
# XML-in-Action 2001

**Mario Jeckle**

# XML Schemagenerierung

dc soft GmbH
Softwareentwicklung und -vertrieb

DevGroup

msdn
Microsoft Developer Network

# Inhaltsübersicht

- Herausforderung: Erstellung von XML-Sprachen
- XML und OO
- Ansatz: Generierung von XML-Strukturen
- Ein Beispiel ...
  - Klassendiagramm
  - Schrittweise XSD-Generierung

# XML everywhere ...


- Zugriff auf, und Nutzung von Daten, Systemen und Diensten erfolgt zunehmend XML-basiert.
- XML wird zum ASCII des 21. Jahrunderts

# Herausforderungen

- Erzeugung von XML-Sprachen
  - flexibel

    (*zukunftssichere Sprachen*)
  - zeitnah

    (*schnelle Erzeugung von Sprachvarianten*)
  - korrekt

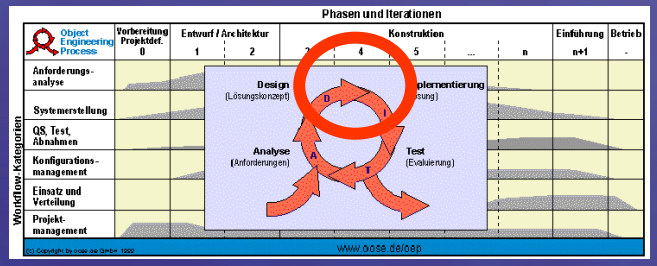
    (Sprache soll System(daten)strukturen wiederspiegeln
  - reproduzierbar

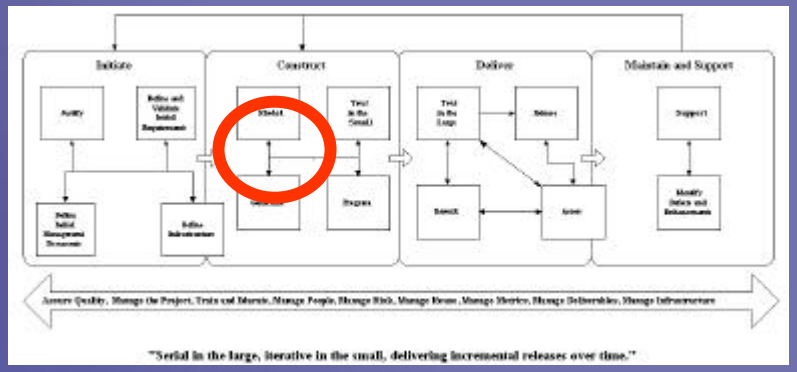
    (ähnliche Strukturierungsprinzipien in verschiedenen XML-Sprachen)
  - integriert

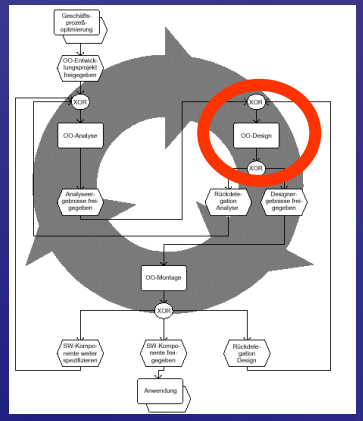
    (Eingliederung in bestehende Entwicklungsprozesse)

- Übergang von DTDs zu Schemasprachen

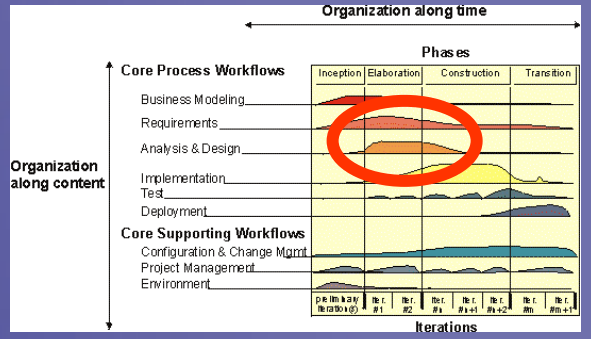
# Objektorientierung und XML?!



