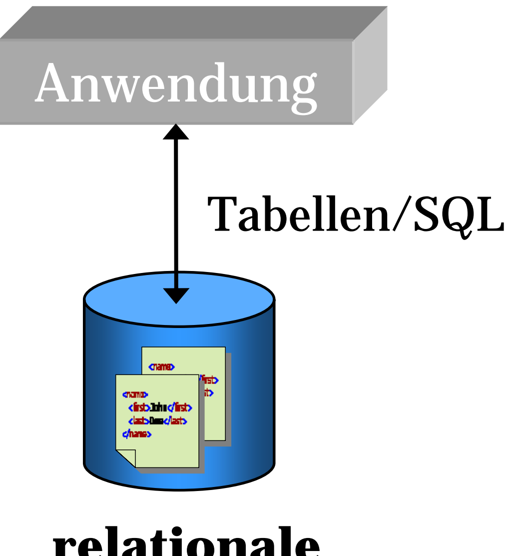
Lernziele

- XPath: XPath-Ausdrücke, XPath-Funktionen
- Warum XML transformieren?
- Was ist XSLT?
- Iteration/Rekursion bei XSLT
- Welche vordefinierten Templates gibt es?
- Vor- und Nachteile von XSLT

1.5 XML & Datenbanken

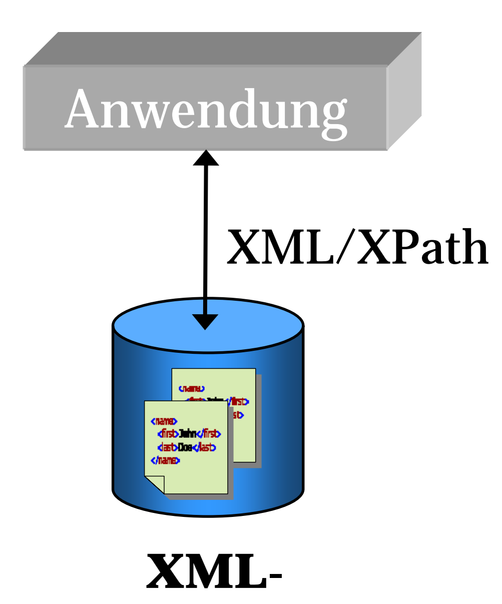
© K. Schild, 2006/M.Mochol, 2007

Wie XML persistent speichern?



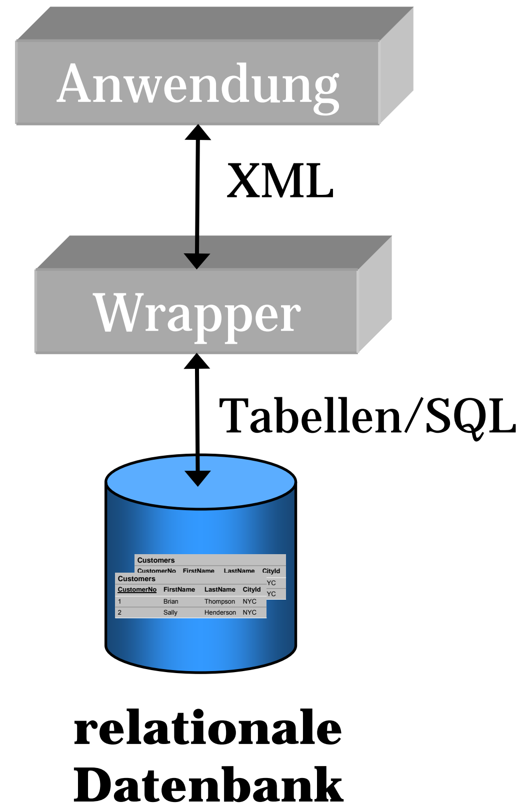
relationale Datenbank

XML als einfachen String speichern



XML-Datenbank

XML als strukturiertes XML-Dokument speichern



relationale Datenbank

XML als Tabellen speichern

© K. Schild, 2006/M.Mochol, 2007

1. XML als String speichern

- hierarchische XML-Struktur als Wert eines Feldes in einer Tabelle speichern
- XML-Struktur als String serialisieren

Vor- und Nachteile

+ vorhandenes relationales Datenbanksystem (RDBMS) kann genutzt werden
+ triviale Schnittstelle zwischen XML und RDBMS

