XML-in-Action 2001

Mario Jeckle

XML Schema

dc soft GmbH
Softwareentwicklung und -vertrieb

DevGroup

msdn
Microsoft Developer Network

Inhaltsübersicht

Warum? benötigt XML *noch eine* Sprache zur Grammatikdefinition

Wozu? kann XML-Schema eingesetzt werden

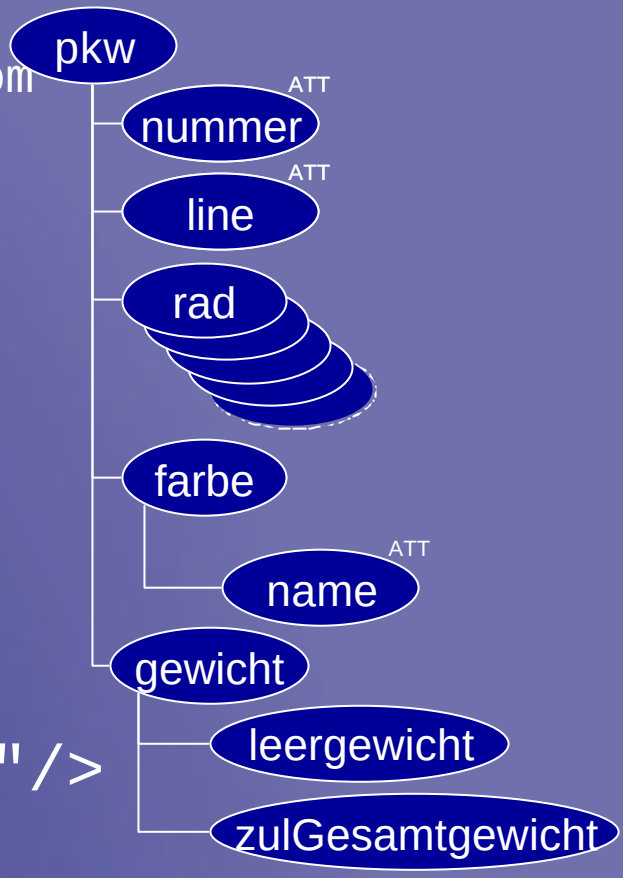
Wie? „funktioniert“ XML-Schema

Was? nützt Ihnen der Einsatz von XML-Schema

Der DTD-Mechanismus

Der DTD-Mechanismus

```
<?xml version="1.0" encoding="UTF-8" ?>
<pkw xmlns = "schema:www.daimlerchrysler.com"
     nummer = "S-NE 4229"
     line = "avantgarde">
   <rad> ... </rad>
   <rad> ... </rad>
   <rad> ... </rad>
   <rad> ... </rad>
   <farbe name = "Amethystviolett"/>
   <gewicht>
      <leergewicht>1485</leergewicht>
      <zulGesamtgewicht>1935</zulGesamtgewicht>
   </gewicht>
```



Der DTD-Mechanismus

```
<!ELEMENT pkw (rad+, farbe , gewicht)>
<!ATTLIST pkw
          nummer CDATA  #REQUIRED
          line CDATA #REQUIRED>
<!ELEMENT rad ANY>
<!ELEMENT farbe EMPTY>
<!ATTLIST farbe  name CDATA  #REQUIRED >
<!ELEMENT gewicht
         (leergewicht , zulGesamtgewicht)>
<!ELEMENT leergewicht (#PCDATA)>
<!ELEMENT zulGesamtgewicht (#PCDATA)>
```

Der DTD-Mechanismus

```
<!NOTATION non-empty SYSTEM
   "javascript:currentNode.value.length()>0;">
<!ELEMENT pkw (rad, rad, rad, rad?,
          farbe , gewicht)>
<!ATTLIST pkw
        nummer NOTATION (non-empty) "non-empty"
        line CDATA #REQUIRED>
<!ELEMENT rad ANY>
<!ELEMENT farbe EMPTY>
<!ATTLIST farbe  name
     (Dunkelbau|Firnweiß|Magmarot|...)
     #REQUIRED >
<!ELEMENT gewicht
         (leergewicht , zulGesamtgewicht)>
<!ELEMENT leergewicht (#PCDATA)>
<!ELEMENT zulGesamtgewicht (#PCDATA)>
```

Der DTD-Mechanismus

```
<!NOTATION non-empty SYSTEM
   "javascript:currentNode.value.length()>0;">
<!ELEMENT pkw (rad, rad, rad, rad?,
           farbe , gewicht)>
<!ATTLIST pkw
         nummer NOTATION (non-empty) "non-empty"
         line CDATA #REQUIRED>
<!ELEMENT rad ANY>
<!ELEMENT farbe EMPTY>
<!ATTLIST farbe  name
      (Dunkelbau|Firnweiß|Magmarot|...)
      #REQUIRED >
<!ELEMENT gewicht
          (leergewicht , zulGesamtgewicht)>
<!ELEMENT leergewicht (#PCDATA)>
<!ELEMENT zulGesamtgewicht (#PCDATA)>
```

Der DTD-Mechanismus

- Festlegung der Dokumentstruktur
  - Elemente
  - Attribute
  - Notationen
  - Entitäten (Textmakro auf Dokumentebene)
  - Parametrische Entitäten (Textmakro auf DTD-Ebene)
- Festlegung des Inhaltsmodells
  - Datentypen
    - Zeichenketten-artig
    - Vorgabewerte
    - Auswahltypen
  - Schlüssel/Referenzen