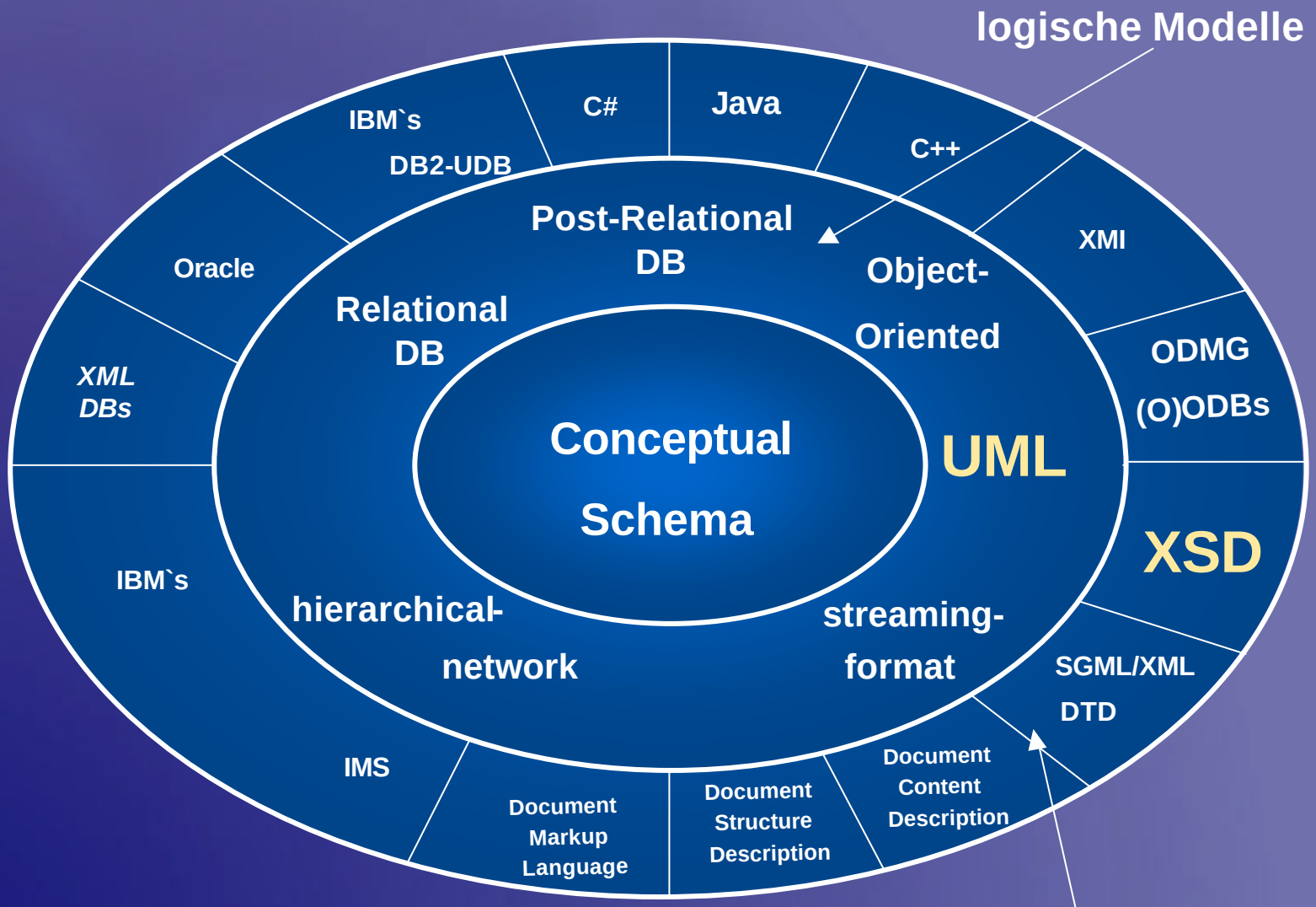




(XML-)Schemaentwicklung als Teil des OO-Designprozesses?!



# UML und XML

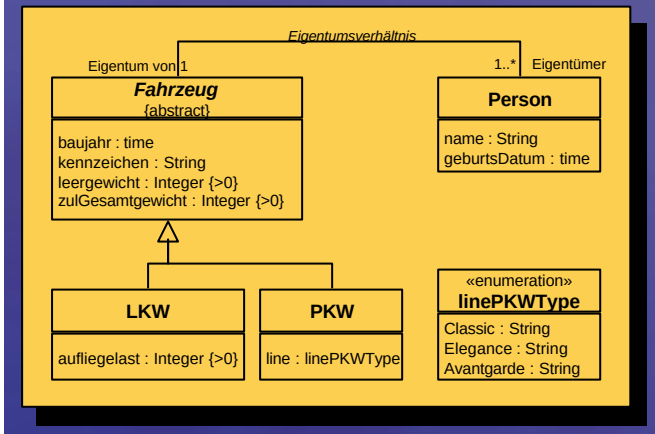

logische Modelle
IBM`s
DB2-UDB
C#
Java
C++
XMI
ODMG
(O)ODBs
XSD
SGML/XML
DTD
Document
Content
Description
Document
Structure
Description
Document
Markup
Language
IMS
IBM`s
XML
DBs
Oracle
Post-Relational
DB
Object-
Oriented
Relational
DB
Conceptual
Schema
UML
hierarchical-
network
streaming-
format
physische Modelle

# UML-Modell enthält ...

- Strukturinformation als Klassendiagramm
  - Klassen
  - Attribute
  - Assoziationen
  - Generalisierungsbeziehungen
  - Kardinalitäten
- Datentypen
  - Nutzung der vordefinierten
    `(Integer, UnlimitedInteger, String, Name, Boolean, ...)`
  - Selbstdefinierte
    - implizit durch Verwendung (innerhalb Attributdefinition)
    - explizit durch Definition als Aufzählungstyp
- Konsistenzregeln
  - ... soweit nicht bereits strukturell und inhaltlich abgedeckt, *Tagged Values, Properties,* OCL-Ausdrücke

