



Rückblick – Klausurvorbereitung

Dr. Malgorzata Mochol
Freie Universität Berlin
Institut für Informatik
Netzbasierte Informationssysteme
mochol@inf.fu-berlin.de

Vorlesungen – 7 Termine	Übungen – 5 Termine
XML-Grundlagen einsch. Namenräume	XML-Syntax, Namensräume
DTD & XML-Schema	DTD
XML-Schema im Detail	XML Schema
SAX & DOM Parser	
XPath	XPath & Co.
XSLT	XSLT
XLST 2.0	

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

Vorlesungstermin	Vorlesung
08.07. (heute)	Rückblick = Klausurvorbereitung
15.07.	Klausur



1.Block - XML



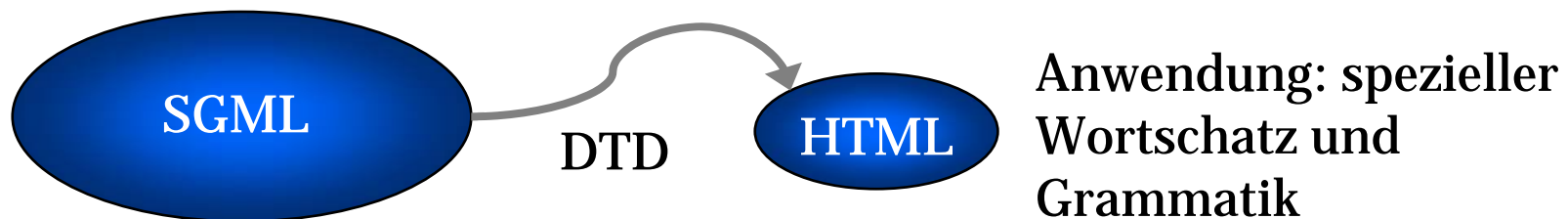
1.1. XML

- **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

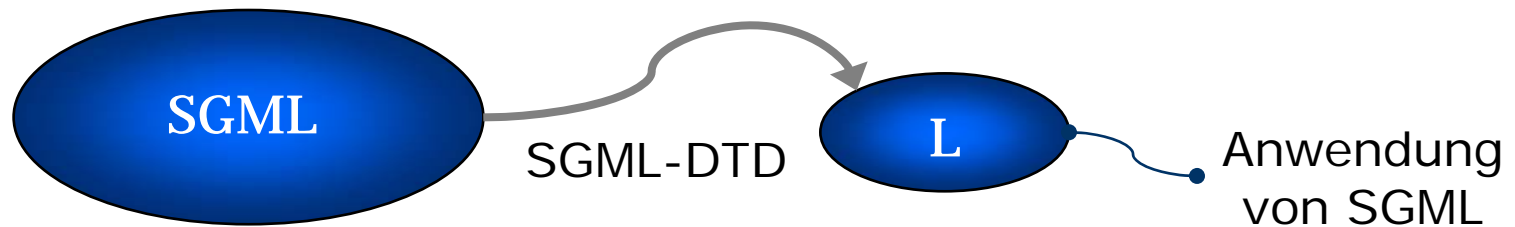
- eine **Methode**, um **strukturierte Daten** in einer **Textdatei** darzustellen
- Text, aber nicht zum Lesen
- eine **Familie von Technologien**
- **lizenzfrei** und **plattformunabhängig**
- ein **offener Standard**, der sich weit verbreitet hat
- ein Protokoll zur Aufnahme und Verwaltung von Informationen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

SGML

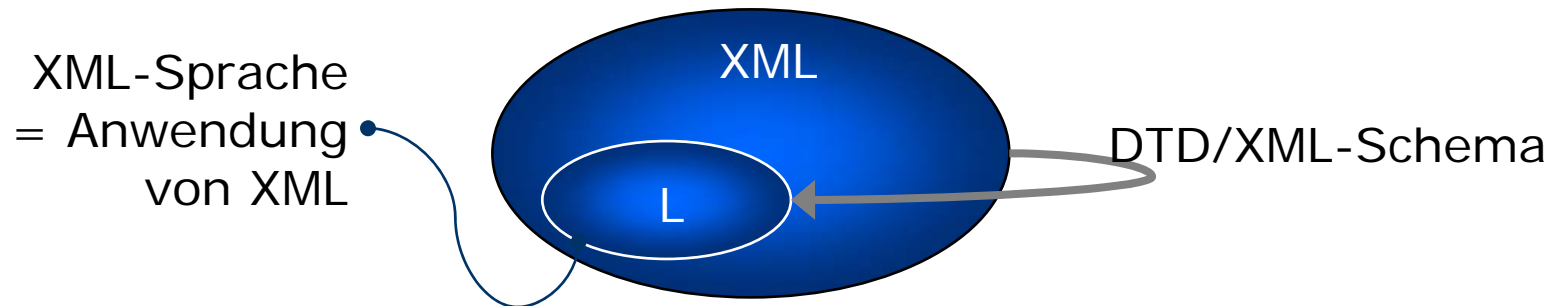
- Standard Generalized Markup Language
- *keine* vorgegebenen Tags, auch keine für das Layout von Dokumenten
- Vorgänger von XML
- Anwendungen von SGML → mit Document Type Definitions (DTDs) können spezielle Auszeichnungssprachen mit konkreten Tags definiert werden:
 - bekannteste Anwendung von SGML: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



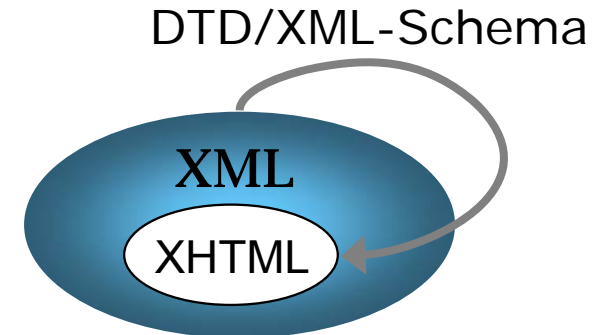
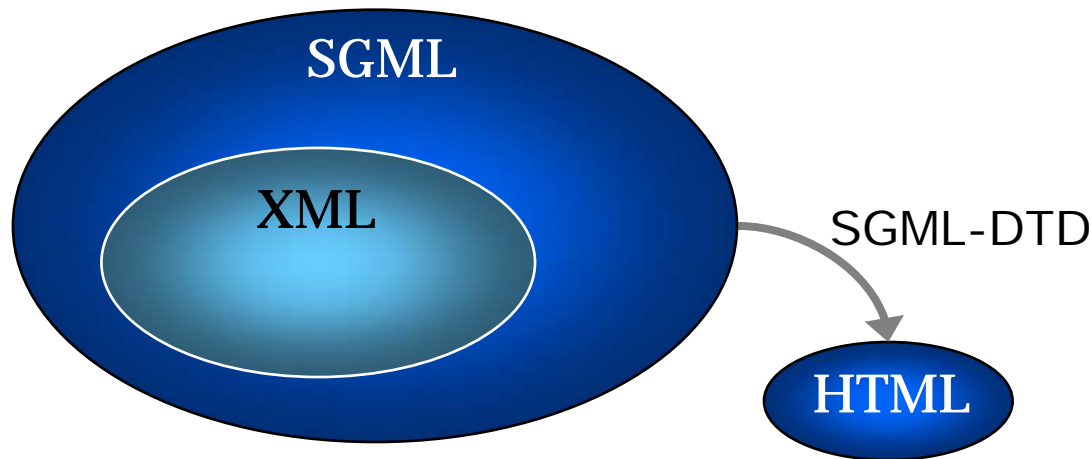
- Anwendung selbst kann keine Anwendung definieren



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



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



HTML

- Anwendung von SGML

XML

- Teilsprache von SGML

XHTML

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

- **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 Elementen eine Bedeutung

1. unstrukturierter Inhalt:

- einfacher Text ohne Kind-Elemente

2. strukturierter Inhalt:

- Sequenz von $> \emptyset$ Kind-Elementen

3. gemischter Inhalt:

- enthält Text mit mind. einem Kind-Element

4. leerer Inhalt

```
<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

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

- enthält Informationen für Parser: z.B. verwendete XML-Version und Kodierung
- wenn vorhanden, dann immer **am Anfang der Datei**
- in XML 1.0 optional, in XML 1.1 obligatorisch

- **Attribut: version**

```
<?xml version="1.0" encoding="UTF-8"?>
```

- 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" – default) oder nicht ("yes").
- Optional

Beachte: immer in dieser Reihenfolge!

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.
8. Alle Attributwerte stehen in Anführungszeichen

{
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.

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
- Beachte: Wahl des Präfixes egal!

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**

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



1.1. XML → Lernziele

Lernziele

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



1.2. 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)>
```

ähnelt einer
regulären
Grammatik

<!ELEMENT *Name Content-Modell*>

Element-Deklaration

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)