- keine komplexen Anfragen (z.B. mit XPath) auf XML-Struktur

Anwendung

Tabellen/SQL

relationale Datenbank

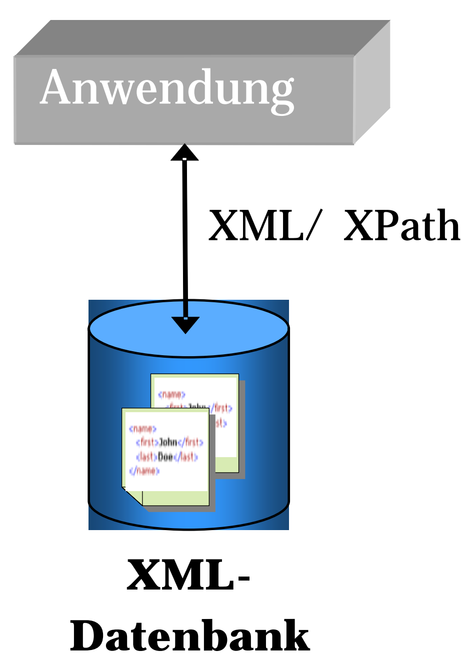
© K. Schild, 2006/M.Mochol, 2007

2. XML in XML-Datenbank speichern

- Internes Model basiert auf XML
- Übersicht XML-Datenbanken:
  www.rpbourret.com/xml/XMLDatabaseProds.htm

Vorteile

+ komplexe Anfragen auf XML-Struktur möglich
+ XPath wird unterstützt
+ Schnell bei einheitlichen Views



© K. Schild, 2006/M.Mochol, 2007

2. XML in XML-Datenbank speichern

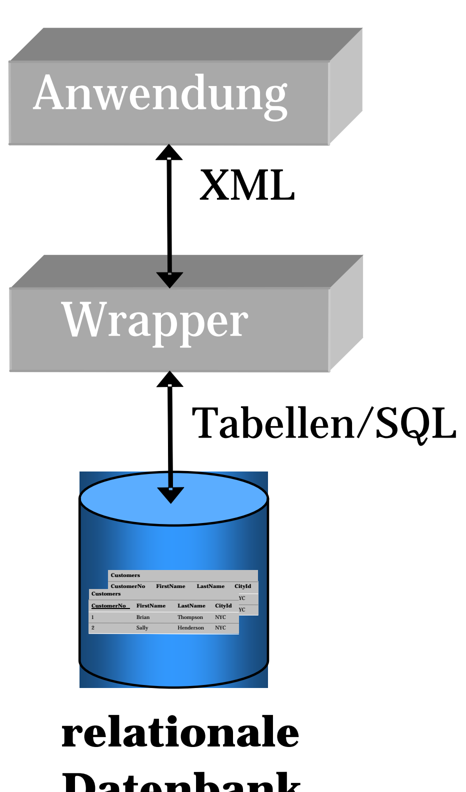
Nachteile

- neue Datenbank nötig
- XML-Datenbanken nicht interoperabel
- XML-Datenbanken liefern nur XML zurück
- schwierige Integration mit bestehenden relationalen Datenbanken (RDB)
- keine Systematik der Datenmodellierung
- Langsam bei Anfragen, die unterschiedliche Views verlangen

© K. Schild, 2006/M.Mochol, 2007

3. XML als Tabellen speichern

- Abbildung XML ➔Tabellen möglich
- Abbildung Tabellen ➔ XML problemlos
- Anfragen: SQL mit XML-Funktionen (z.B. SQL/XML)



Vor- und Nachteile

+ vorhandene RDB kann genutzt werden
+ von modernen RDBMS unterstützt
+ Systematik der Datenmodellierung für Tabellen

\- Abbildung XML ➔Tabellen ➔ XML liefert nicht unbedingt ursprüngliche XML-Struktur

Fazit: Wie XML speichern?

- Sollen Daten oder Dokumente gespeichert werden?
- Wie tief XML-Strukturen in die Datenbank integrieren?
  1. XML als einfachen String in Feld einer Tabelle speichern: **gar nicht integrieren**
  2. XML-Datenbanken: **voll integrieren**
  3. XML als Tabellen speichern: **soweit wie möglich integrieren**

- nur zu beantworten, wenn XML mit relationalem Datenmodell verglichen wird

Systematik der Datenmodellierung

relationales Modell

- schrittweise Normalformbildung: Ergebnis formal definiert

XML

- Normalformen aus relationalem Modell nicht auf XML übertragbar
- Grund: relationales Model erlaubt keine geschachtelten Tabellen
- bisher keine Systematik der Datenmodellierung
- informelles Verfahren:
  Asset-Oriented Modeling (AOM), www.aomodeling.org,
  (Daum & Merten, System Architecture with XML, 2003).