# Lösungsansatz

## ... Generierung aus objektorientierten Strukturen



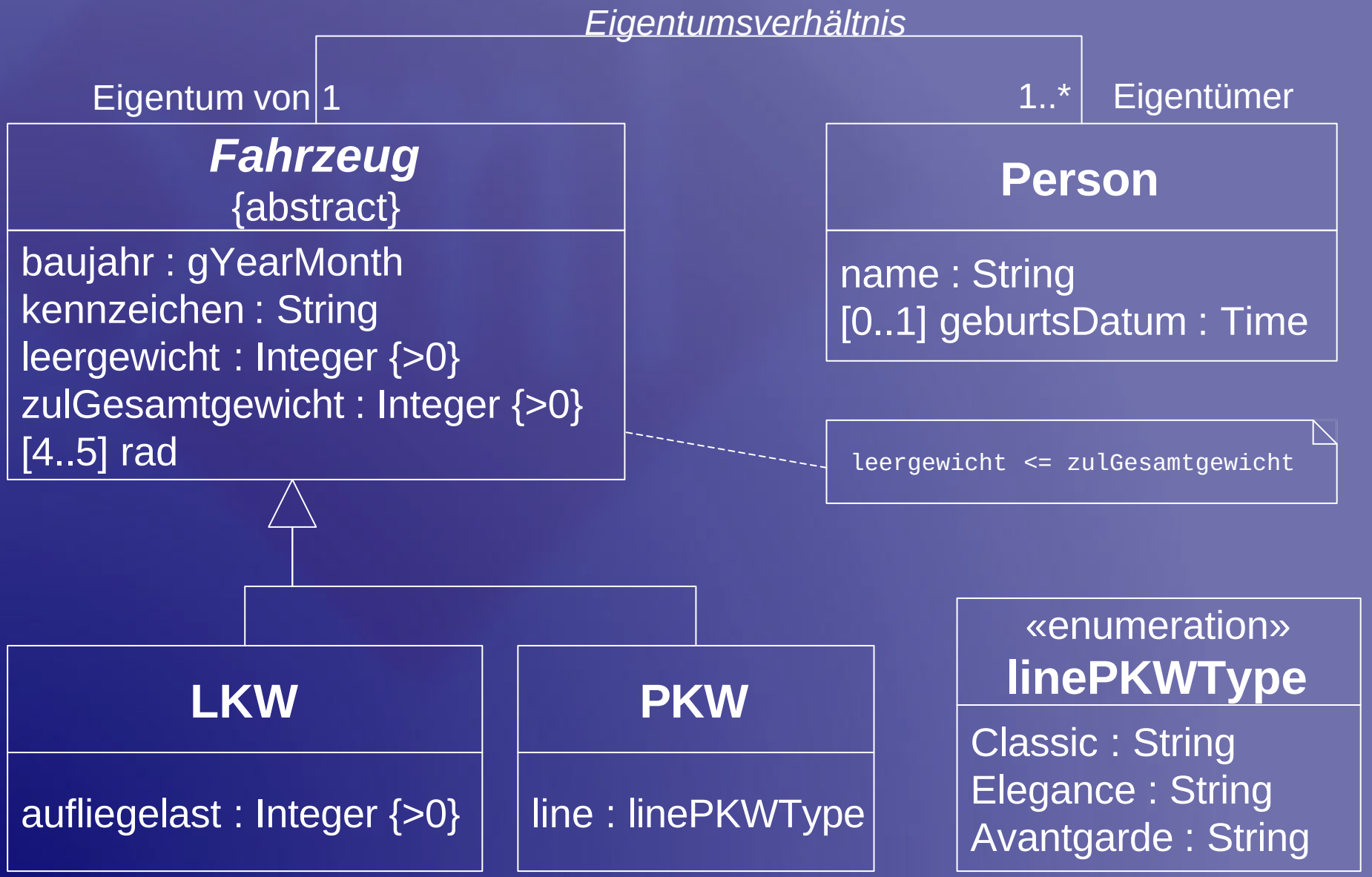

1. XMI-Export aus UML-Werkzeug
2. Transformation XMI[UML] nach XSD
3. Weiterverwendung des erzeugten Schemas

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE XMI SYSTEM "ad99-10-05_UML.dtd">
<XMI xmi.version="1.1"
 xmlns:UML="//org.omg/UML/1.3"
 timestamp="Sun Apr 01 15:31:09 2001">
  <XMI.header>
    <XMI.documentation>
      <XMI.exporter>Unisys.JCR.1</XMI.exporter>
      <XMI.exporterVersion>1.3</XMI.exporterVe
      Plus some necessary hand editing ...
    </XMI.documentation>
    <XMI.metamodel xmi.name="UML"
     xmi.version="1.3"/>
  </XMI.header>
  ...
```

XSLT

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Example">
    <xsd:complexType mixed="true">
      <xsd:choice>
        <xsd:element ref="Fahrzeug"
         minOccurs="0"
         maxOccurs="unbounded"/>
        <xsd:element ref="Person"
         minOccurs="0"
         maxOccurs="unbounded"/>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
...
```

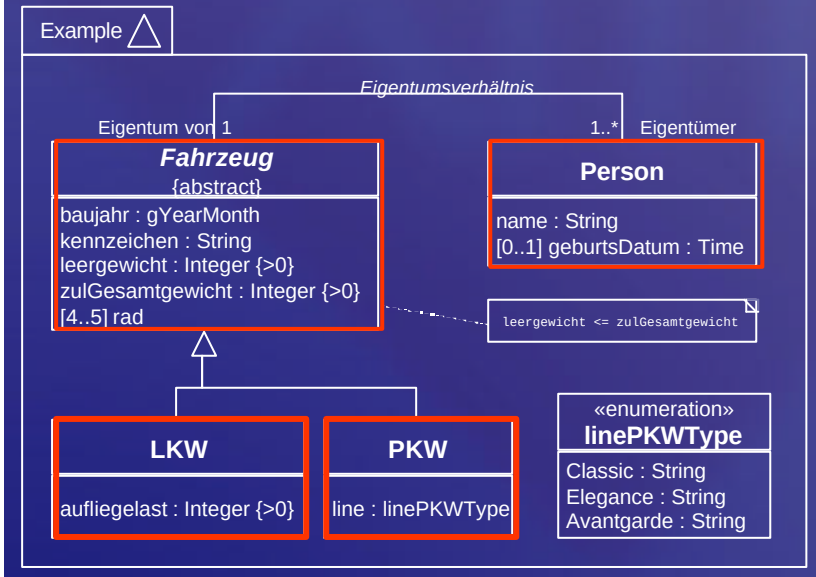
# Ein Beispiel ...