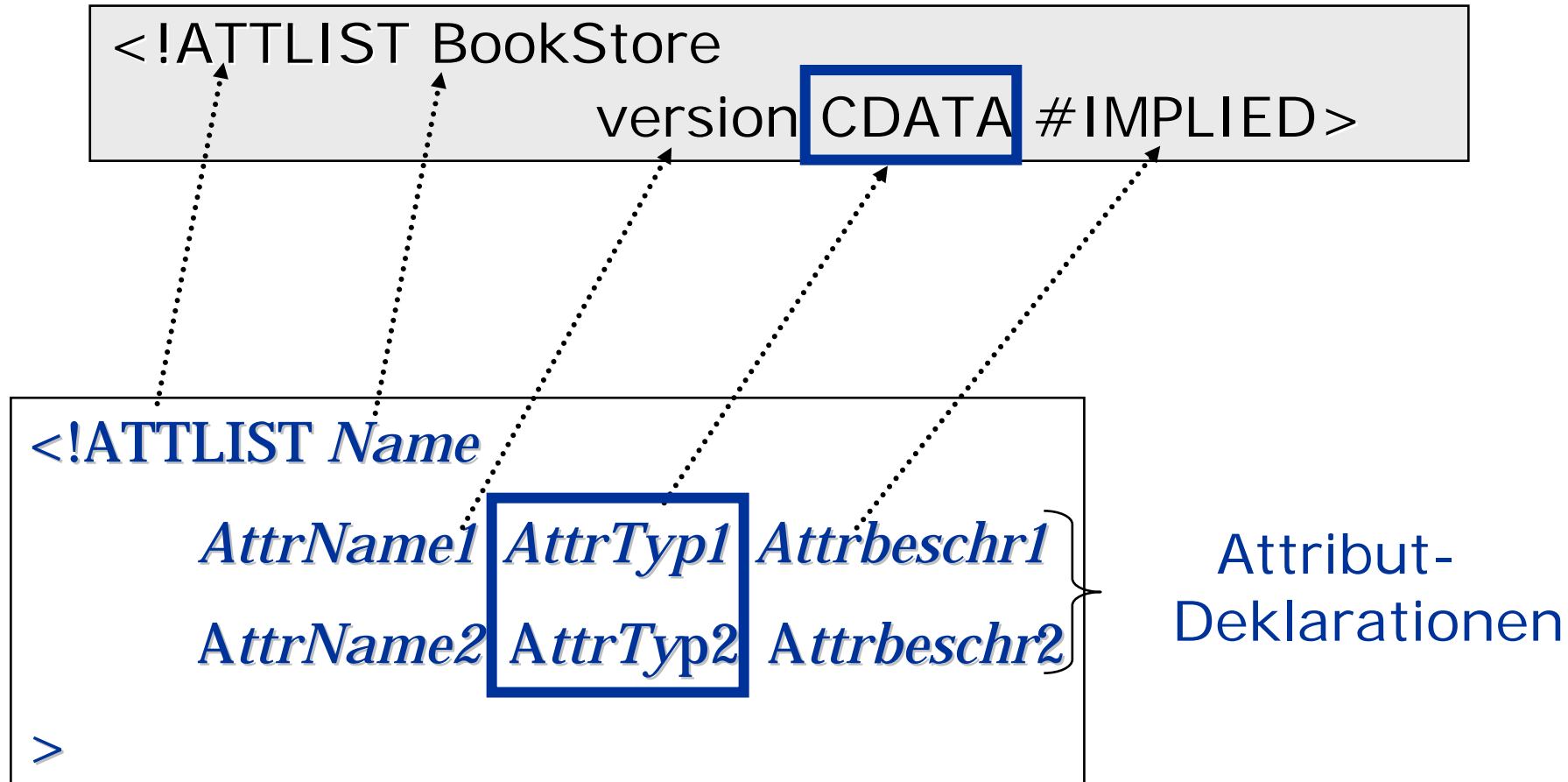
<!ELEMENT title ANY>

Datentypen wie INTEGER oder FLOAT stehen nicht zur Verfügung.

- + 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



```
<!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**.

```
<!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



1.3. XML Schema

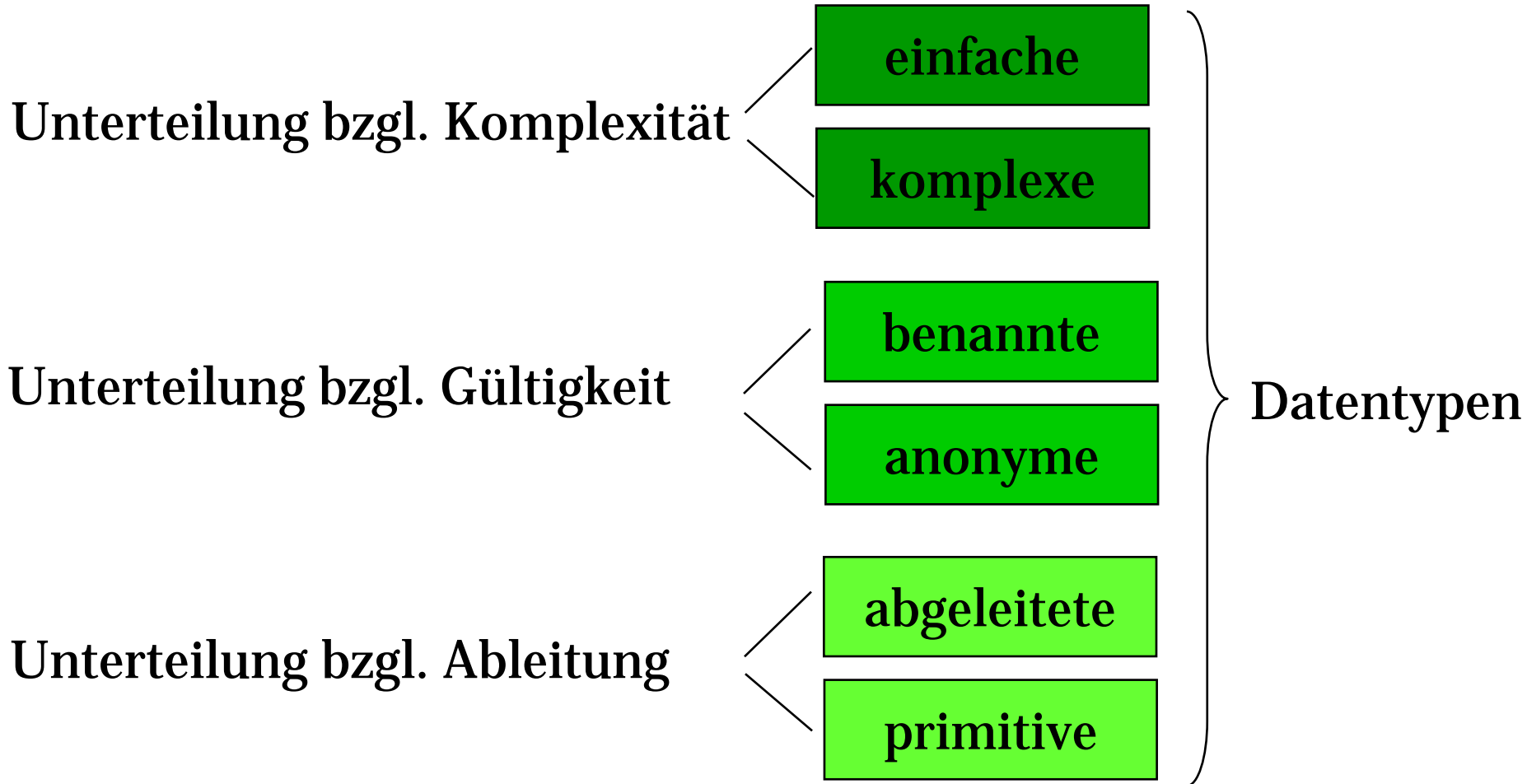
- eine XML basierte Alternative zu DTD
- beschreibt die Struktur eines XML Dokuments
- eine W3C Recommendation
- statt XML Schema wird oft die Abkürzung XSD (XML Schema Definition) benutzt
- definiert
 - Elemente/Attribute , die im Dokument vorkommen dürfen
 - die Reihenfolge & Anzahl der (Kinder-)Elemente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

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

```
<?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.



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

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.