© K. Schild, 2006/M.Mochol, 2007

XML & Datenbanken → Lernziele

© K. Schild, 2006/M.Mochol, 2007

Lernziele

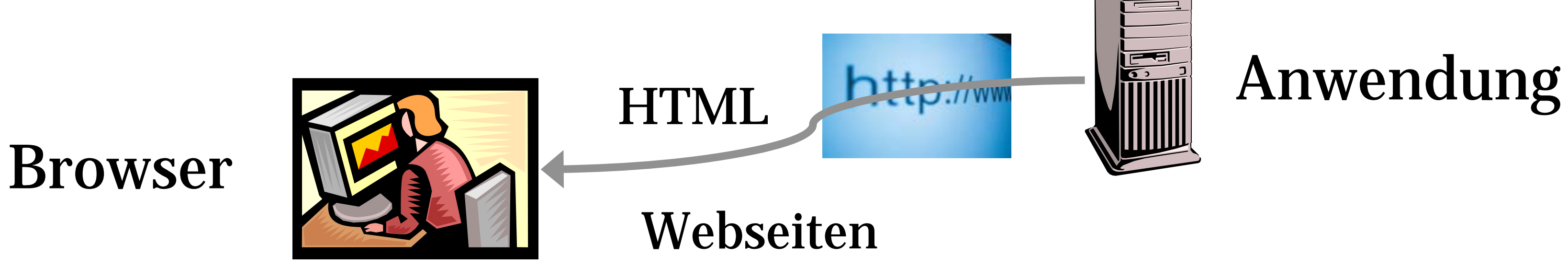
- drei Arten, XML persistent zu speichern
- Wie können Primär- und Fremdschlüssel in einem XML-Schema definiert werden?
- XML als Datenmodell oder relationales Datenmodell: Vor- und Nachteile

2. BLOCK
Web Services

2.1 Web Services

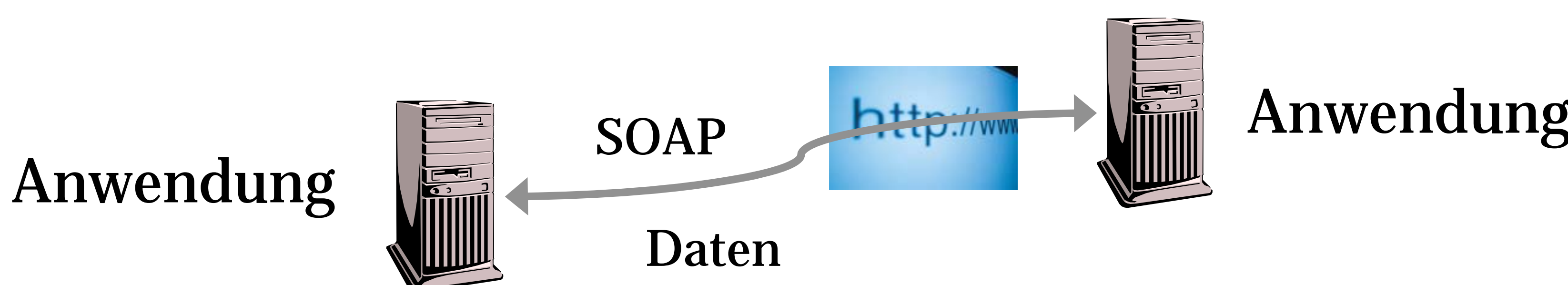
Was sind Web Services?