# Von UML zum XML-Schema

Regel:

- Modell --> complexType
- Klasse --> complexType

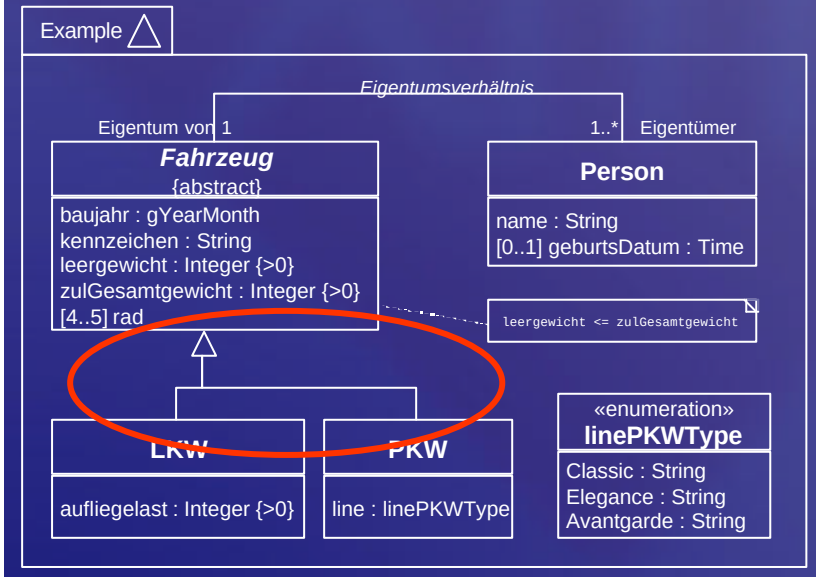



```
<xsd:complexType name="ExampleType">
  ...
</xsd:complexType>

<xsd:complexType name="FahrzeugType"
 abstract="true">
  ...
</xsd:complexType>

<xsd:complexType name="LKWType">
  ...
</xsd:complexType>

<xsd:complexType name="PKWType">
  ...
</xsd:complexType>

<xsd:complexType name="PersonType">
  ...
</xsd:complexType>
```

# Von UML zum XML-Schema

Regel:

- Vererbung --> extension

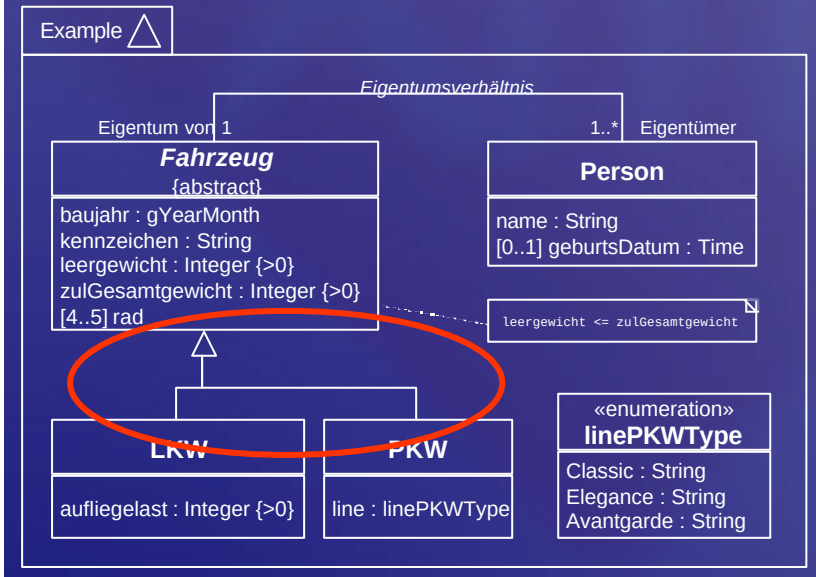



```
<xsd:complexType name="FahrzeugType">
  ...
</xsd:complexType>

<xsd:complexType name="LKWType">
   <xsd:complexContent>
    <xsd:extension base="FahrzeugType">
      ..
    </xsd:extension>
  </xsd:complexType>
</xsd:complexType>

<xsd:complexType name="PKWType">
 <xsd:complexContent>
    <xsd:extension base="FahrzeugType">
      ..
    </xsd:extension>
  </xsd:complexType>
</xsd:complexType>
```

# Von UML zum XML-Schema

Regel:

- Vererbung --> extension, substitutionGroup

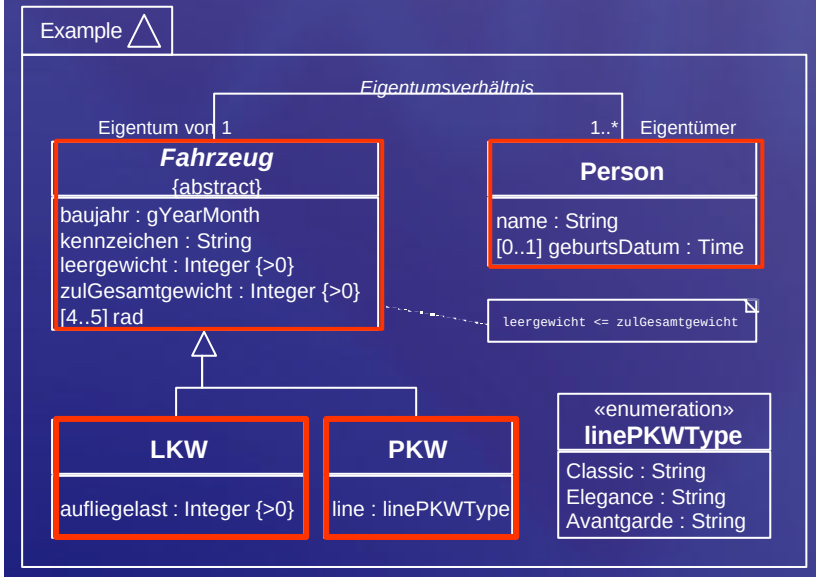



```
<xsd:element
 name="Example"
 type="ExampleType"/>
<xsd:element
 name="Fahrzeug"
 type="FahrzeugType"/>
<xsd:element
 name="PKW"
 type="PKWType"
 substitutionGroup="Fahrzeug"/>
<xsd:element
 name="LKW"
 type="LKWType"
 substitutionGroup="Fahrzeug"/>
```

# Von UML zum XML-Schema

Regel:

- Model --> complexType, element
- Klasse --> complexType, element

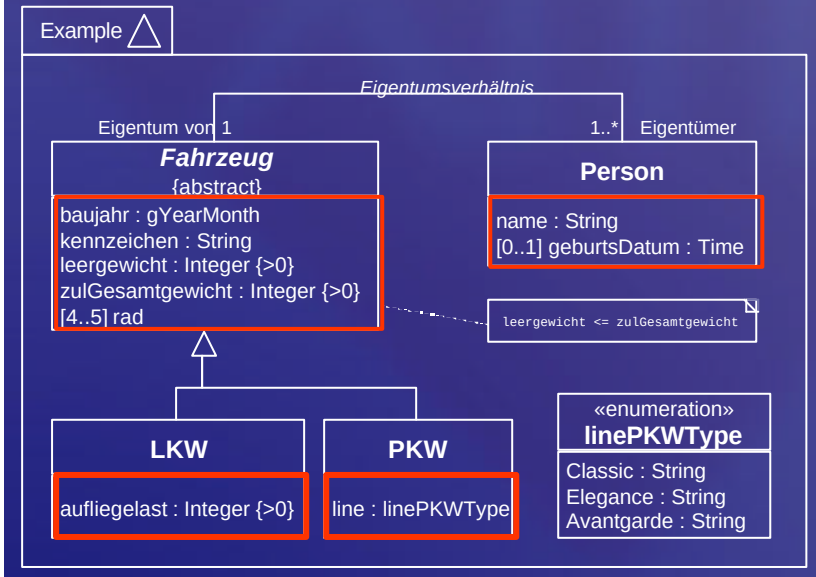



```
<xsd:complexType name="ExampleType">
  ...
<xsd:complexType name="FahrzeugType">
  ...
<xsd:complexType name="LKWType">
  ...
<xsd:complexType name="PKWType">
  ...
<xsd:complexType name="PersonType">
<xsd:element
 name="Example"
 type="ExampleType"/>
<xsd:element
 name="Fahrzeug"
 type="FahrzeugType"/>
<xsd:element
 name="PKW"
 type="PKWType" .../>
<xsd:element
 name="LKW"
 type="LKWType" .../>
```

# Von UML zum XML-Schema

Regel:

- Attribute --> element



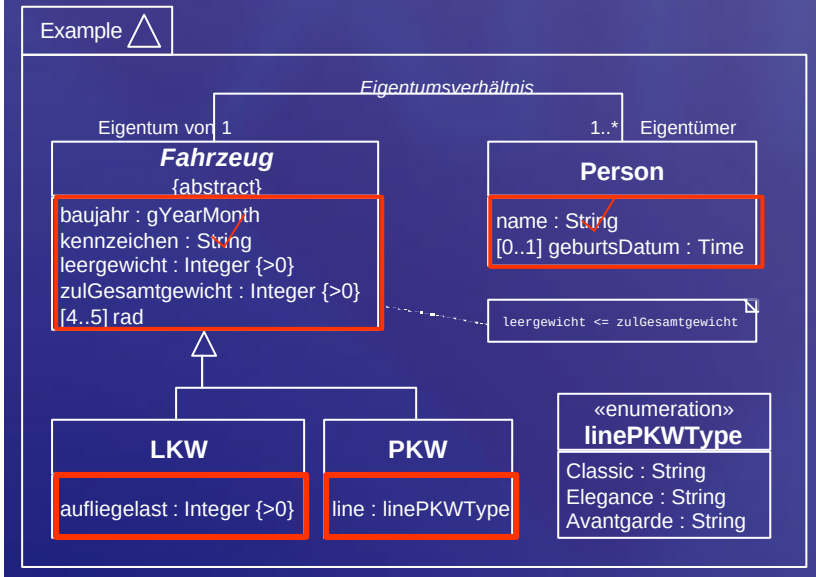

```
<xsd:element
 name="name"
 type="..."/>
<xsd:element
 name="geburtsDatum"
 type="..."/>
<xsd:element
 name="baujahr"
 type="..."/>
<xsd:element
 name="kennzeichen"
 type="..."/>
<xsd:element
 name="leergewicht"
 type="..."/>
...
```

# Von UML zum XML-Schema

Regel:

- Attribute --> element

  (I. vordefinierte UML-Datentypen)