```
<xsd:element name="BookStore" >
```

```
  <xsd:complexType>
```

```
    Liste von Büchern
```

```
  </xsd:complexType>
```

```
</xsd:element >
```

- **anonymer** Datentyp
- lokale Definition

```
<xsd:complexType name="BookStoreType" >
```

```
  Liste von Büchern
```

```
</xsd:complexType >
```

- **benannter** Datentyp
- globale Definition
- wiederverwendbar

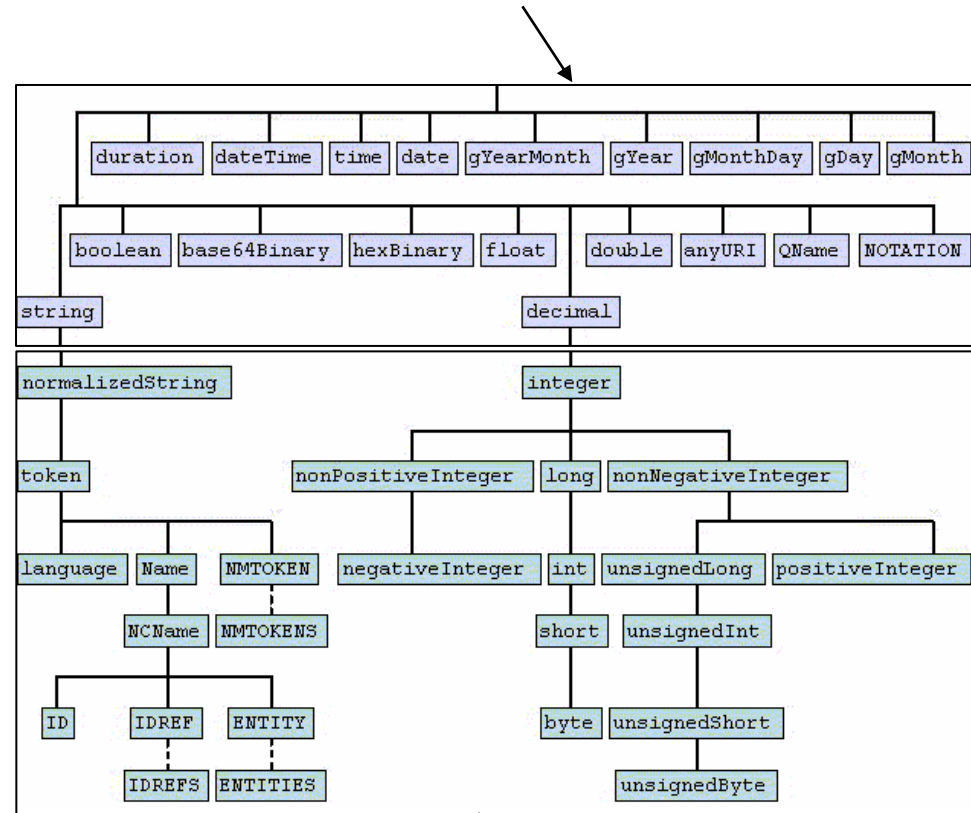
primitive Datentypen (primitive types)

- nicht von anderen Datentypen abgeleitet

abgeleitete Datentypen (derived types)

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

Primitive einfache Datentypen



Abgeleitete einfache Datentypen

Einfache Datentypen **ABLEITEN**

```
<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

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

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

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:choice>...</xsd:choice>`

- Inhalt besteht aus **genau einem** der aufgezählten Alternativen.

Beachte: Alle Operatoren können `minOccurs` und `maxOccurs` selbst spezifizieren

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`

```
<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

- **anonymer Datentyp ist entweder komplex:**

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

- **oder einfach:**

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

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

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

- oder Deklaration mit **anonymen Datentyp**:

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

```
<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

DTDs	XML Schema
vereinfachte SGML-DTD, Teil von XML 1.0/1.1	eigener W3C-Standard
eigene Sprache/Syntax	XML-Schema = XML-Sprache
kompakter und lesbarer	ausdrucksstark & mächtig
nur wenige „Datentypen“	unterstützt Datentypen
reihenfolgeunabhängige Strukturen schwierig zu definieren	reihenfolgeunabhängige Strukturen einfach zu definieren
Datentypen nicht erweiterbar, d.h. keine eigenen Datentypen	Datentypen erweiterbar, d.h. Definition von eigenen Datentypen möglich
keine Namenräume	Unterstützt Namenräume
zur Beschreibung von Text- Dokumenten ausreichend	zur Beschreibung von Daten besser geeignet



1.2. & 1.3 DTD & XML Schema → Lernziele

Lernziele

- DTDs und XML-Schemata lesen, verstehen und anpassen können (!)
- Vorteile von XML-Schema gegenüber DTDs
- Welche Arten von Datentypen gibt es in DTDs & welche in XML Schema?
- Wie kann man Datentypen in XML Schema bilden/ableiten?



1.4. XML Parser

Kategorien von Parser

Validierender vs. nicht-validierender Parser

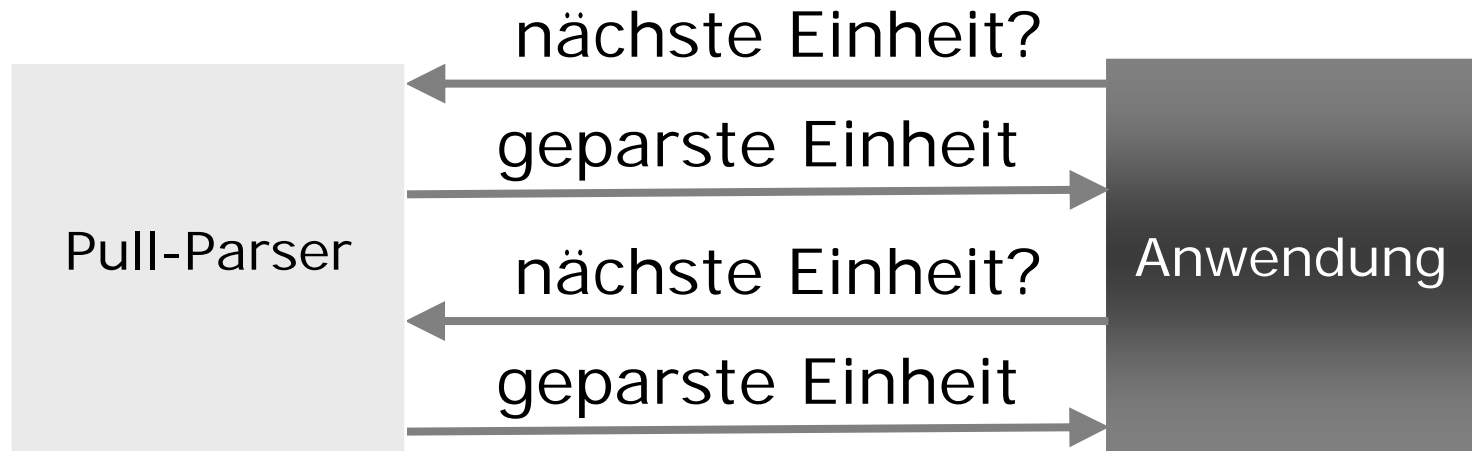
- Wird die Validität des Dokumentes untersucht?

Pull- vs. Push-Parser

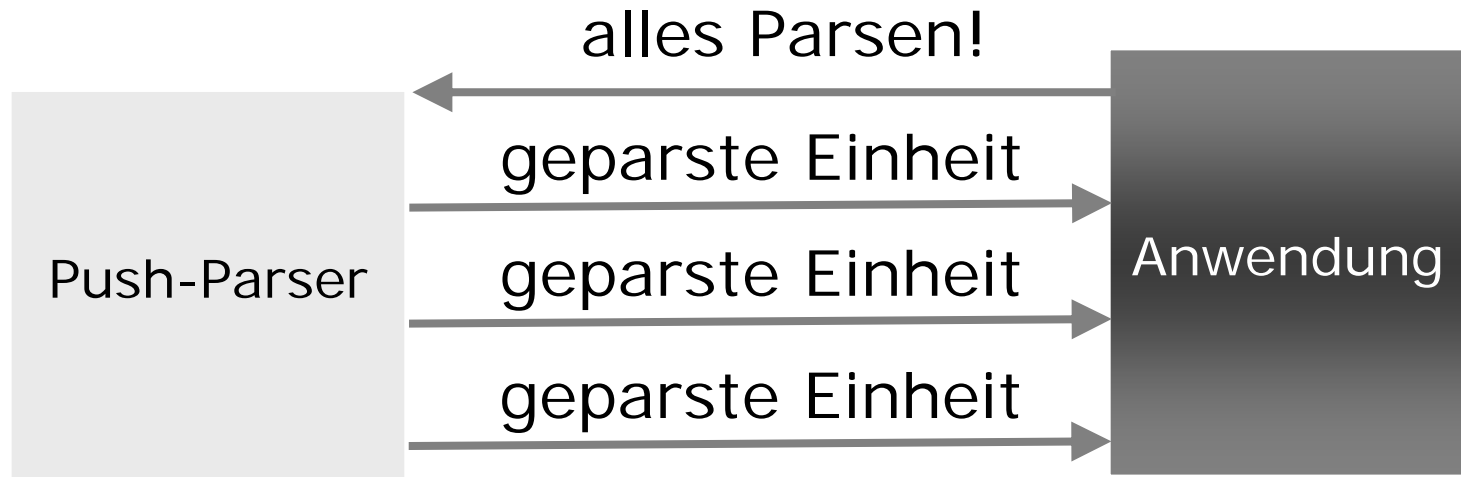
- Wer hat Kontrolle über das Parsen: die Anwendung oder der Parser?