traditionelle Web-Anwendung



➔ Mensch-Maschine-Kommunikation

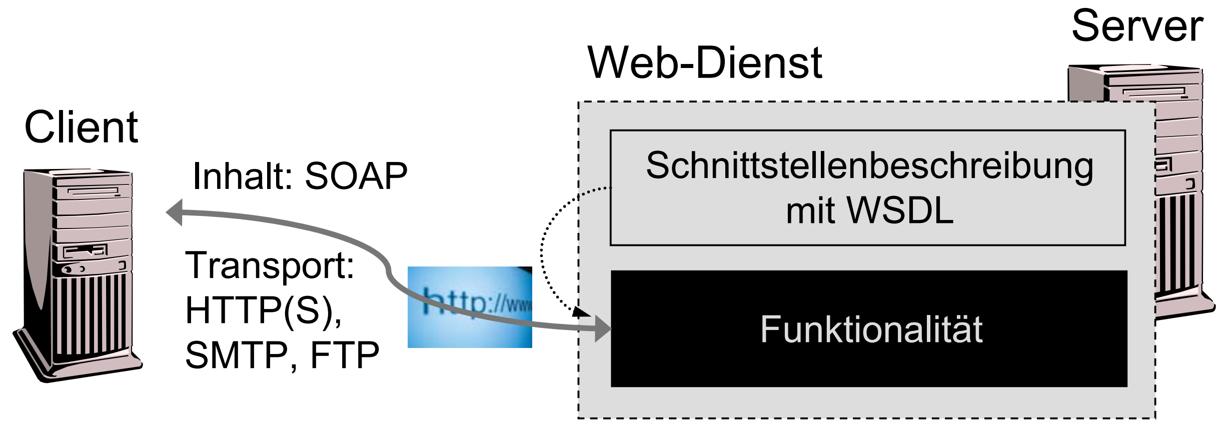
Web Service



➔ Maschine-Maschine-Kommunikation

© K. Schild, 2006/M.Mochol, 2007

Definition



Ein Web Service ist eine Softwareanwendung, die

1. mit einer URI eindeutig identifizierbar ist,
2. über eine WSDL-Schnittstellenbeschreibung verfügt,
3. nur über die in ihrer WSDL beschriebenen Methoden zugreifbar ist und
4. über gängige Internet-Protokolle unter Benutzung von XML-basierten Nachrichtenformaten wie z.B. SOAP zugreifbar ist.

© K. Schild, 2006/M.Mochol, 2007

Web Services

UDDI
Verzeichnis

Verweist auf
die Service-Beschreibung

WSDL

Verweist auf den Service

findet
Service

beschreibt
Service

Service Nutzer
(Service Consumer)

SOAP

Web Service

© K. Schild, 2006/M.Mochol, 2007

Dienstorientierte Architektur

- engl. service-oriented architecture, kurz SOA
- statt Anwendungen isoliert zu entwickeln, nur um sie später zu integrieren:
- neue Anwendungen von Anfang an auf existierenden Web Services aufbauen
- neue Anwendung wiederum als Web Service anbieten

*„... eine Systemarchitektur, die vielfältige, verschiedene und eventuell inkompatible Methoden oder Applikationen als wiederverwendbare und offen zugreifbare Dienste repräsentiert und dadurch eine plattform- und sprachunabhängige Nutzung und Wiederverwendung ermöglicht.“**

*Quelle: „ *Service-orientierte Architekturen mit Web Services: Konzepte – Standards – Praxis*“, W. Dostal, M. Jeckle, I. Melzer, B. Zengler; Spektrum Akademischer Verlag, 2005

Dienstorientierte Architektur

Verzeichnis
1. Dienst publizieren
3. Verweis auf Dienst
2. Dienst suchen
0. Dienst implementieren
4. Abfrage der Beschreibung
Anbieter
Nutzer
5. Dienst benutzen

- **publizieren** (**publish**): Beschreibung eines Dienstes in einem Verzeichnis (registry) veröffentlichen.
- **suchen** (**find**): Beschreibung eines Dienstes suchen, entweder dynamisch oder zur Entwicklungszeit
- **abrufen** (**bind**): Beschreibung des Dienstes verwenden, um Dienst abzurufen, entweder dynamisch oder zur Entwicklungszeit

© K. Schild, 2006/M.Mochol, 2007

Standards

öffentliches Verzeichnis

WSDL-Beschreibung

UDDI

WSDL-Beschreibung

Anbieter

SOAP-Nachrichten

Nachfrager

- SOAP und WSDL allgemein akzeptiert
- UDDI: Standard zur Beschreibung von Web-Service-Verzeichnissen
- UDDI umstritten und wenig genutzt

© K. Schild, 2006/M.Mochol, 2007

2.1 Web Services → Lernziele

© K. Schild, 2006/M.Mochol, 2007

Lernziele

- Was sind Web Services?
- Was ist neu an Web Services und was nicht?
- Was ist SOA?