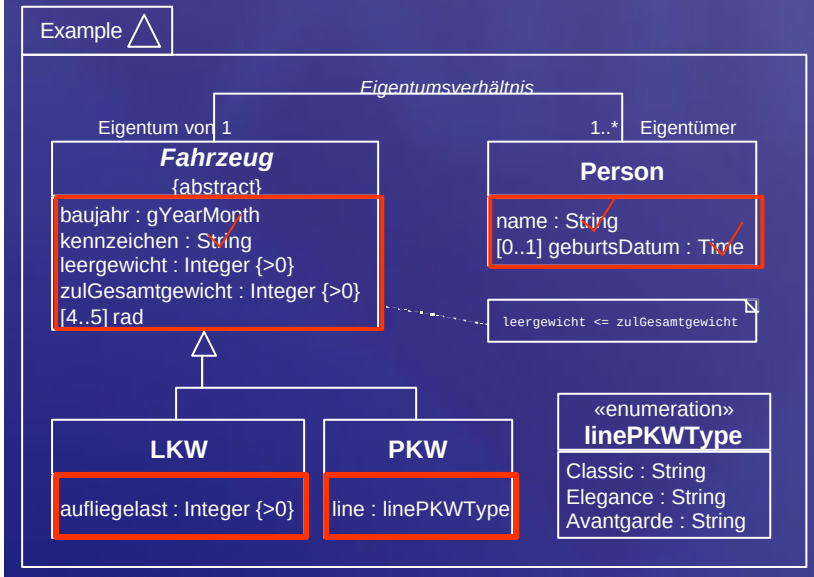

```
Integer      ──▶  integer
String       ──▶  string
Name         ──▶  Name
Boolean      ──▶  boolean
Time         ──▶
Uninterpreted ──▶ { base64Binary
                    hexBinary
```

# Von UML zum XML-Schema

Regel:

- Attribute --> element

  (I. vordefinierte UML-Datentypen)




Time ⟶

```
<xsd:simpleType name="UDtime">
  <xsd:union
    memberTypes="
      xsd:time
      xsd:date
      xsd:gMonth
      xsd:gYear"/>
</xsd:simpleType>
```

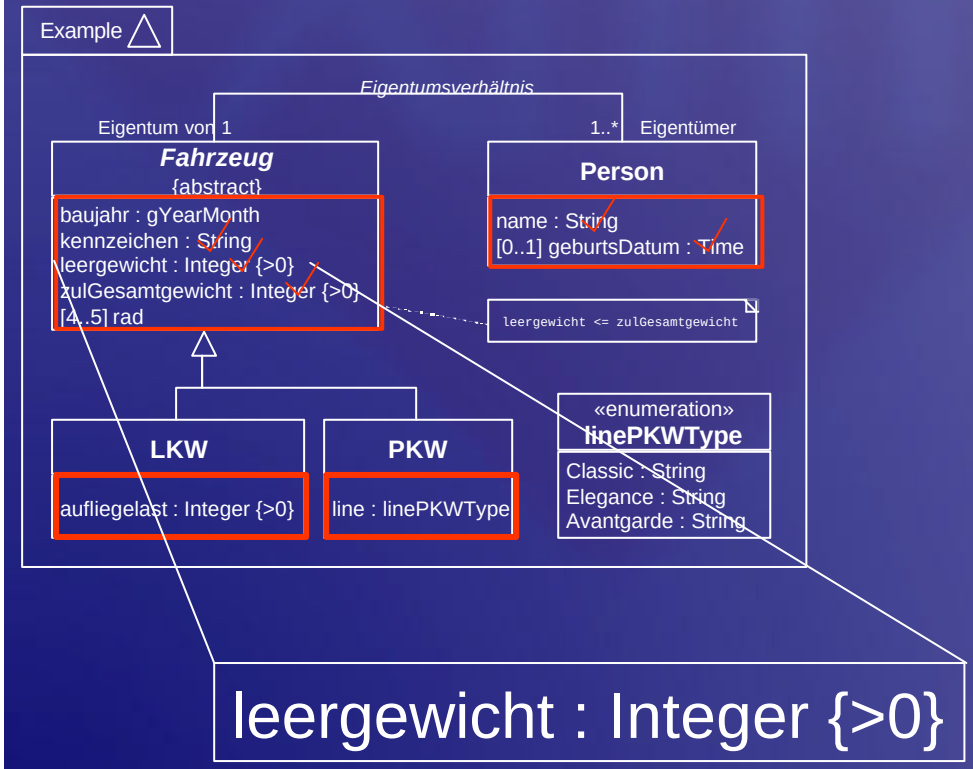
Uninterpreted ⟶ { base64Binary, hexBinary

# Von UML zum XML-Schema

Regel:

- Attribute --> element

  (I. modifizierte vordefinierte UML-Datentypen)



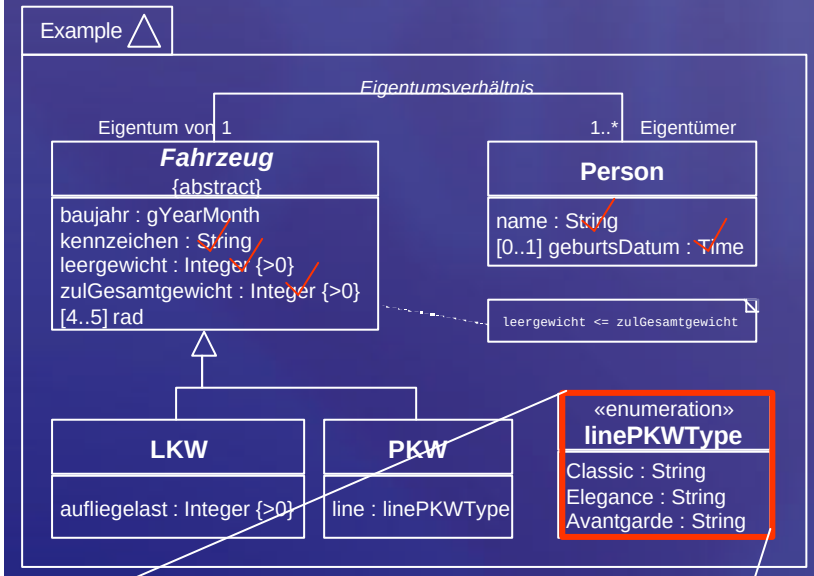

```
<xsd:element
 name="leergewicht"
 type="positiveInteger"/>
```