Einschritt- vs. Mehrschritt-Parser

- Wird das XML-Dokument in einem Schritt geparkt oder Schritt für Schritt?
- Beachte: Kategorien unabhängig voneinander, können kombiniert werden



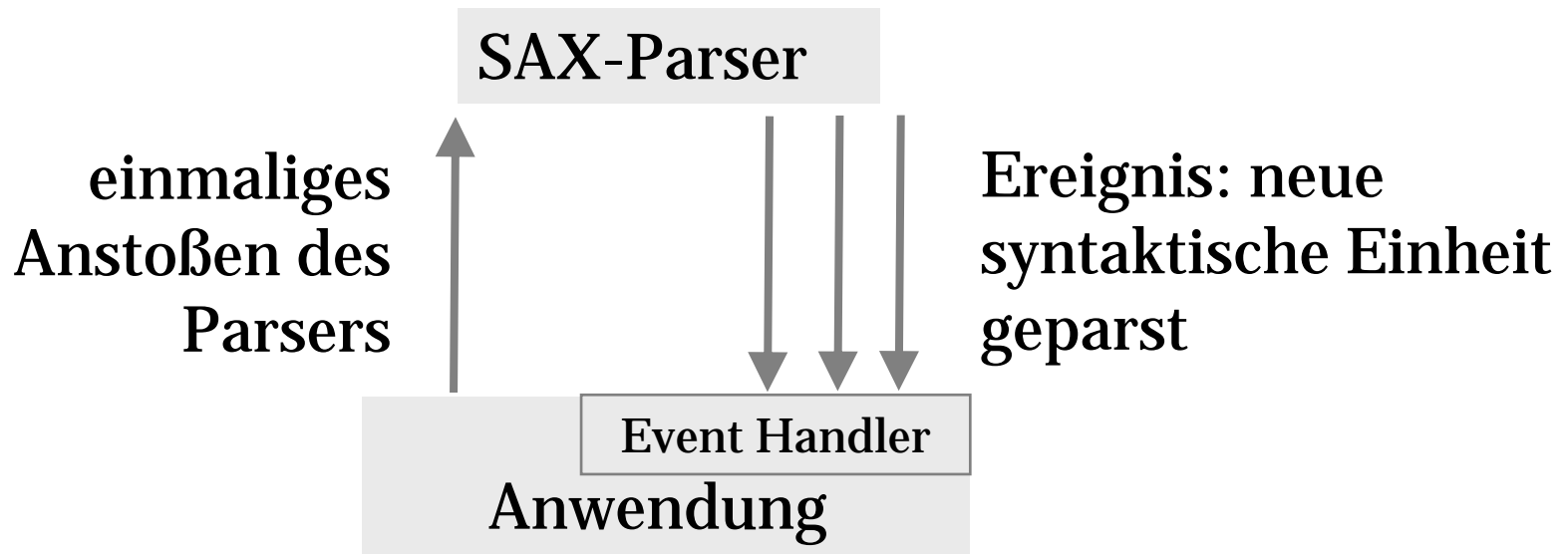
- Anwendung hat Kontrolle über das Parsen.
- Analyse der nächsten syntaktischen Einheit muss aktiv angefordert werden.
- Beachte: „Pull“ aus Perspektive der Anwendung.



- 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.

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



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 geparkt wurde, wird Anwendung benachrichtigt.
- Beachte: Es wird kein Parse-Baum aufgebaut!

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

Vor- und Nachteile von SAX

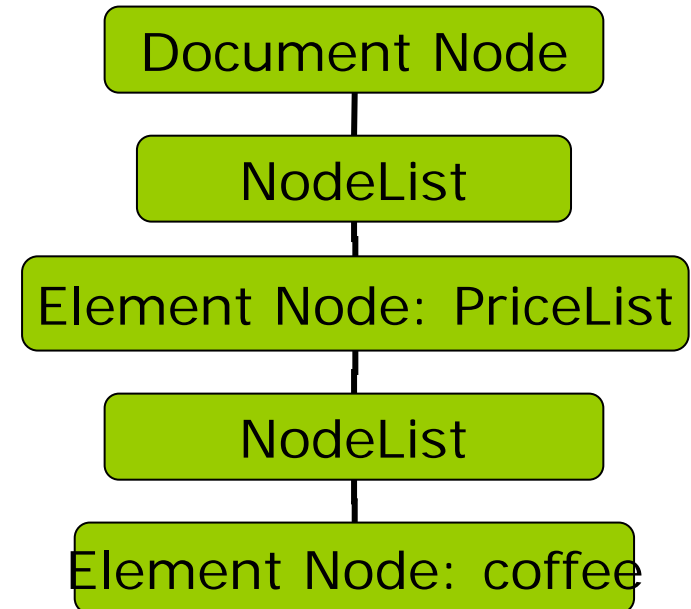
- + sehr effizient und schnell, auch bei großen XML-Dokumenten
- + relative einfach
- Kontext (Parse-Baum) muss von Anwendung selbst verwaltet werden.
- abstrahiert nicht von XML-Syntax
- nur Parsen möglich, keine Modifikation oder Erstellung von XML-Dokumenten

Document Object Model (DOM)



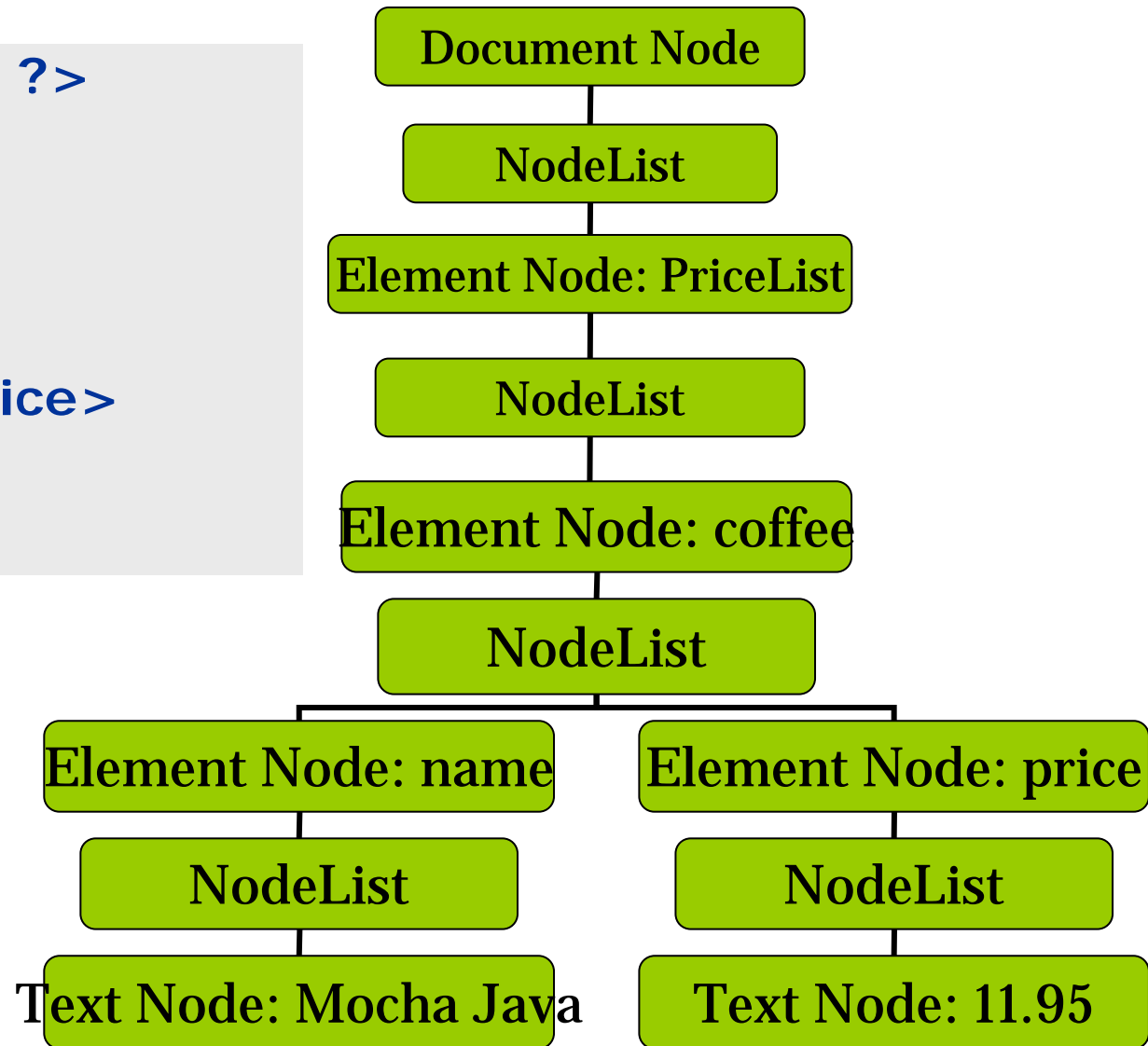
- streng genommen kein Parser, sondern abstrakte Schnittstelle zum Zugreifen, Modifizieren und Erstellen von Parse-Bäumen
- W3C-Standard
- unabhängig von Programmiersprachen
- nicht nur für XML-, sondern auch für HTML-Dokumente
- im Ergebnis aber **Einschritt-Pull-Parser**

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



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

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



- Beachte: PCDATA wird als eigener Knoten dargestellt.