2.2 SOAP

Web Services

UDDI
Verzeichnis

Verweist auf
die Service-Beschreibung

WSDL

Sucht nach
Service

Verweist auf den Service

Beschreibt
Service

Service Nutzer
(Service Consumer)

SOAP

Web Service

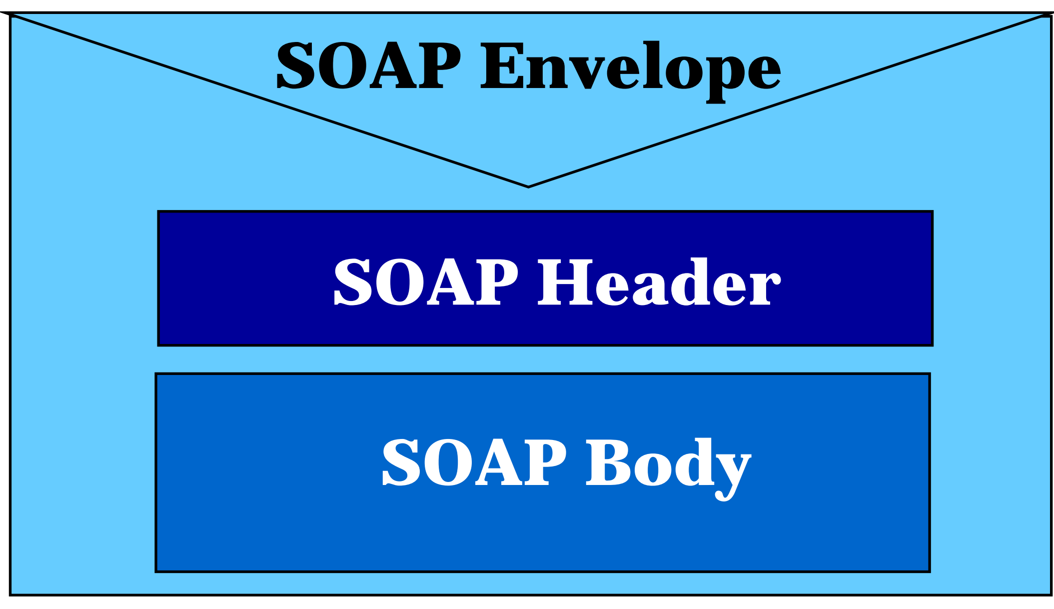
© K. Schild, 2006/M.Mochol, 2007

Was ist SOAP?

- Kommunikationskomponente von Web Services.
- Protokoll für Nachrichtenaustausch zwischen Web Service-Konsument und Web Service-Anbieter
- XML-basiert
- Plattformunabhängig
- Programmierspracheunabhängig
- basierte auf Entwicklungen von Microsoft und IBM

© K. Schild, 2006/M.Mochol, 2007

Aufbau einer SOAP-Nachricht



<**env**:Envelope xmlns:env=" **http://www.w3.org/2003/05/soap-envelope**/">

<!- SOAP Header -->

<!- SOAP Body -->

</**env**:Envelope>

SOAP Version 1.2

© K. Schild, 2006/M.Mochol, 2007

Version = Envelope-Namensraum

SOAP 1.1

- XMLSOAP-Namensraum

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env=" http://schemas.xmlsoap.org/soap/envelope">
  ...
</env:Envelope>
```

- unterschiedliche Namensräume

⇨ nicht kompatibel

SOAP 1.2

- W3C-Namensraum

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  ...
</env:Envelope>
```

Prinzipieller Aufbau (SOAP V.1.1)

```
<?xml version='1.0' encoding='UTF-8'?>
<env:Envelope
      xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header>
    Zusatzinformationen
  </env:Header>
  <env:Body>
    Nachrichtinhalt
  </env:Body>
</env:Envelope>
```

- Wurzel-Element: Envelope aus SOAP-Namensraum
- kein W3C-Namensraum
- Header: optional
- Body: obligatorisch

SOAP Envelope Element

```
<?xml version="1.0"?>
<env:Envelope
  xlmns:env="http://www.w3.org/2003/05/soap-envelope">
   ...
</env:Envelope>
```