Der DTD-Mechanismus

- Streng hierarchische Sichtweise
- *Die-ganze-Welt-ist-ein-Dokument*-Sicht
  (von SGML übernommen)
  - ... mit entsprechenden Auswirkungen auf Referenzen
- Keine (praktisch verwendbare) Namespace Unterstützung
- DTD ist keine XML-Sprache
  - => zusätzliche Werkzeuge notwendig
- es existiert keine DTD für DTDs
  - => Grammatik kann nicht validiert werden
- Angebotene Datentypen unzureichend
- Angebotene Strukturprimitiven unzureichend
- => Notwendige Konstrukte zum Ausdruck mächtigerer Semantik müssen aufwendig und proprietär realisiert werden

Der DTD-Mechanismus

... Erweiterungsforderungen

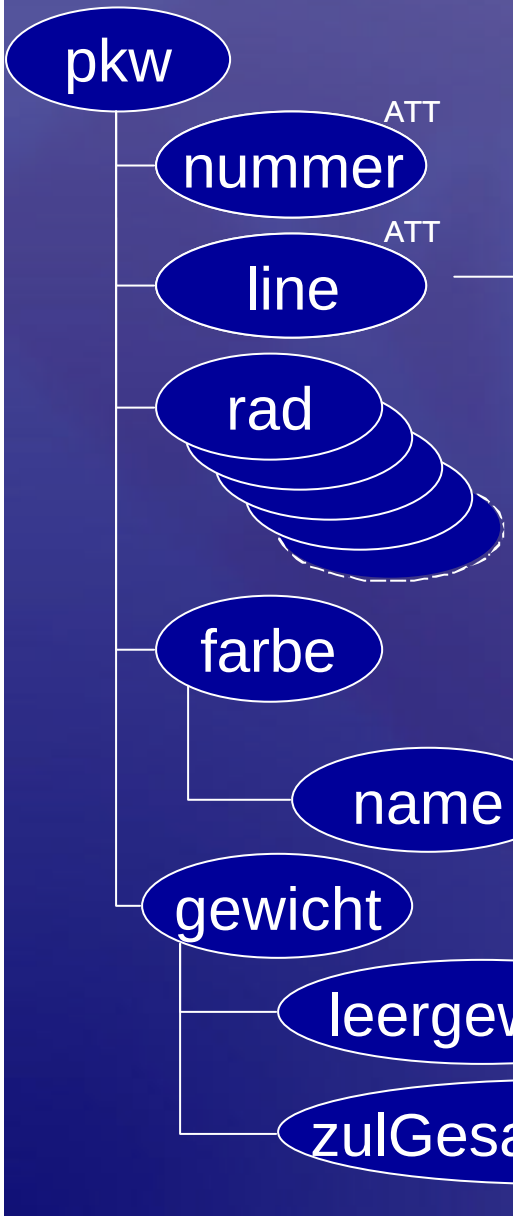
Strukturell ...

- Namespace Integration (und damit XML 2nd edition)
- Steuerung der Auftretensreihenfolge
  (Beschränkung und kontrollierte Freigabe)
- Vererbung (Kopier- und Substitutionssemantik)
- Wiederverwendungsunterstützung
- Praktisch einsetzbarer Referenzierungsmechanismus

Inhaltlich ...

- Primitive Datentypen (`int`, `float`, `boolean`, ...)
- Binärstrukturen
- Komplexe Datentypen (`date`, Elemente, Strukturen, ...)
- Eigendefinierte lexikalische Datentypen
- Konsistenzsichernde Einschränkungen

Der DTD-Mechanismus



... Erweiterungsforderungen

- Selbstdefinierte Typen (reguläre Ausdrücke)
- Kontextbezogene Einschränkung
- Beschränkung der Auftretenshäufigkeit (Kardinalität)
- Auswahl aus vorgegebener Liste (selbst definierter Aufzählungstyp)
- Einschränkung des zulässigen Wertebereichs (Domänenrestriktion)
- Beliebige Reihenfolge

Ein Schema für XML

Datentypen ...
- Validierung
- Vereinfachte Applikationsprogrammierung
- Separiertes Typkonzept

Namespaces ...
- Keine Unterstützung in DTDs
- Notwendig für integrierte Anwendung verschiedener XML-Sprachen

Syntax ...
- DTD-Syntax ist nicht XML
- Keine *Meta-DTD*

Gesteigerte Ausdrucksmächtigkeit ...
- Strukturen die nicht durch DTD-Mechanismen ausdrückbar sind

Lösungsalternativen

Erweiterungen des bestehenden (SGML-/XML-)DTD-Mechanismus

- Data Types for DTD (DT4DTD)

Wissensbeschreibung

- Document Content Description for XML (DCD)

  (*RDF basierte Weiterentwicklung von XML-Data*)

Inspiriert durch XML-API-Entwicklung

- Schema for Object oriented XML (SOX)

XML-Sprachen zur Inhaltsbeschreibung

- Document Definition Markup Language/XSchema (DDML)
- Schematron (XSL-basierte Auswertung der Dokumentstruktur)
- XML-Data/XML-Data Reduced (XDR) *(erster Ansatz noch vor Verabschiedung XML 1.0)*
- Document Structure Description (DSD)