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

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



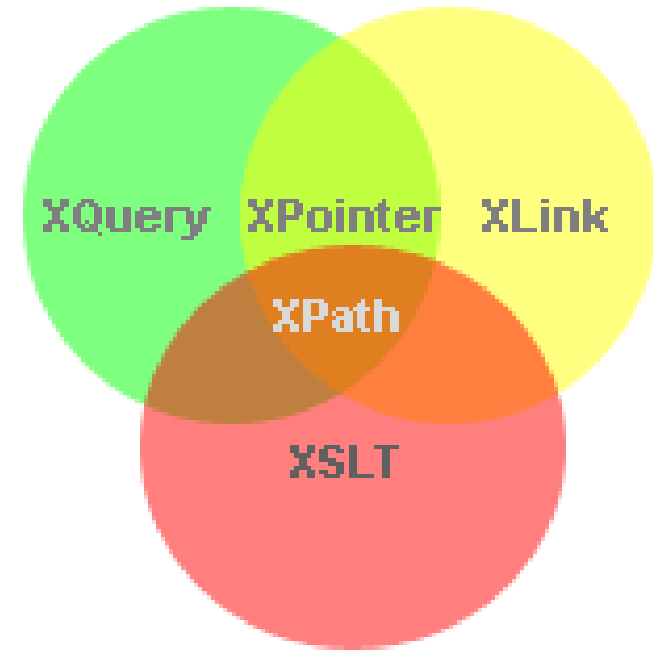
1.4. XML Parser → Lernziele

Lernziele

- Welche Kategorien von Parser gibt es?
- Wie arbeitet ein SAX-Parser?
 - Was sind Call-back Methoden?
 - Was ist ein Event-Handler?
- Wie funktioniert ein DOM-Parser?
 - DOM-Baum
- Vor- und Nachteile von SAX- und DOM-Parser



1.5. XPath & Co



Absolute und relative Pfade

- absolute Pfade

- beginnen mit "/", „

z.B. /order/item

lesen: (➔) Folge dem Pfad von dem Wurzelknoten 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` kann an beliebiger Stelle des XML-Dokumentes

- . aktueller Knoten
- .. Eltern-Knoten
- * beliebiges Kind-Element
- @* beliebiges Attribut
- // überspringt ≥ 0 Hierarchie-Ebenen nach unten
- [] Prädikatbeschreibung (Ziel \rightarrow genauere Element-Spezifikation)
- | Auswahl (Vereinigung)
- Beispiel: *|@*
„Kind-Element oder Attribut des aktuellen Knotens“

- `order/item[@item-id = 'E16-2']`
 - item-Elemente, die Kind von order sind und Attribut item-id mit Wert 'E16-2' haben

```
<order id="4711">  
  <item item-id="E16-2">  
    <name>buch</name>  
  </item>  
</order>
```

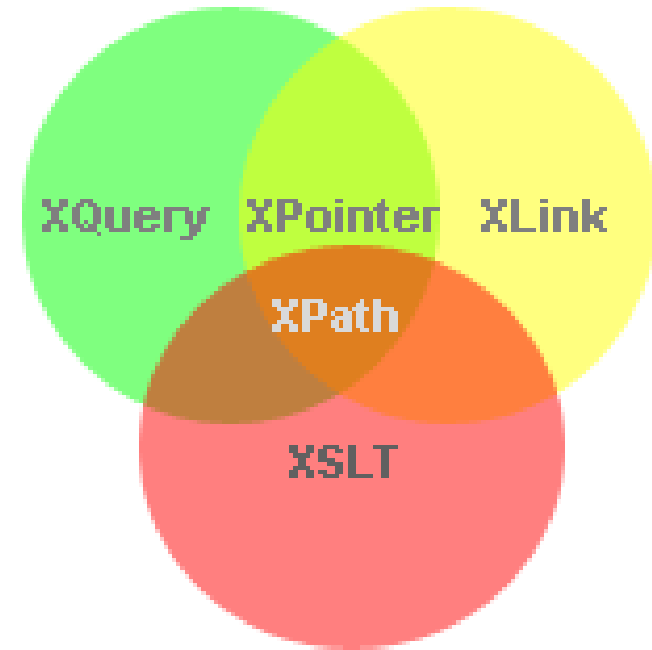
- Randbedingungen können an beliebiger Stelle in einem Pfad vorkommen:

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

```
<orders>  
  <order ored-id="4711">  
    <item item-id="E16-2">  
      <name>buch</name>  
    </item>  
  </order>  
  <order id="4711">  
    <item item-id="E16-3"/>  
  </order>  
</orders>
```



1.6. XSLT



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

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

Template

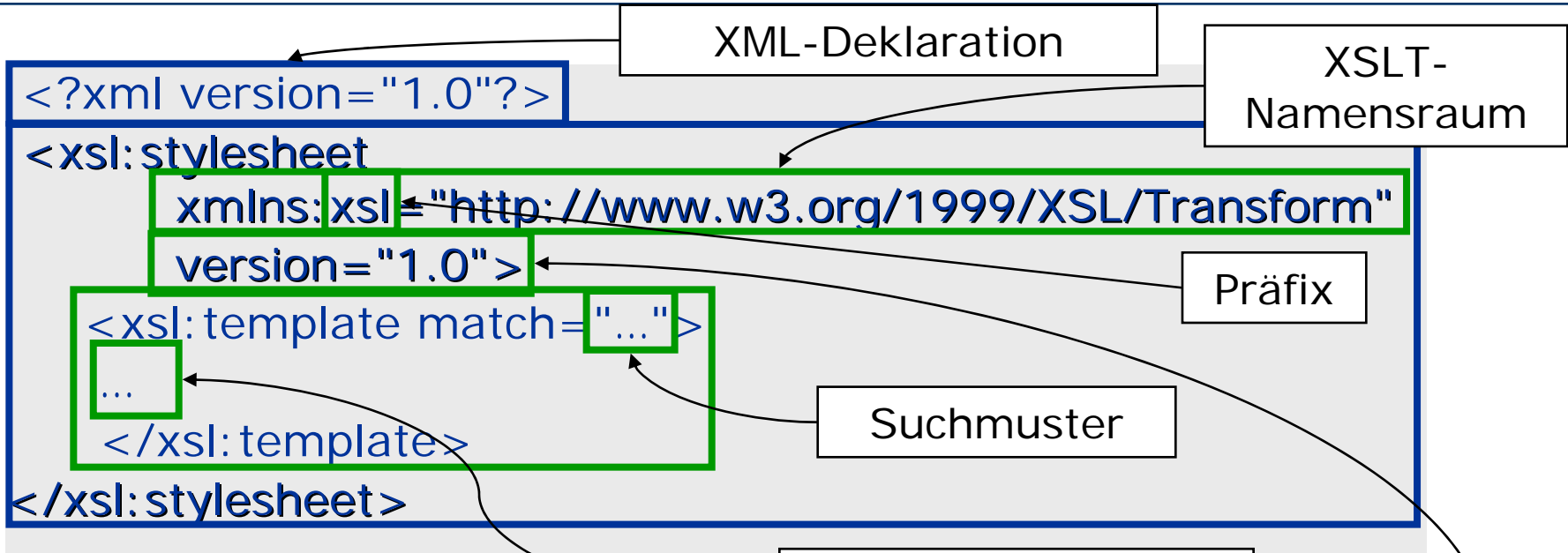
```
<p>Production-Class Widget</p>
```

Ursprungsdokument →
Ursprungsbaum (source
document → source tree)

Transformation

Ergebnisbaum →
Ergebnisdokument (result
tree → result document)

Grundstruktur von Stylesheets



- XML-Dokument
- Dokument-Wurzel:
 - `stylesheet` oder `transform` aus entsprechendem W3C-Namensraum
 - `stylesheet` und `transform` gleichbedeutend
 - obligatorisches Attribut: `version`

1. Neue Inhalte erzeugen (I)

- Templates können alle XML-Inhalte erzeugen: PCDATA, Elemente und Attribute
- einfach normale XML-Syntax verwenden:

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

- Beachte: Stylesheets müssen wohlgeformte XML-Dokumente sein, daher z.B. nicht erlaubt:

```
<xsl:template match="...">
  <br>neuer Text
</xsl:template>
```

1. Neue Inhalte erzeugen (II)

- 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>
```

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

Transformations-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>
```

Dokument

```
<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

Ausgabe

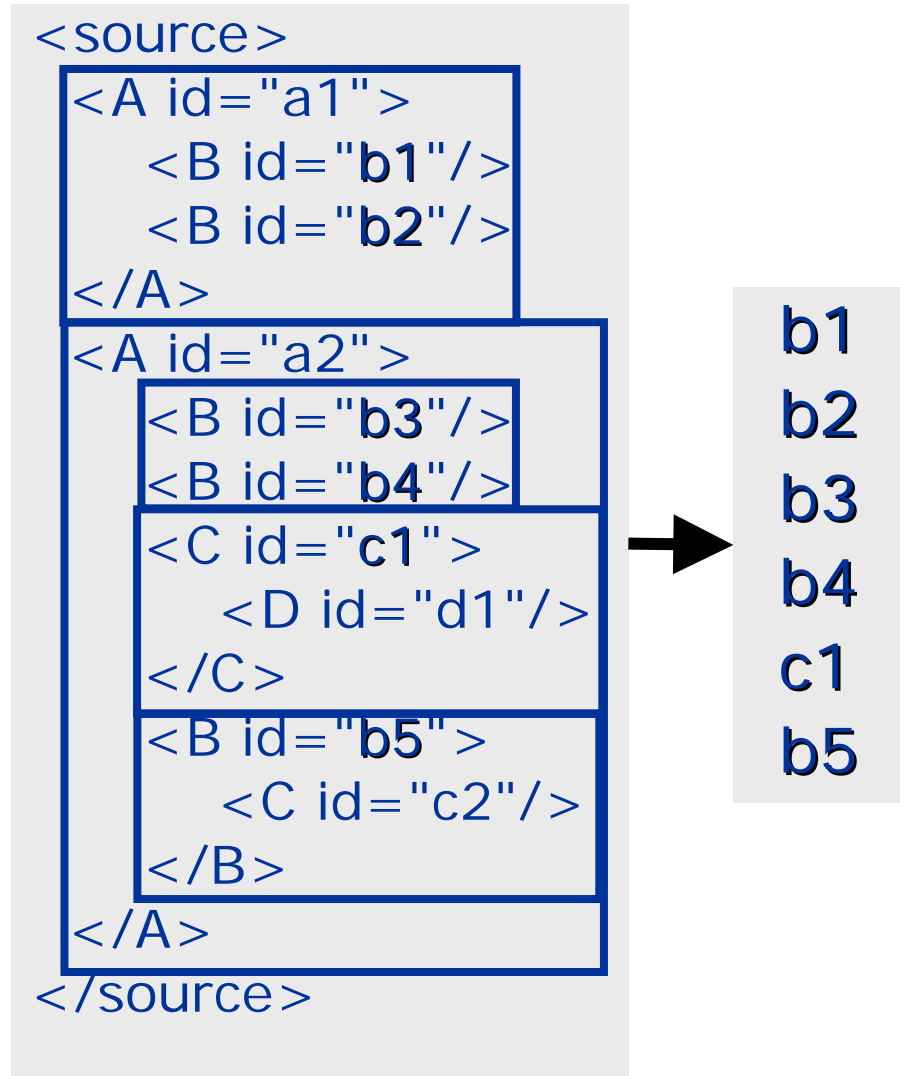
a1
a2

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

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/*".



<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>
```