- Name des Elements: Envelope
- Envelope: Wurzel-Element einer SOAP Nachricht
- beinhaltet SOAP Namespace
- identifiziert SOAP Nachricht

© K. Schild, 2006/M.Mochol, 2007

SOAP Body Element → Nachrichteninhalt

```
<env:Envelope ...>
   <env:Body xmlns:ns="URI">
     <ns:Nachrichtinhalt-Teil-1>...</ns:Nachrichteninhalt-Teil-1>
          ...
     <ns:Nachrichteninhalt-Teil-n>...</ns:Nachrichteninhalt-Teil-n>
   </env:Body>
</env:Envelope>
```

- Body: beliebige XML-Inhalte erlaubt
- Struktur von Anwendung festgelegt, z.B. durch:
  - speziellen Namensraum und/oder
  - WSDL-Beschreibung

© K. Schild, 2006/M.Mochol, 2007

SOAP Header Element → Briefkopf

```
<env:Envelope ...>
  <env:Header xmlns:ns="URI" >
    <ns:Zusatzinformation-1>...</ns:Zusatzinformation-1>
        ...
    <ns:Zusatzinformation-n>...</ns:Zusatzinformation-n>
  </env:Header>
  <env:Body>...</env:Body>
</env:Envelope>
```

Header Block

- Header: beliebige XML-Inhalte erlaubt
- Struktur von Anwendung festgelegt
- Header Block
  - Kind-Element von Header
  - Zusatzinformation zur eigentlichen Nachricht

mustUnderstand Attribut

```
<env:Header>
    <alertcontrol xmlns="http://example.org/alertcontrol"
                  env:mustUnderstand="true">
      ...
    </alertcontrol>
</env:Header>
```

- mustUnderstand="true": Empfänger muss Header Block verstehen oder mit Fehlermeldung antworten
- mustUnderstand="false": Empfänger kann Header Block (ohne Fehlermeldung) ignorieren
- kann für jeden Header Block unterschiedlich sein
- Beachte: Standard-Wert ist "false"

© K. Schild, 2006/M.Mochol, 2007

HTTP-GET vs. HTTP-POST

HTTP GET

- URL ➔ Antwort
- Parameter können in URL kodiert werden, z.B.:
- http://google.com/doGoogleSearch?q=Beginning+XML
- = Aufruf doGoogleSearch(q="Beginning XML")

HTTP POST

- URL + Datenanhang ➔ Antwort

© K. Schild, 2006/M.Mochol, 2007

SOAP über HTTP GET

```
GET /search/beta2/doGoogleSearch?q=Beginning+XML HTTP/1.1
Host: api.google.com
Accept: application/soap+xml
```

- ruft doGoogleSearch(q="Beginning XML") auf
- gesamte SOAP-Nachricht als URL kodiert
- Antwort wie bei HTTP POST
- Amazon bietet HTTP-GET-Schnittstelle an, Google jedoch nicht
- sehr beliebt weil leichtgewichtig

© K. Schild, 2006/M.Mochol, 2007

REpresentational State Transfer

- kein Protokoll sondern ein Architekturstil
- Verwaltet beliebige Menge von Ressourcen
- RESTful → eine Anwendung konform zum REST-Architekturstil

- Amazon – der populärster Anbieter einer REST-Anwendung
- Google bietet solche REST-Schnittstelle NICHT an

© K. Schild, 2006/M.Mochol, 2007

REST-Grundsatz

- REST-Architektur des WWW (Fielding 2000):

  jede Web-Ressource soll eindeutig über eine URI identifiziert werden

- Beispiel: online gebuchte Reise = Web-Ressource
- gebuchte Reise sollte daher auch über eine URI eindeutig identifiziert werden

© K. Schild, 2006/M.Mochol, 2007

REST oder nicht?