XML-Schema

Zwei konkurrierende Philosophien:

- **Regelbasiert**
  - Definiert Validierungsregeln
  - Bekannteste Implementierung: *Schematron*
- **Reguläre Grammatik**
  - Definiert alle gültigen Dokumentstrukturen explizit
  - Bekannt aus Programmiersprachen
  - Implementierungen: *XSD, XML-Data, XDR, DSD, ...*

  - Stand 1998/9: Viele verschiedene (konkurrierende) Sprachen
    - Zunehmend kaum noch erkennbare inhaltliche Unterschiede
  - Ab Mai 1999: W3C Arbeitsgruppe entwickelt eigenen Ansatz
    - Trennung zwischen *Struktur* und *Datentypen*
    - Berücksichtigt (nahezu) alle bedeutenden Vorgänger (explizit: DCD, DDML, SOX, XDR, XML-Data)

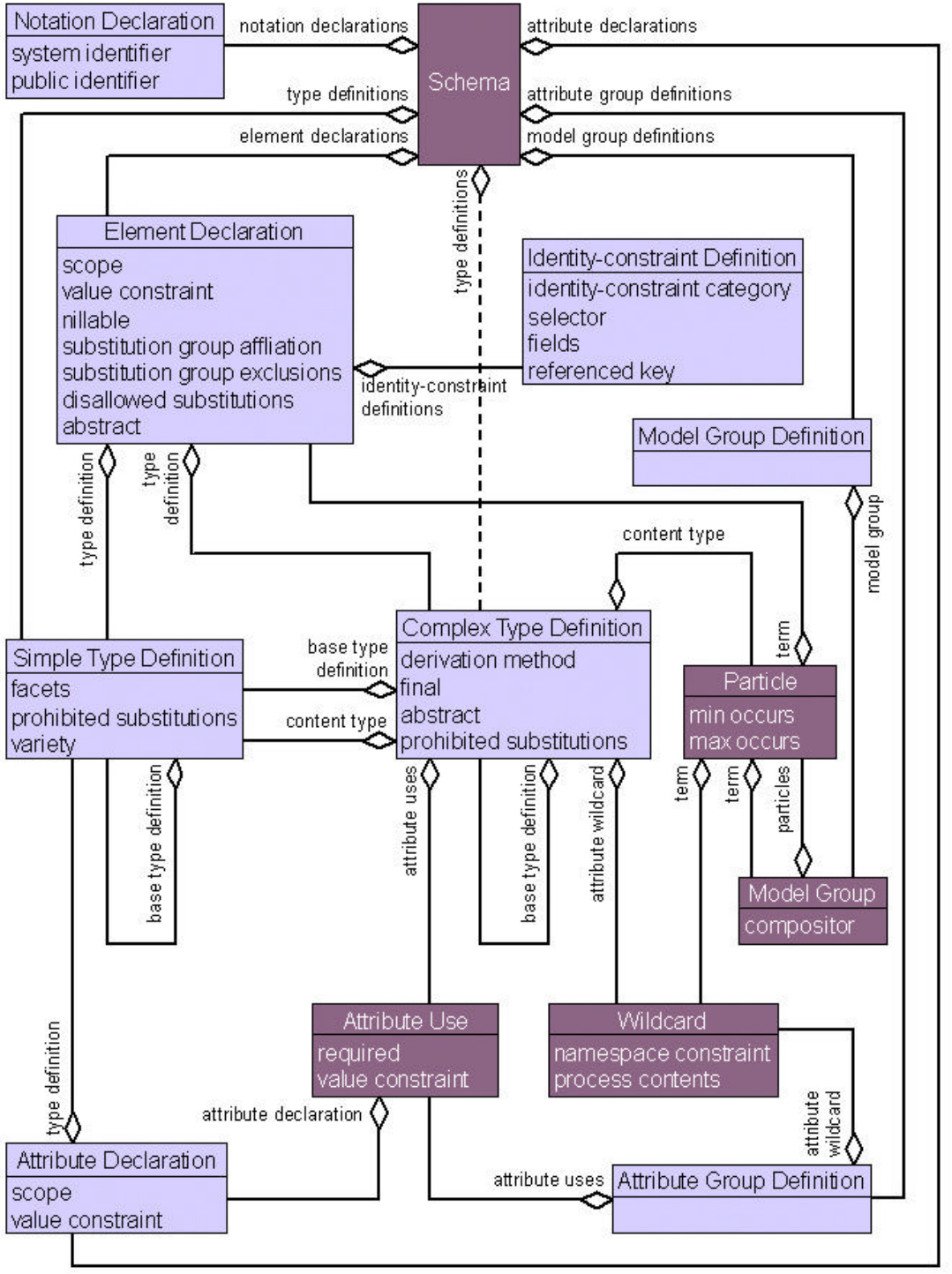
XML-Schema

... Mächtigkeit

- Attribute und Elemente (wie in DTDs)
- Namespace-Unterstützung
- Atomare Datentypen (`int`, `float`, `boolean`, ...)
- Anwenderdefinierte
  - atomare Datentypen
    - Einschränkung des Wertebereichs (Domänenrestriktion)
  - lexikalische Muster (reguläre Ausdrücke)
  - Aufzählungstypen
  - Mengentypen
  - komplexe Datentypen (`complexType`)
- Vererbung
  - Restriktion und Erweiterung
- Substitution
- Erweiterter Schlüsselmechanismus
- NULL-Werte