Vordefinierte Templates

1.vordefinierte Template

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

2.vordefinierte 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



1.4 & 1.5. XPath & Co und XSLT → Lernziele

Lernziele

- XPath-Ausdrücke & XPath-Funktionen
- Wie funktioniert XSLT?
- Iteration/Rekursion bei XSLT
- Welche vordefinierten Templates gibt es?



2. Block: Web Services

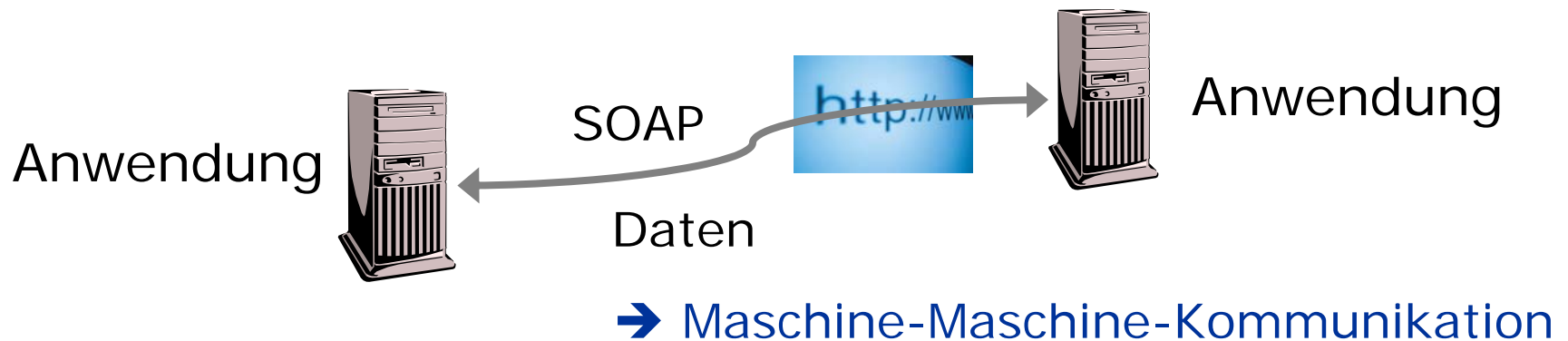


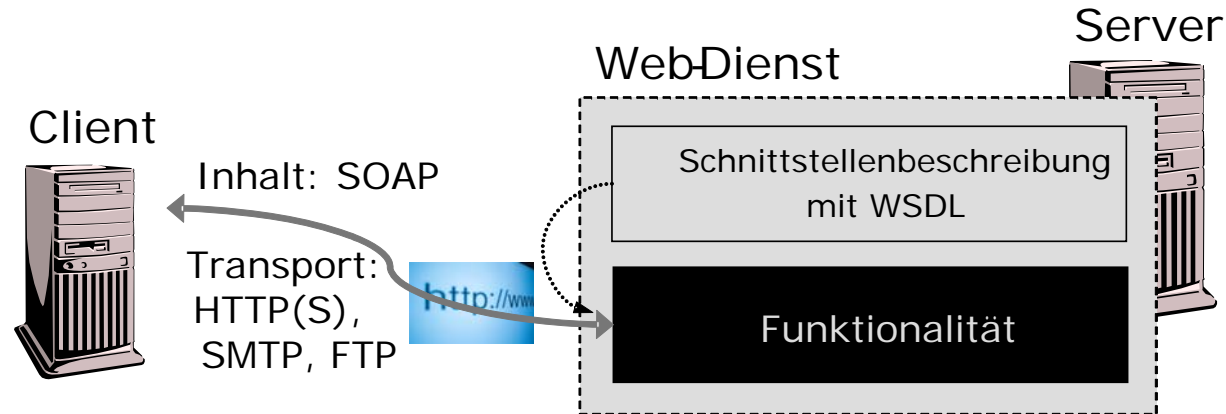
2.1. Web Services

traditionelle Web-Anwendung



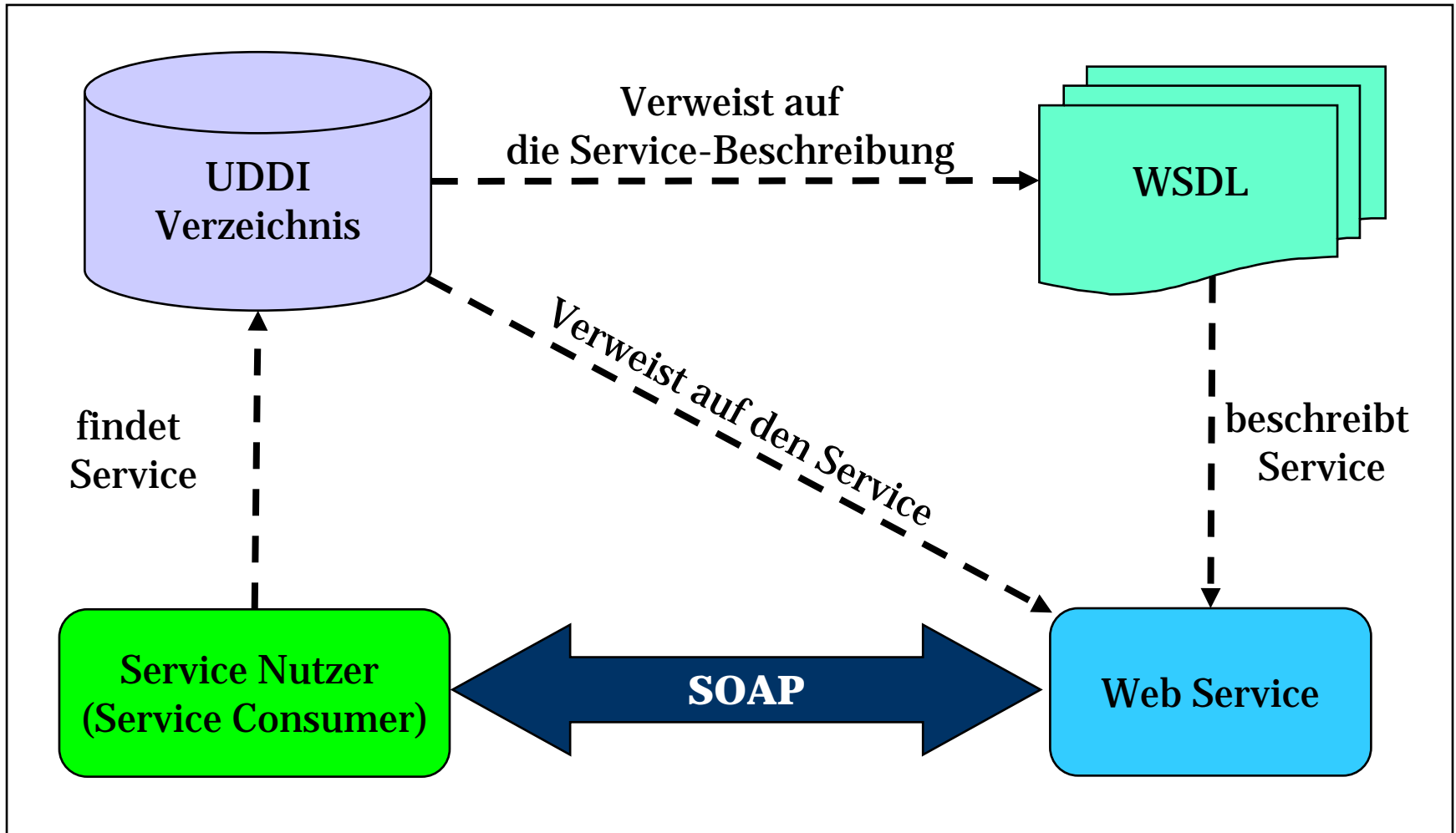
Web Service





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.