# Von UML zum XML-Schema

Regel:

- Attribute --> element

  (I. eigene UML-Datentypen (Aufzählungstypen))




«enumeration»
**linePKWType**

Classic : String
Elegance : String
Avantgarde : String

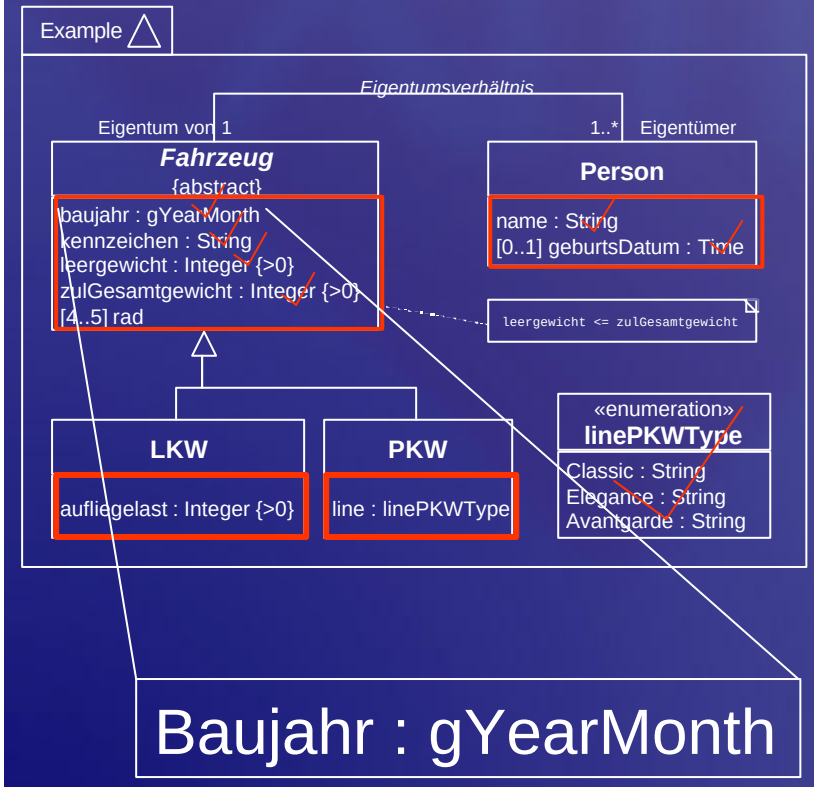
```
<xsd:simpleType name="linePKWType">
  <xsd:restriction base="xsd:string">
   <xsd:enumeration
    value="Classic"/>
   <xsd:enumeration
    value="Elegance"/>
   <xsd:enumeration
    value="Avantgarde"/>
 </xsd:restriction>
</xsd:simpleType>
<xsd:element
 name="line"
 type="linePKWType"/>
```

# Von UML zum XML-Schema

Regel:

- Attribute --> element

(II. vordefinierte XSD-Datentypen)



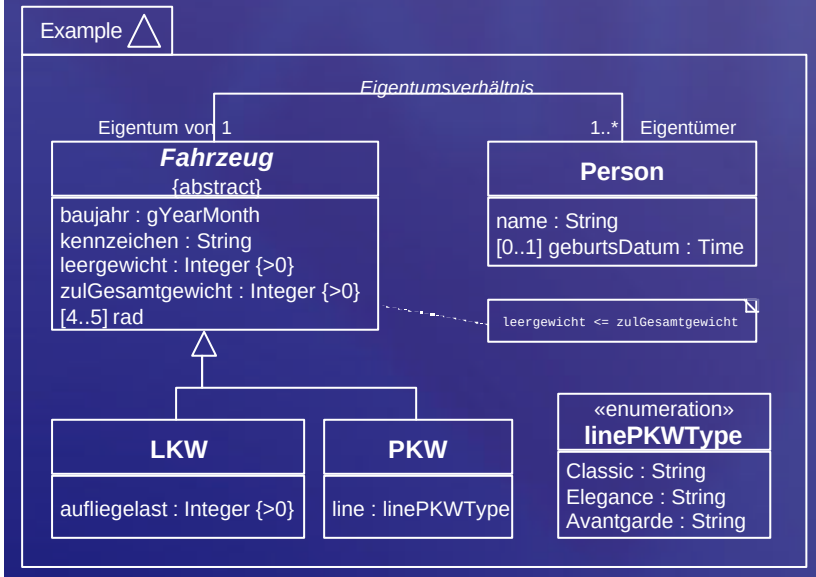

| | |
|---|---|
| normalizedString | integer |
| token | nonPositiveInteger |
| language | negativeInteger |
| IDREFS | long |
| ENTITIES | int |
| NMTOKEN | short |
| NMTOKENS | byte |
| Name | nonNegativeInteger |
| NCName | unsignedLong |
| ID | unsignedInt |
| IDREF | unsignedShort |
| ENTITY | unsignedByte |
| gYearMonth | positiveInteger |
| gYear | string |
| gMonthDay | boolean |
| gDay | decimal |
| gMonth | float |
| hexBinary | double |
| base64Binary | duration |
| anyURI | dateTime |
| QName | time |
| NOTATION | date |