XSD
Component Data Model

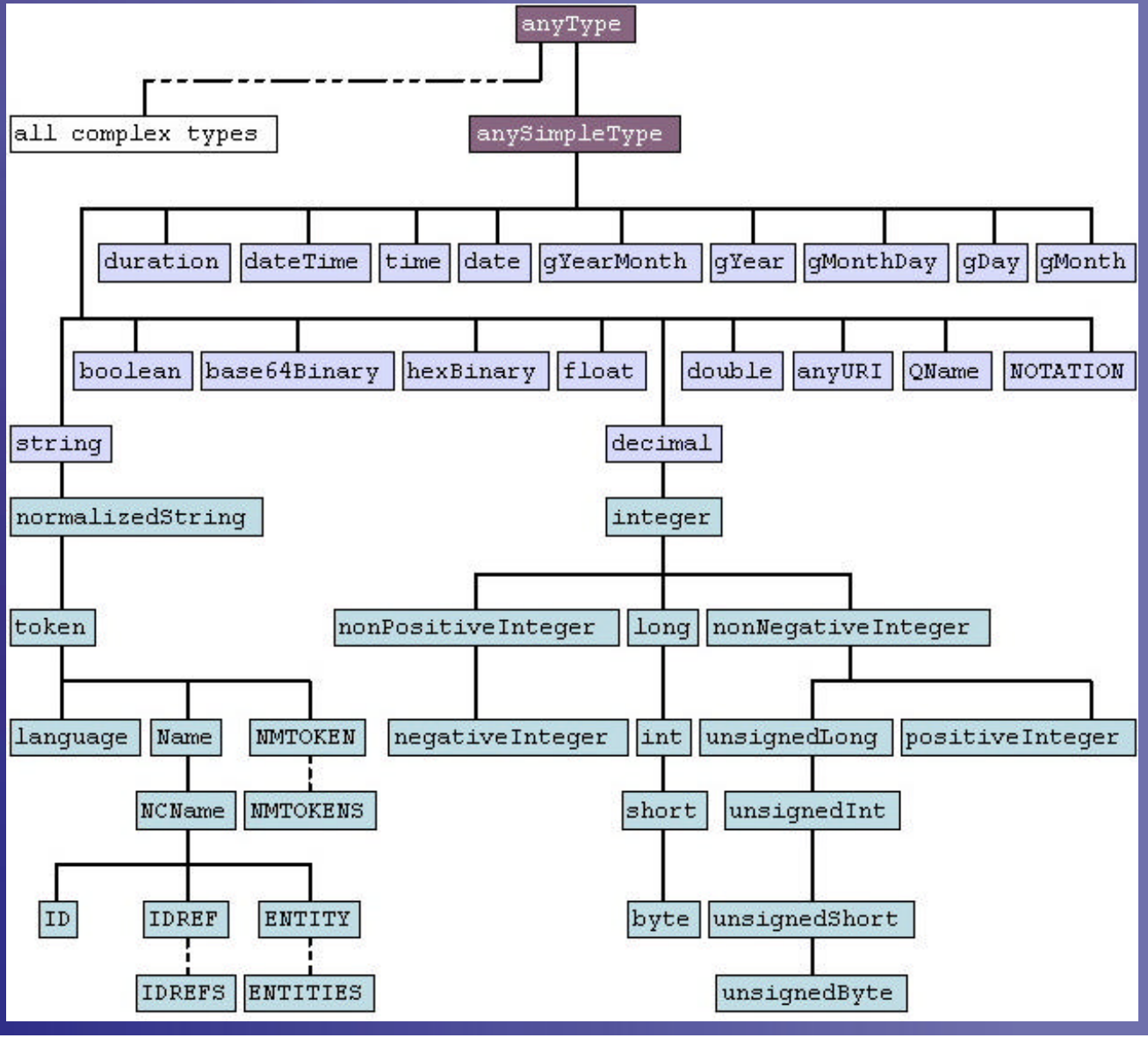
- Benannte Komponente mit *target namespace*
- Unbenannte Komponte



cf. XML-Schema Part 1: Structures, PR 2001-03-30, Appendix E

XSD -- Datentyphierarchie

- Ur-typ
- Vordefinierter Primitivtyp
- Vordefinierter abgeleiteter Typ
- —— Typeinschränkung
- ----- Aggregierter Typ



cf. XML-Schema Part 2: Datatypes, PR 2001-03-30, chap. 3

XML-Schema

Primitive Datentypen:

- string
- boolean
- decimal
- float
- double
- duration
- dateTime
- time
- date
- gYearMonth
- gYear
- gMonthDay
- gDay
- gMonth
- hexBinary
- base64Binary
- anyURI
- QName
- NOTATION

Abgeleitete Datentypen:

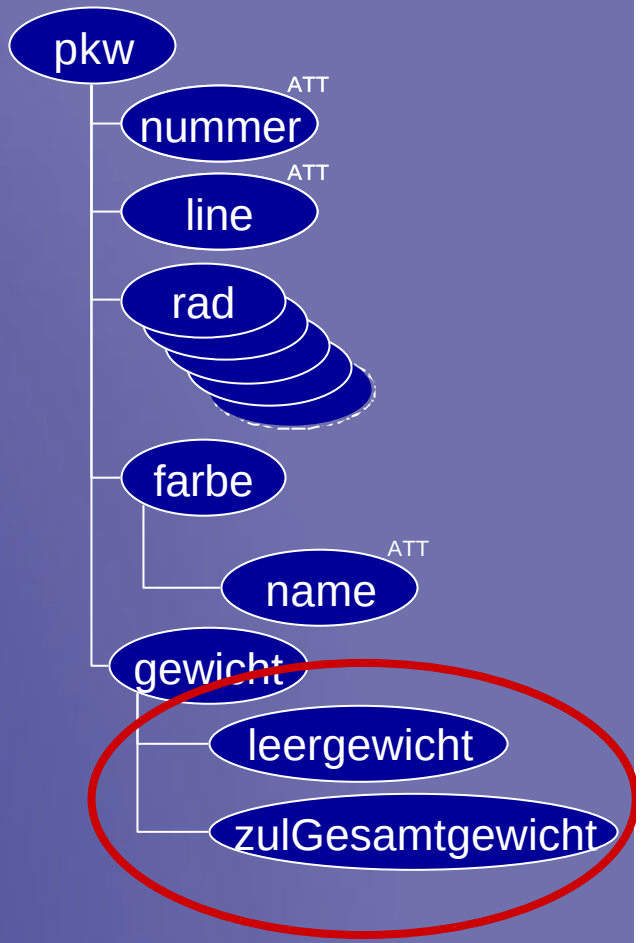
- normalizedString
- token
- language
- IDREFS
- ENTITIES
- NMTOKEN
- NMTOKENS
- Name
- NCName
- ID
- IDREF
- ENTITY
- integer
- nonPositiveInteger
- negativeInteger
- long
- int
- short
- byte
- nonNegativeInteger
- unsignedLong
- unsignedInt
- unsignedShort
- unsignedByte
- positiveInteger

XML-Schema

Datentypen:

```
<xsd:element name = "gewicht">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element
            name="leergewicht"
            type = "xsd:positiveInteger"/>
        <xsd:element
            name="zulGesamtgewicht"
            type="xsd:positiveInteger"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

... am Beispiel



Integer > 0

XML-Schema

Eigene Datentypen:

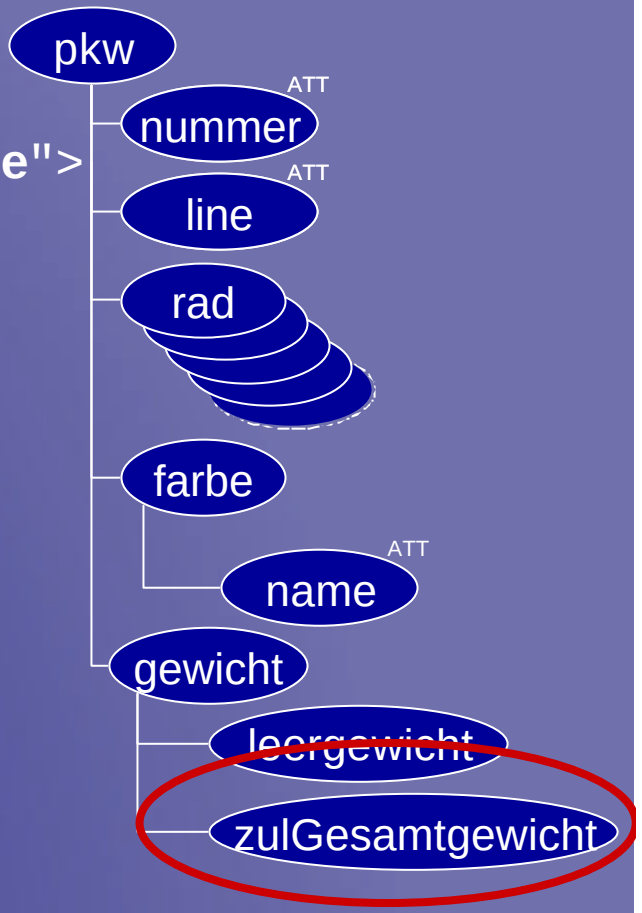
Definition:

```
<simpleType name="pkwZulGesamtgewichtType">
  <restriction base="decimal">
    <maxInclusive value="7500"/>
    <minInclusive value="0"/>
  </restriction>
</simpleType>
```

Verwendung:

```
<element
 name = "leergewicht"
 type = "xsd:positiveInteger"/>
<element
 name = "zulGesamtgewicht"
 type = "dcx:pkwZulGesamtgewichtType"/>
```

... am Beispiel



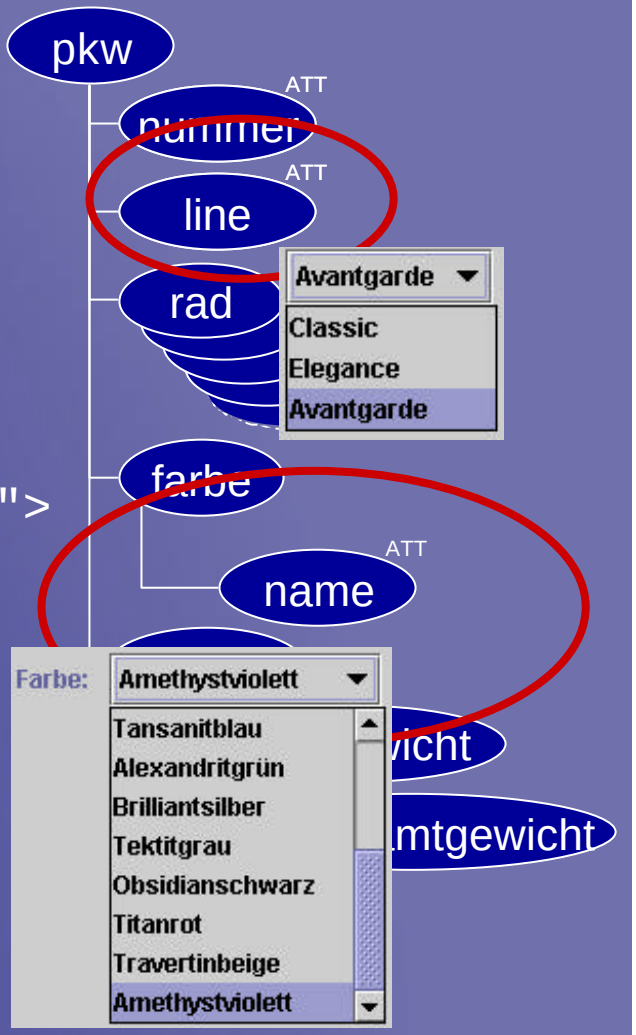
7500 >= zulGesamtgewicht >= 0

XML-Schema

Eigene Datentypen (Vorgabewerte):

```
<xsd:element name = "farbe">
  <xsd:complexType>
     <xsd:attribute
        name="name"
        use="required">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
<xsd:enumeration value = "Dunkelblau"/>
<xsd:enumeration value = "Firnweiß"/>
...
          </xsd:restriction>
        </xsd:simpleType>
     </xsd:attribute>
  </xsd:complexType>
</xsd:element>
```

... am Beispiel



XML-Schema

... am Beispiel

Eigene Datentypen (lexikalische Ausdrücke):

Definition:

```
<xsd:simpleType name = "autoNummerType">
  <xsd:restriction base = "xsd:string">
    <xsd:pattern
    value="[A-Z]{1,3}-[A-Z]{1,2} \d{1,4}"/>
  </xsd:restriction>
</xsd:simpleType>
```

Verwendung:

```
<xsd:element
  name = "nummer"
  type = "autoNummerType"/>
```

Definition eigener spezialisierter Typen erhöht Sicherheit und Wiederverwendbarkeit

XML-Schema

Eigene komplexe Typen:

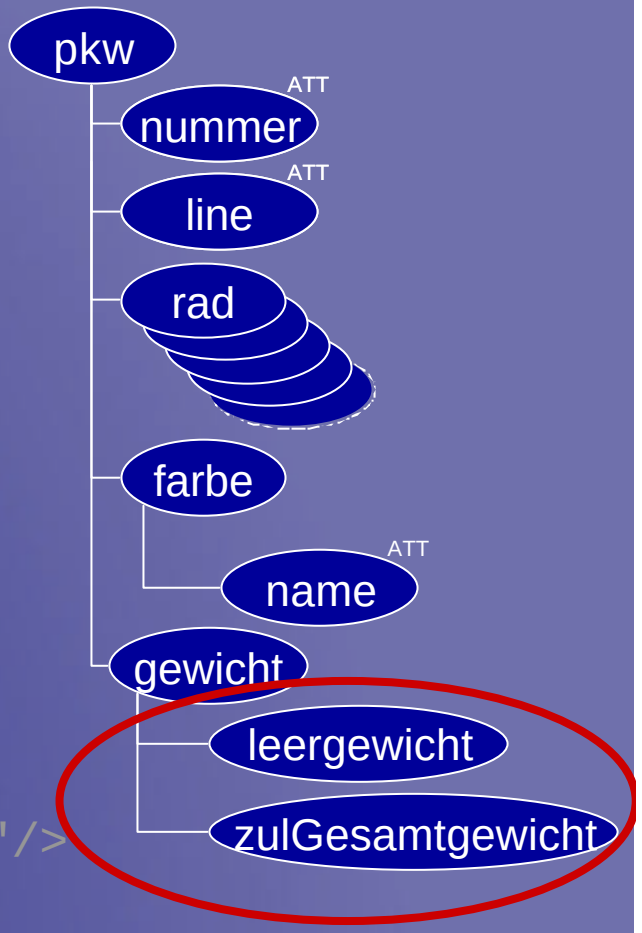
... am Beispiel

Definition:

```
<complexType name="gewichtType">
  <all>
    <element
     name = "leergewicht"
     type = "xsd:positiveInteger"/>
    <element
     name = "zulGesamtgewicht"
     type = "dcx:pkwZulGesamtgewichtType"/>
  </all>
</complexType>
```

Verwendung:

```
<element
 name = "nummer" type = "dcx:autoNummerType"/>
<element
 name = "gewicht" type = "dcx:gewichtType"/>
```



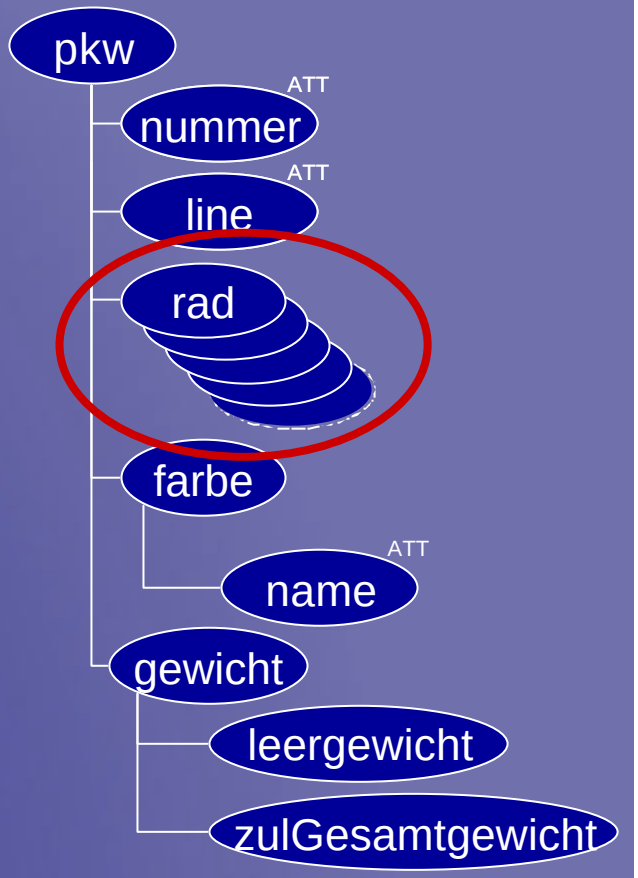
Definition eigener spezialisierter Typen erhöht Sicherheit und Wiederverwendbarkeit

XML-Schema

Strukturen (Auftretenshäufigkeit):

```
<xsd:element name = "pkw">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element
       name = "rad"
       minOccurs = "4"
       maxOccurs = "5"/>
...
</xsd:sequence>
```

... am Beispiel



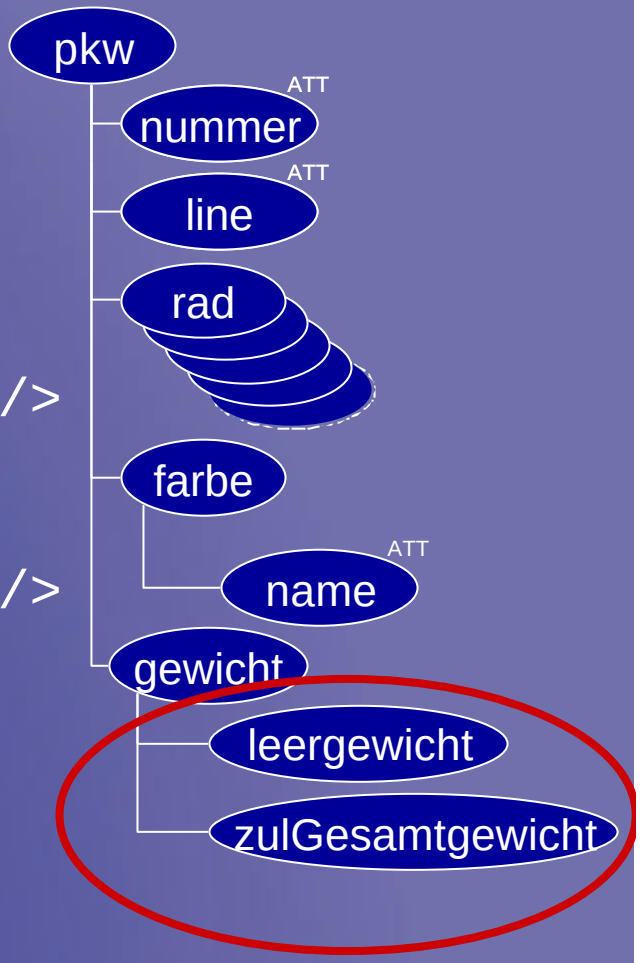
XML-Schema

Strukturen ((beliebige) Reihenfolge):

```
<xsd:element name="gewicht">
  <xsd:complexType>
    <xsd:all>
      <xsd:element
        name = "leergewicht”
        type = "xsd:positiveInteger"/>
      <xsd:element
        name = "zulGesamtgewicht"
        type = "xsd:positiveInteger"/>
    </xsd:all>
  </xsd:complexType>
</xsd:element>
```

Alle Elemente einer *all*-Gruppe können in beliebiger Reihenfolge auftreten.

... am Beispiel



XML-Schema

... Namespaces

XML-Dokument:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<pkw
  xmlns = "schema:www.daimlerchrysler.com"
  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation = "schemas.dcx.com?name=example?type=XSD"
  nummer = "S-NE 4229"
  line = "Avantgarde">
    <rad/>
    <rad/>
    <rad> ... </rad>
    <rad> ... </rad>
    <farbe name="Amethystviolett"/>
    <gewicht>
      <leergewicht>1485</leergewicht>
      <zulGesamtgewicht>1935</zulGesamtgewicht>
   </gewicht>
</pkw>
```

XML-Schema

XML-Schema:

... Namespaces

```
<schema xmlns = "http://www.w3.org/2001/XMLSchema"
        targetNamespace = "schema:www.daimlerchrysler.com"
        xmlns:dcx = "schema:www.daimlerchrysler.com"
        xmlns:xsd = "http://www.w3.org/2001/XMLSchema"
        elementFormDefault = "qualified">
  <simpleType name="autoNummerType">...</simpleTyple>
  <element name = "nummer"
           type = "dcx:autoNummerType"/>
  <element name = "gewicht">
    <complexType>
      <all>
        <element name="leergewicht"
          type="xsd:positiveInteger"/>
        <element name="zulGesamtgewicht"
          type="dcx:pkwZulGesamtgewichtType"/>
      </all>
    </complexType>
  </element>
...
</schema>
```

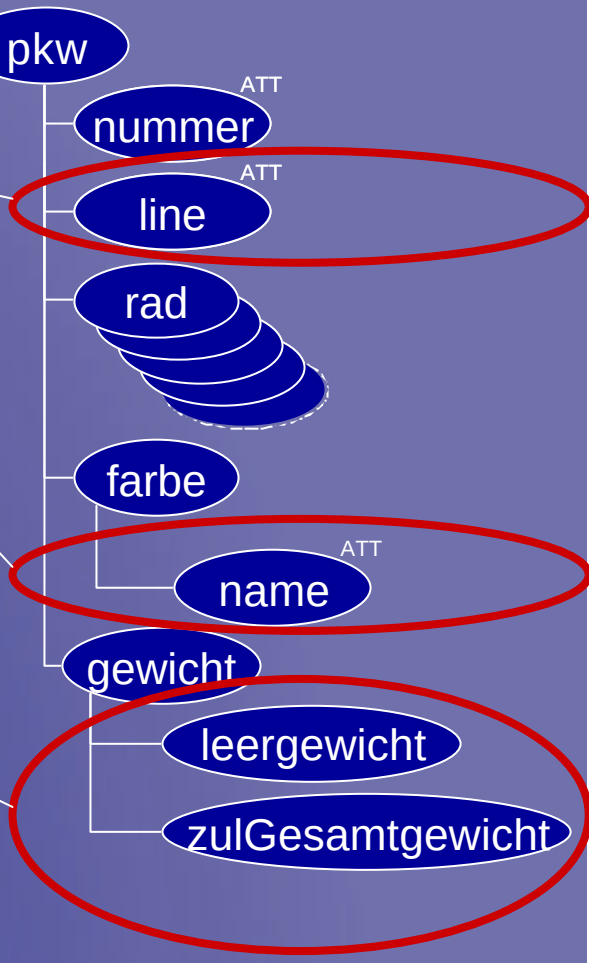
XML-Schema

... Grenzen

Konsistenzregeln:

```
farbe/@name = "Amethystviolett"
=>
line = "Avantgarde"
```

```
leergewicht <= zulGesamtgewicht
```



XSD erlaubt als reguläre Sprache
keine inhaltsabhängigen Konsistenzregeln

XML-Schema

... Grenzen, und ihre Überwindung

```
farbe/@name = "Amethystviolett"
=>
line = "Avantgarde"
```

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<schema xmlns='http://www.ascc.net/xml/schematron'>
  <ns prefix='dcx' uri='schema:www.daimlerchrysler.com'/>
  <pattern name = 'Farbe Test'>
    <rule context = "dcx:pkw">
      <assert test = "@line ='Avantgarde' and
       (//dcx:farbe/@name='Amethystviolett'
       or //dcx:farbe/@name='Dunkelblau'
       or //dcx:farbe/@name='Firnweiß'
       ...
       )">Farbe Amethystviolett ist nur
          für Line Avantgarde zugelassen</assert>
    </rule>
  </pattern>
</schema>
```

Beispiel in der Schemasprache *Schematron*

XML-Schema

... Grenzen, und ihre Überwindung

leergewicht <= zulGesamtgewicht

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<schema xmlns='http://www.ascc.net/xml/schematron'>
  <ns prefix='dcx' uri='schema:www.daimlerchrysler.com'/>
  <pattern name = 'Gewicht Test'>
    <rule context = "dcx:gewicht">
      <assert test = "leergewicht &gt; zulGesamtgewicht">
      Leergewicht übersteigt zulässiges Gesamtgewicht
      </assert>
    </rule>
  </pattern>
</schema>
```

Beispiel in der Schemasprache *Schematron*

Referenzen

W3C's XML-Schema:

- Part 0: Primer: `http://www.w3.org/TR/xmlschema-0/`
- Part 1: Structures: `http://www.w3.org/TR/xmlschema-1/`
- Part 2: Datatypes: `http://www.w3.org/TR/xmlschema-2/`

Einige Schema-Ansätze:

- DT4DTD: `http://www.w3.org/TR/dt4dtd`
- DCD: `http://www.w3.org/TR/NOTE-dcd`
- SOX: `http://www.w3.org/TR/NOTE-SOX/`
- DDML: `http://www.w3.org/TR/NOTE-ddml`
- XML-Data: `http://www.w3.org/TR/1998/NOTE-XML-data/`
- XDR: `http://www.ltg.ed.ac.uk/~ht/XMLData-Reduced.htm`
- DSD: `http://www.brics.dk/DSD/`
- Schematron: `http://www.ascc.net/xml/resource/schematron/schematron.html`