HTTP GET ⇨ entspricht REST-Grundsatz

- würde z.B.
  travel.com/Reservations/itinerary?reservationCode=FT35ZBQ
  anfragen

⇨ URL identifiziert eindeutig gebuchte Reise

HTTP POST ⇨ widerspricht REST-Grundsatz

- würde z.B.
  travel.com/Reservations/
  anfragen mit SOAP-RPC als Datenhang:
  itinerary(reservationCode="FT35ZBQ")

⇨ URL identifiziert nicht gebuchte Reise

© K. Schild, 2006/M.Mochol, 2007

SOAP → Lernziel

SOAP

Lernziele

- Aufbau einer SOAP-Nachricht
- SOAP Version 1.1 vs. SOAP Version 1.2
- Was ist ein Kodierungsstil in SOAP?
- SOAP mit HTTP GET, HTTP POST und REST

© K. Schild, 2006/M.Mochol, 2007

2.3 WSDL

© K. Schild, 2006/M.Mochol, 2007

WSDL = Web Services Description Language

- baut auf XML-Schema auf
- stellt ein XML-Vokabular zur Beschreibung von Web Services (Schnittstellen, Operationen und Dienste) dar

  → Standard für die Beschreibung dessen, was zwischen Konsument und Anbieter geschickt wird

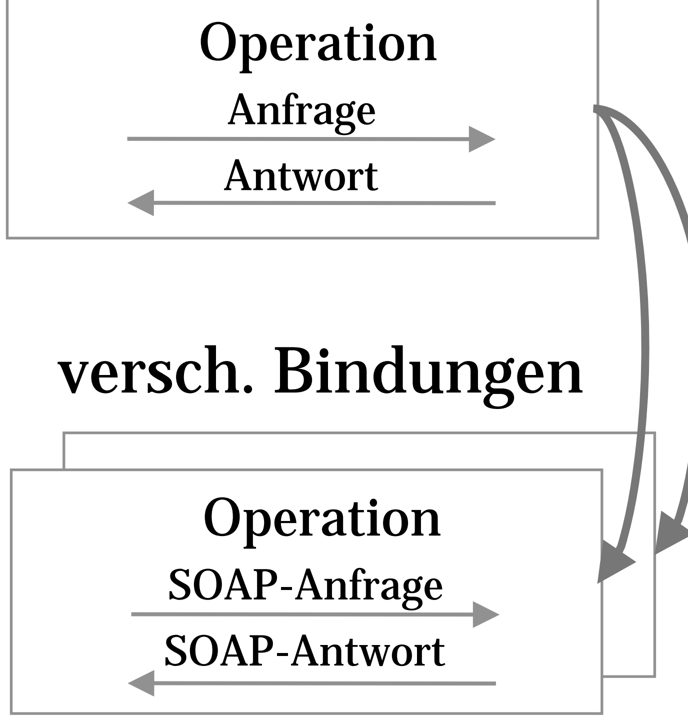
  → Syntax einer Schnittstelle kann bis ins kleinste Detail festgelegt werden

  → Beschreibung von Grundlegende Interaktionsmuster (wie Anfrage-Antwort)

- WSDL Version 2.0 Part 1: Core Language - W3C Recommendation seit 26. Juni 2007

© K. Schild, 2006/M.Mochol, 2007

Grundidee



abstrakte Schnittstelle

- Beschreibung der Schnittstelle unabhängig von
  - Nachrichtenformaten wie SOAP
  - Übertragungsprotokollen wie HTTP

Bindung

- Realisierung einer abstrakten Schnittstelle mit bestimmtem Nachrichtenformat und Übertragungsprotokoll

© K. Schild, 2006/M.Mochol, 2007

Grundstruktur

© K. Schild, 2006/M.Mochol, 2007

WSDL 1.1. – Elemente (1)

Element	Beschreibung
Abstrakte Beschreibung	
`<types>` … `</types>`	- Maschinen- und sprachunabhängige Typdefinitionen → definiert die verwendeten Datentypen
`<message>` … `</message>`	- Nachrichten, die übertragen werden sollen - Funktionsparameter (Trennung zwischen Ein- und Ausgabeparameter) oder Dokumentbeschreibungen
`<portType>` … `</portType>`	- Nachrichtendefinitionen im Messages-Abschnitt - definiert Operationen, die beim Web Service ausgeführt werden

© K. Schild, 2006/M.Mochol, 2007

WSDL 1.1. – Elemente (2)

Element	Beschreibung
Konkrete Beschreibung	
`<binding>…</binding>`	- Kommunikationsprotokoll, der beim Web Service benutzt wird - Gibt die Bindung(en) der einzelnen Operationen im portType-Abschnitt an
`<service>…</service>`	- gibt die Anschlussadresse(n) der einzelnen Bindungen an (Sammlung von einem oder mehrere Ports)