# Von UML zum XML-Schema

Regel:

- Model --> element-Inhalt

  (*von Netzen und Bäumen ...*)



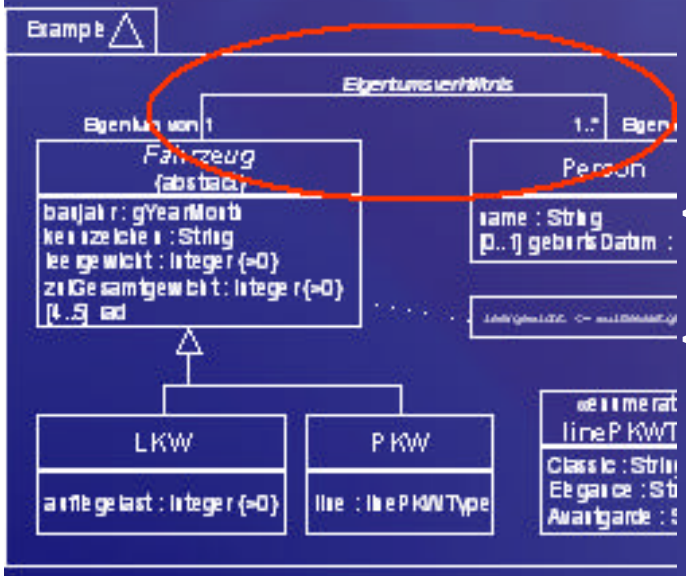

```
<xsd:complexType
 name="ExampleType"
 mixed="false">
  <xsd:choice>
    <xsd:element
     ref="Fahrzeug"/>
    <xsd:element
     ref="Person"/>
  </xsd:choice>
</xsd:complexType>
```

# Von UML zum XML-Schema

Regel:

- Assoziation --> element

  (*von Netzen und Bäumen ...*)

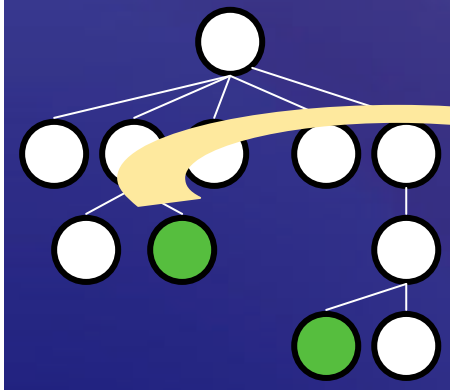



```
<xsd:complexType name="PersonType">
  ...
 <xsd:element
  ref="EigentümerVon.Fahrzeug"
  minOccurs="0"
  maxOccurs="unbounded"/>
</xsd:complexType>

<xsd:element name="EigentümerVon.Fahrzeug">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element
        ref="Fahrzeug"/>
    </xsd:sequence>
```

# Redundanz und ihre Kontrolle

Grundidee:

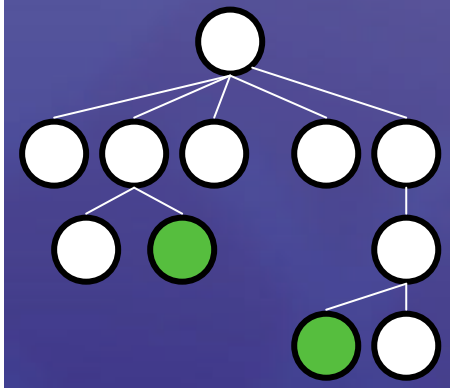
- Alle Charakteristika (Attribute und Kindelemente) eines Elements werden als optional deklariert
- Jedes Element erhält zusätzlich ein identifizierendes Attribut, Referenzierung durch *simple XLink*



```
<ElementB>
    <ElementB.has>
        <ElementA xmiid="ElementA1">
        ...
        </ElementA>
    </ElementB.has>
</ElementB>
```

```
<ElementC>
    <ElementC.has>
        <ElementA xlink:type="simple"
                  xlink:href="ElementA1" .../>
```

# Redundanz und ihre Kontrolle



```
<xsd:complexType name="foo">
  ...
 <xsd:attribute name="xmiid" type="ID">
 <xsd:attribute name="xlink:type" type="xsd:string" …/>
 <xsd:attribute name="xlink:href" type="xsd:string" …/>
 <xsd:attribute name="xlink:role" type="xsd:string" …/>
 <xsd:attribute name="xlink:arcrole" type="xsd:string" …/>
 <xsd:attribute name="xlink:title" type="xsd:string" …/>
 <xsd:attribute name="xlink:show"
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
           <xsd:enumeration value="embed"/>
           <xsd:enumeration value="replace"/>
           <xsd:enumeration value="new"/>
         </xsd:restriction>
       </xsd:simpleType>
 </xsd:attribute>
 <xsd:attribute name="xlink:actuate">
   <xsd:simpleType>
     <xsd:restriction base="xsd:string">
       <xsd:enumeration value="show"/>
         <xsd:enumeration value="user"/>
     </xsd:restriction>
   </xsd:simpleType>
 </xsd:attribute>
…
```

# 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/`

**OMG's XML Metadata Interchange:**

- `http://www.omg.org/XML`

**DTD- und Schemagenerierung:**

- `http://www.jeckle.de`