2.1. Web Services → Lernziel

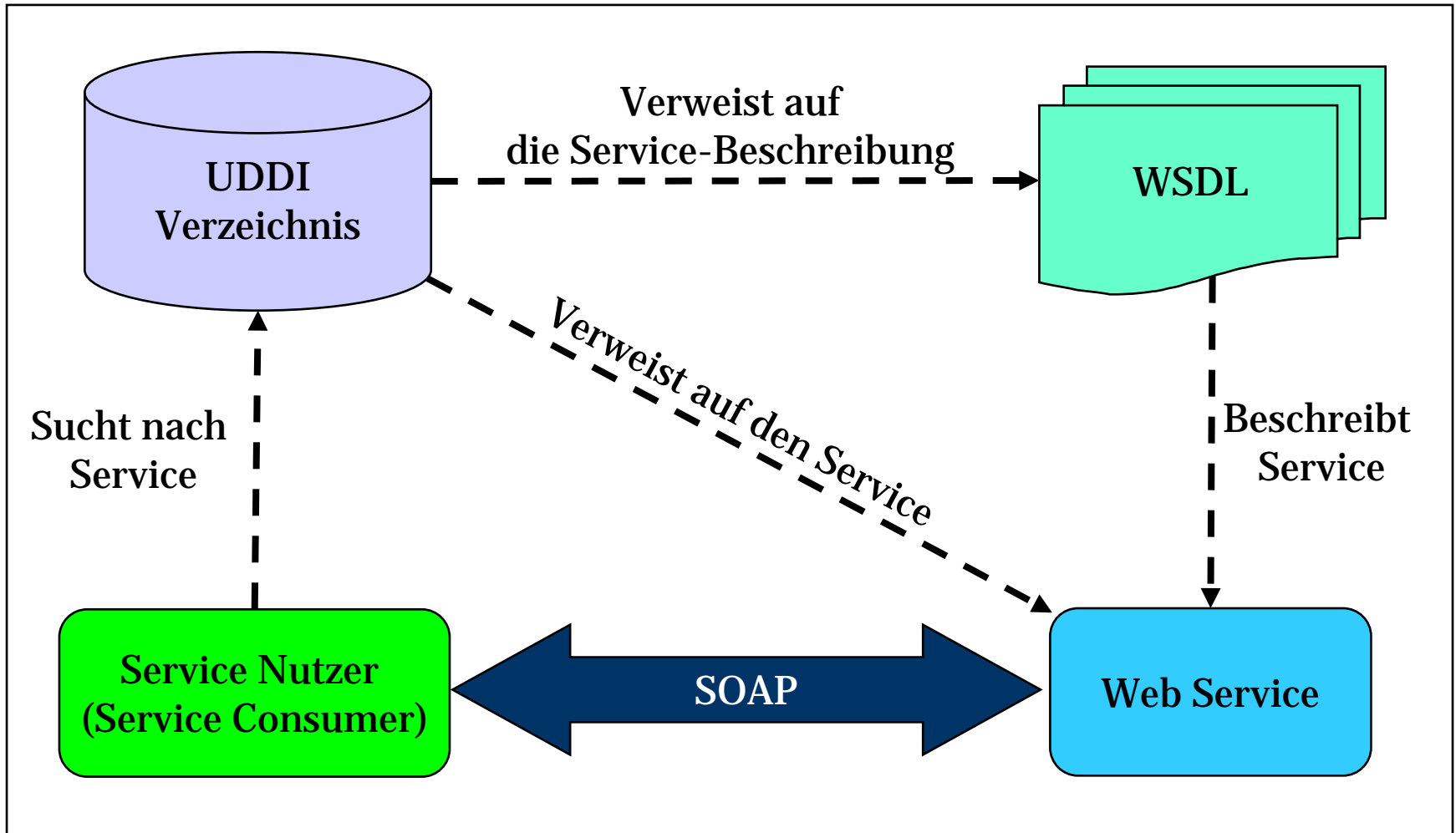
Lernziele

- Was sind Web Services?
- Haupttechnologien bei Web Services

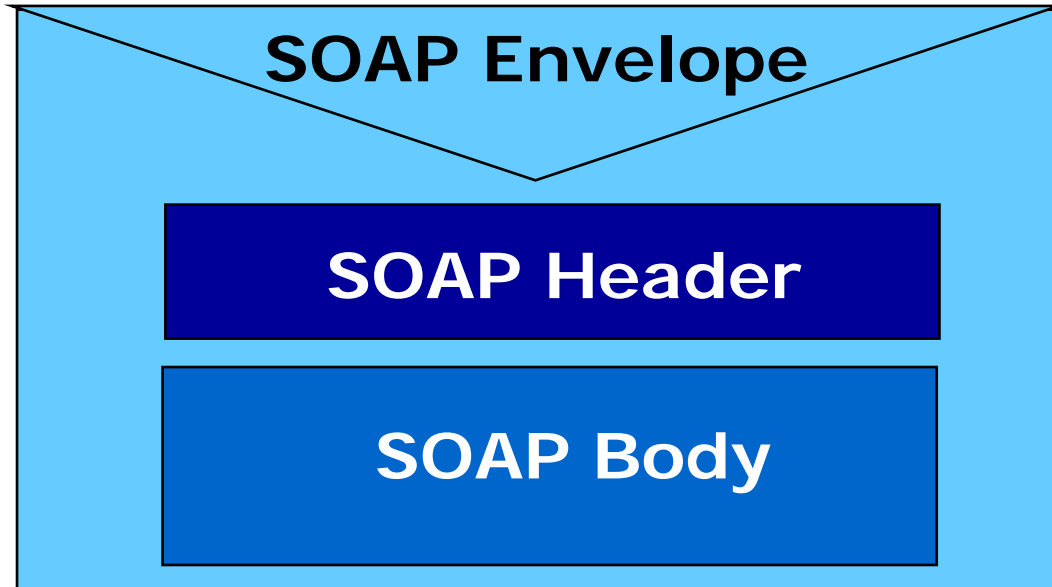


2.2. SOAP





- 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



XML-Deklaration

```
<?xml version='1.0' encoding='UTF-8'?>
```

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap/envelope/">  
  <!-- SOAP Header -->  
  <!-- SOAP Body -->  
</env:Envelope>
```

SOAP Version 1.2

SOAP Version 1.1:

<http://schemas.xmlsoap.org/soap/envelope/>

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>
```

```
<?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

Prinzipieller Aufbau (II)

- **Envelope: Wurzel-Element einer SOAP Nachricht**
 - beinhaltet SOAP Namespace
 - identifiziert SOAP Nachricht
- **Body: beliebige XML-Inhalte erlaubt**
 - Struktur von Anwendung festgelegt, z.B. durch:
 - speziellen Namensraum und/oder
 - WSDL-Beschreibung
- **Header: beliebige XML-Inhalte erlaubt**
 - Struktur von Anwendung festgelegt
 - Header Block
 - Kind-Element von Header
 - Zusatzinformation zur eigentlichen Nachricht

```
<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"



2.2. SOAP → Lernziel



Lernziele

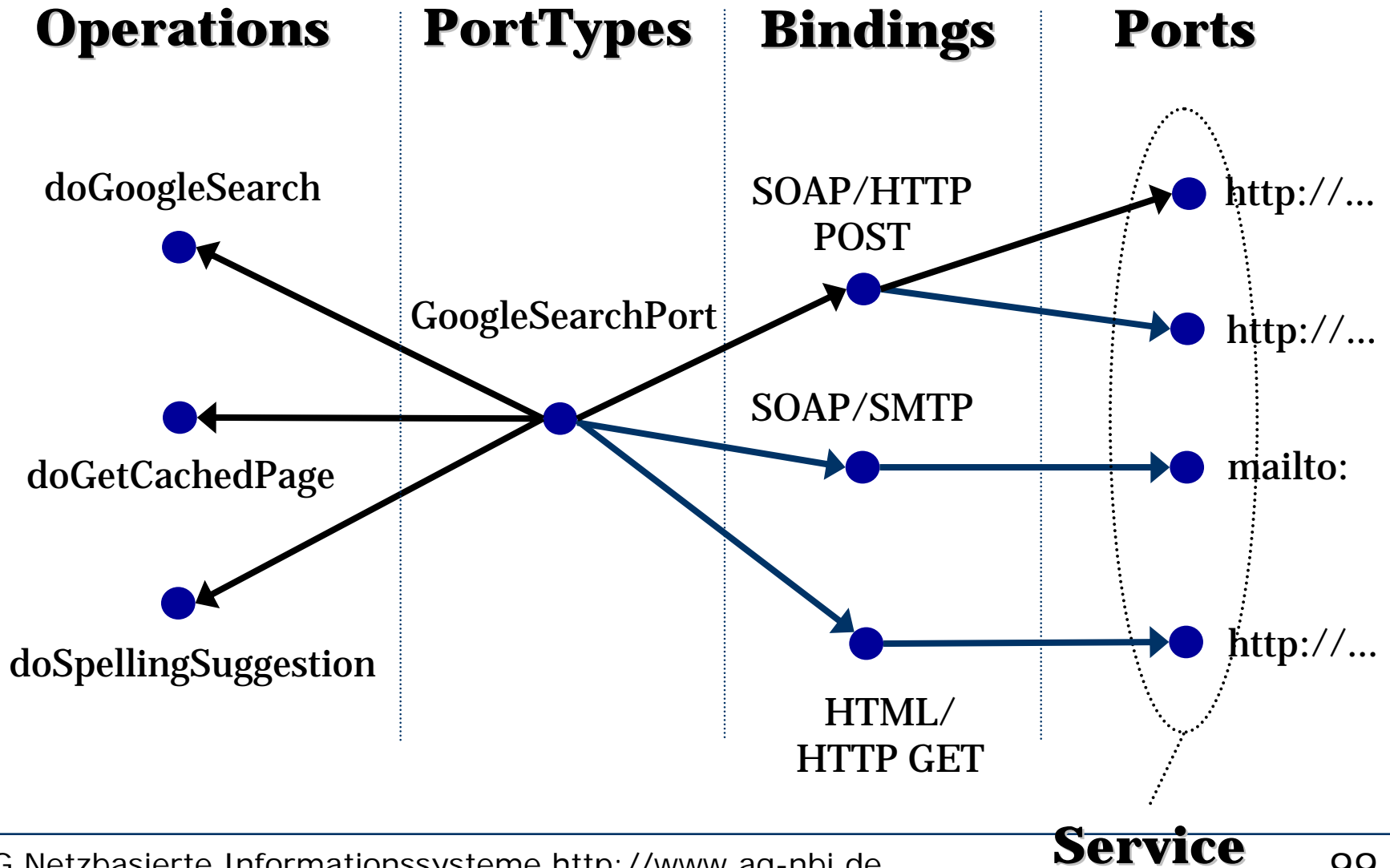
- Was ist SOAP?
- SOAP Version 1.1 vs. SOAP Version 1.2
- Aufbau einer SOAP-Nachricht
 - Eigenschaften von Envelope, Header & Body (!)
- SOAP Eigenschaften
 - Was ist ein Kodierungsstil in SOAP?
 - Was bedeutet „mustUnderstand“ Attribut?