© K. Schild, 2006/M.Mochol, 2007

Datentyp / Nachricht /Porttyp

```
<types>
  <xsd:schema  xmlns:xsd="..."  xmlns:tns="..." targetNamespace="...">
    <xsd:complexType name="GoogleSearchResult">
      ...
    </xsd:complexType>
  </schema>
 </types>
<message name="doGoogleSearch">...</message>
<message name="doGoogleSearchResponse">
  <part name="return" type="tns:GoogleSearchResult"/>
</message>
 ...
<portType>
  <operation name="doGoogleSearch">
    <input message="tns:doGoogleSearch"/>
    <output message="tns:doGoogleSearchResponse"/>
  </operation>
 ...
</portType>
```

Definition des Datentyps

Definition einer abstrakten Nachricht

Definition einer abstrakten Schnittstelle

portType

message

types

<service>

```
<definitions name="GoogleSearch"
        targetNamespace="urn:GoogleSearch"
         ...
         xmlns="http://schemas.xmlsoap.org/wsdl/">
<!-- abstrakte Definition -->
  <types>...</types>
  <message name="doGoogleSearch">...</message>
  <message name="doGoogleSearchResponse">...</message>
  <portType name="GoogleSearchPort">...</portType>
<!-- konkrete Definition -->
  <binding name="GoogleSearchBinding"  type="tns:GoogleSearchPort">
    ...
   </binding>
  <service name="GoogleSearchService">...</service>
</definitions>
```

© K. Schild, 2006/M.Mochol, 2007

WSDL→ Lernziel

© K. Schild, 2006/M.Mochol, 2007

WSDL

Lernziele

- Was ist WSDL?
- Abstrakte / Konkrete Definition in WSDL
- Grundstruktur von WSDL
- SOAP-Bindungen (HTTP GET, SOAP)

© K. Schild, 2006/M.Mochol, 2007

Klausur → Organisatorisches

Klausur

Organisatorisches

- ➔ 18.Juli um 12:00 **pünktlich!!!**
- ➔ 2 Gruppen:
  - ➔ 1.Gruppe – Hörsaal Informatik (Nachname A bis L)
  - ➔ 2. Gruppe – ZIB Hörsaal (Nachname M bis Z)

Was sind die Voraussetzungen?

- Sie haben die Folien der Vorlesung verstanden und können dieses Wissen anwenden.
- Sie haben die Übungen gemacht und die Musterlösungen verstanden.
- Sie beherrschen den Vorlesungsinhalt passiv und soweit aktiv, dass Sie kleine Änderungen vornehmen können

Klausur

Struktur: Sie haben 90 Minuten

- 25 Multiple Choice Fragen (25 Punkte insgesamt)
- 7 ausführliche Fragen (65 Punkte insgesamt)

Beispiel: Multiple Choice

- Ähnlich zu den Fragen, die in der Übung vorgestellt wurden
- Beachte: **nicht immer nur eine korrekte Antwort!**

Beispiel: ausführliche Fragen

- Typischerweise mit gegebenem XML Code
- Aufgabe: Korrektur, Änderung, Erweiterung des Codes.
- Oder: XML Instanz von Schema, XSLT Ausgabe usw.

© K. Schild, 2006/M.Mochol, 2007

Notenschema

Punkte	Note
≥ 81	1,0
≥ 77	1.3
≥ 73	1,7
≥ 69	2,0
≥ 65	2,3
≥ 61	2,7
≥ 57	3,0
≥ 53	3,3
≥ 49	3,7
≥ 45	4,0

© K. Schild, 2006/M.Mochol, 2007

Klausur – Sprachen

- Fragen sind nur auf Deutsch
- Antworten auf Deutsch oder Englisch möglich
- keine eigenen (Wörter-)Bücher, Papiere usw. erlaubt
- ein paar Englisch-Deutsch Wörterbücher werden wir mitbringen.

© K. Schild, 2006/M.Mochol, 2007

Klausur – allg. Infos

- Während der Prüfung dürfen Sie Hand hochheben wenn Sie Hilfe brauchen.
- Sitzordnung: **jede zweite Reihe, jeder dritte Platz!**
- Studentenausweis und auch Ausweis mit Foto mitbringen!
- Handy aus !

© K. Schild, 2006/M.Mochol, 2007

- Zusätzliche Sprechstunde vor der Klausur:

→ Heute, 14:00-16:00, Fabeckstr. 15

Viel Glück!