2.3. WSDL

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)



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

Element	Beschreibung
Konkrete Beschreibung	
<code><binding>...</binding></code>	<ul style="list-style-type: none"> - Kommunikationsprotokoll, der beim Web Service benutzt wird - Gibt die Bindung(en) der einzelnen Operationen im portType-Abschnitt an
<code><service>...</service></code>	<ul style="list-style-type: none"> - gibt die Anschlussadresse(n) der einzelnen Bindungen an (Sammlung von einem oder mehrere Ports)

Datentyp / Nachricht / Porttyp

```
<types>
  <xsd:schema xmlns:xsd="..." xmlns:tns="..." targetNamespace="...">
    <xsd:complexType name="GoogleSearchResult">
      ...
    </xsd:complexType>
  </schema>
</types>
```

Definition des Datentyps

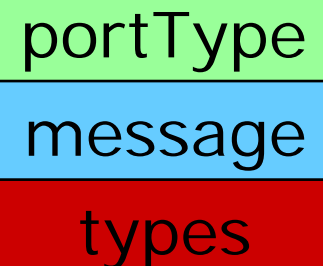
```
<message name="doGoogleSearch">...</message>
<message name="doGoogleSearchResponse">
  <part name="return" type="tns:GoogleSearchResult"/>
</message>
```

Definition einer abstrakten Nachricht

```
<portType>
  <operation name="doGoogleSearch">
    <input message="tns:doGoogleSearch"/>
    <output message="tns:doGoogleSearchResponse"/>
  </operation>
  ...
</portType>
```

Definition einer abstrakten Schnittstelle

portType
message
types



```
<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>
```



2.3. WSDL → Lernziel

Lernziele

- Was ist WSDL?
 - Was Beschreibt WSDL?
- Wozu nutzt man WSDL?
- Grundstruktur von WSDL (Reihenfolge!)
 - Abstrakte / Konkrete Definitionen in WSDL



2.4 Ausblick (Semantic Web)

“The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation.”

Berners-Lee, Hendler, and Lassila, 2001.



Foto: W3C

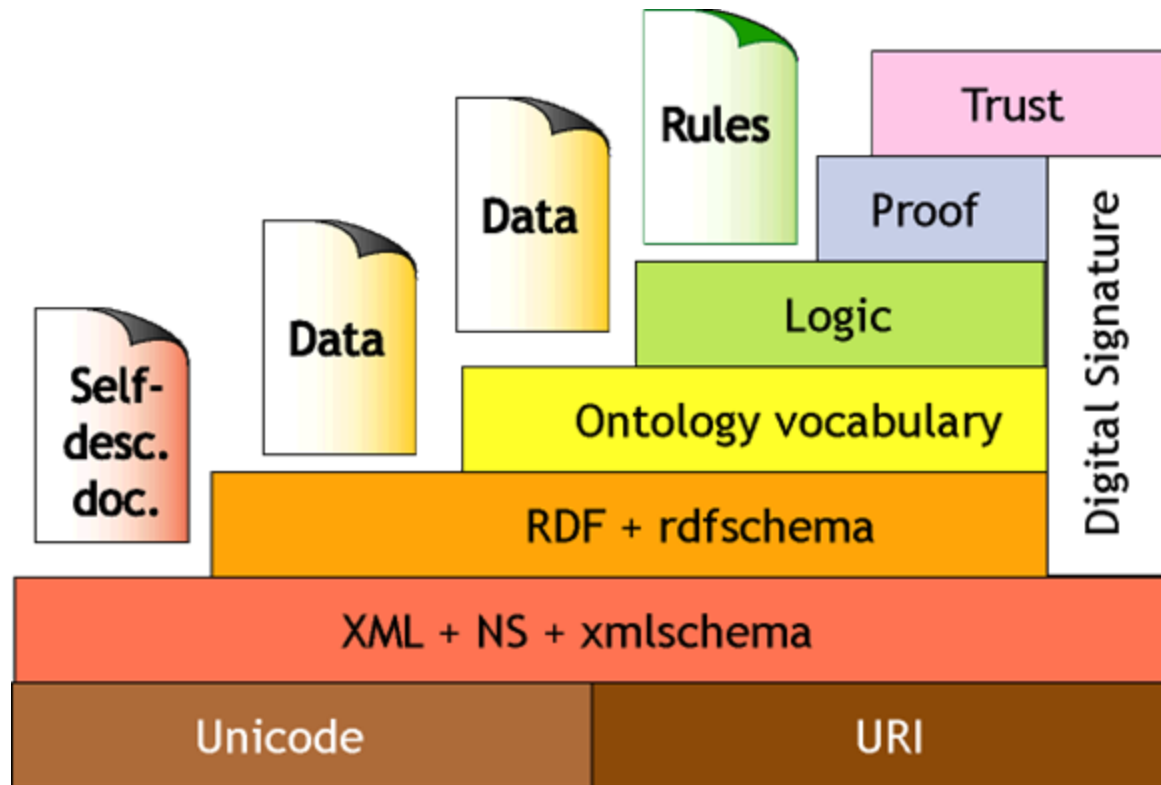


Foto: Homepage



Foto: Homepage

Semantic Web Stack (W3C, 2000)



auch „Semantic Web Layer Cake“ oder „Semantic Web Tower“ genannt

- RDF/XML Syntax Specification – W3C Recommendation seit Feb. 2004
 - RDF Tripel setzen bel. Web-Ressourcen miteinander in Beziehung
 - RDF Statement – die kleinste Informationseinheit, die ein Fakt darstellt
- Beispiel:
This presentation was created by Malgorzata Mochol
 - Subject (Ressource): *This presentation*
 - Predicate (Property): *creator*
 - Object (Wert): *Malgorzata Mochol*
- RDF benutzt URIs :
 - Subject: <http://nbi.inf.fu-berlin.de/SemWeb.ppt>
 - Predicate: <http://purl.org/dc/elements/1.1/creator>
 - Object: <http://page.mi.fu-berlin.de/mochol/>

<URI-s, URI-p, URI-o>



2.4. Semantic Web → Lernziel

Lernziele

- Was bedeutet Semantic Web?
 - Grundidee
 - Ziele
- Semantic Web Technologien / Stack (!)



Klausur → Organisatorisches

Organisatorisches

- ➔ 15 Juli um 14:00 **pünktlich**
- ➔ in Hörsaal Informatik

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

Struktur: Sie haben 90 Minuten

- 23 Multiple Choice Fragen (23 Punkte insgesamt)
- 7 ausführliche Fragen (67 Punkte insgesamt)

Beispiel: Multiple Choice

- Ähnlich zu den Fragen, die in der Übung vorgestellt wurden
- Aufgepasst: **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.

Punkte	Note
$\geq 85,5$	1,0
≥ 81	1,3
$\geq 76,5$	1,7
≥ 72	2,0
$\geq 67,5$	2,3
≥ 63	2,7
$\geq 58,5$	3,0
≥ 54	3,3
$\geq 49,5$	3,7
≥ 41	4,0

- **Fragen** sind nur auf **Deutsch**
- **Antworten** auf **Deutsch** oder **Englisch** möglich
- Keine eigenen (Wörter-)Bücher, Papiere usw. erlaubt
- Während der Prüfung, dürfen Sie Hand hochheben wenn Sie Hilfe brauchen/Frage haben
- Sitzordnung:
 - **jede zweite Reihe & jeder zweite Platz!**
- Studentenausweis + Ausweis mit Foto mitbringen!
- Handy aus!

Weitere Fragen?



Viel Glück!