XSL - Extensible Stylesheet Language

XML-Daten transformieren und präsentieren

Michael Dienert

Walther-Rathenau-Gewerbeschule
Freiburg

11. November 2014

Inhalt

XML Stylesheet Language

- XSL ist die Abkürzung von *Extensible Stylesheet Language*.
- XSL ist selbst wieder eine XML-Anwendung, d.h. XSL-Dokumente müssen immer wohlgeformte XML-Dokumente sein.
- XSL besteht aus drei Teilen:

$$\text{XSL} = \text{XPath} + \text{XSLT} + \text{XSL-FO}$$

XML Stylesheet Language

- XSL ist die Abkürzung von *Extensible Stylesheet Language*.
- XSL ist selbst wieder eine XML-Anwendung, d.h. XSL-Dokumente müssen immer wohlgeformte XML-Dokumente sein.
- XSL besteht aus drei Teilen:

$$\text{XSL} = \text{XPath} + \text{XSLT} + \text{XSL-FO}$$

XML Stylesheet Language

- XSL ist die Abkürzung von *Extensible Stylesheet Language*.
- XSL ist selbst wieder eine XML-Anwendung, d.h. XSL-Dokumente müssen immer wohlgeformte XML-Dokumente sein.
- XSL besteht aus drei Teilen:

$$\text{XSL} = \text{XPath} + \text{XSLT} + \text{XSL-FO}$$

XML Stylesheet Language

- XSL ist die Abkürzung von *Extensible Stylesheet Language*.
- XSL ist selbst wieder eine XML-Anwendung, d.h. XSL-Dokumente müssen immer wohlgeformte XML-Dokumente sein.
- XSL besteht aus drei Teilen:

$$\text{XSL} = \text{XPath} + \text{XSLT} + \text{XSL-FO}$$

XML Stylesheet Language

- XSL ist die Abkürzung von *Extensible Stylesheet Language*.
- XSL ist selbst wieder eine XML-Anwendung, d.h. XSL-Dokumente müssen immer wohlgeformte XML-Dokumente sein.
- XSL besteht aus drei Teilen:

$$\text{XSL} = \text{XPath} + \text{XSLT} + \text{XSL-FO}$$

XML Stylesheet Language

- XSL ist die Abkürzung von *Extensible Stylesheet Language*.
- XSL ist selbst wieder eine XML-Anwendung, d.h. XSL-Dokumente müssen immer wohlgeformte XML-Dokumente sein.
- XSL besteht aus drei Teilen:

$$\text{XSL} = \text{XPath} + \text{XSLT} + \text{XSL-FO}$$

XML Stylesheet Language

- XSL ist die Abkürzung von *Extensible Stylesheet Language*.
- XSL ist selbst wieder eine XML-Anwendung, d.h. XSL-Dokumente müssen immer wohlgeformte XML-Dokumente sein.
- XSL besteht aus drei Teilen:

$$\text{XSL} = \text{XPath} + \text{XSLT} + \text{XSL-FO}$$

XML Stylesheet Language

- XSL ist die Abkürzung von *Extensible Stylesheet Language*.
- XSL ist selbst wieder eine XML-Anwendung, d.h. XSL-Dokumente müssen immer wohlgeformte XML-Dokumente sein.
- XSL besteht aus drei Teilen:

$$\text{XSL} = \text{XPath} + \text{XSLT} + \text{XSL-FO}$$

XML Stylesheet Language

- XSL ist die Abkürzung von *Extensible Stylesheet Language*.
- XSL ist selbst wieder eine XML-Anwendung, d.h. XSL-Dokumente müssen immer wohlgeformte XML-Dokumente sein.
- XSL besteht aus drei Teilen:

$$\text{XSL} = \text{XPath} + \text{XSLT} + \text{XSL-FO}$$

XML Stylesheet Language

- XSL ist die Abkürzung von *Extensible Stylesheet Language*.
- XSL ist selbst wieder eine XML-Anwendung, d.h. XSL-Dokumente müssen immer wohlgeformte XML-Dokumente sein.
- XSL besteht aus drei Teilen:

$$\text{XSL} = \text{XPath} + \text{XSLT} + \text{XSL-FO}$$

Die drei Bestandteile von XSL

Im Detail:

XPath: um Teile eines XML-Dokuments auswählen zu könnnen

XSLT: **XSL-T**ransformations, um ein XML-Dokument in ein anderes (XML)-Dokument transformieren zu könnnen

Hauptanwendung: Transformation von xml (Daten) nach html (Darstellung)

XSL-FO: **XSL-F**ormatting **O**bjects, um ein XML-Dokument in eine Präsentationsform wie z.B. pdf oder PostScript umwandeln zu können.

Die drei Bestandteile von XSL

Im Detail:

XPath: um Teile eines XML-Dokuments auswählen zu könnnen

XSLT: **XSL-T**ransformations, um ein XML-Dokument in ein anderes (XML)-Dokument transformieren zu könnnen

Hauptanwendung: Transformation von xml (Daten) nach html (Darstellung)

XSL-FO: **XSL-F**ormatting **O**bjects, um ein XML-Dokument in eine Präsentationsform wie z.B. pdf oder PostScript umwandeln zu können.

Die drei Bestandteile von XSL

Im Detail:

XPath: um Teile eines XML-Dokuments auswählen zu könnnen

XSLT: **XSL-T**ransformations, um ein XML-Dokument in ein anderes (XML)-Dokument transformieren zu könnnen

Hauptanwendung: Transformation von xml (Daten) nach html (Darstellung)

XSL-FO: **XSL-F**ormatting **O**bjects, um ein XML-Dokument in eine Präsentationsform wie z.B. pdf oder PostScript umwandeln zu können.

Die drei Bestandteile von XSL

Im Detail:

XPath: um Teile eines XML-Dokuments auswählen zu könnnen

XSLT: **XSL-T**ransformations, um ein XML-Dokument in ein anderes (XML)-Dokument transformieren zu könnnen

Hauptanwendung: Transformation von xml (Daten) nach html (Darstellung)

XSL-FO: **XSL-F**ormatting **O**bjects, um ein XML-Dokument in eine Präsentationsform wie z.B. pdf oder PostScript umwandeln zu können.

Die drei Bestandteile von XSL

Im Detail:

XPath: um Teile eines XML-Dokuments auswählen zu könnnen

XSLT: **XSL-T**ransformations, um ein XML-Dokument in ein anderes (XML)-Dokument transformieren zu könnnen

Hauptanwendung: Transformation von xml (Daten) nach html (Darstellung)

XSL-FO: **XSL-F**ormatting **O**bjects, um ein XML-Dokument in eine Präsentationsform wie z.B. pdf oder PostScript umwandeln zu können.

Die drei Bestandteile von XSL

Im Detail:

XPath: um Teile eines XML-Dokuments auswählen zu könnnen

XSLT: **XSL-T**ransformations, um ein XML-Dokument in ein anderes (XML)-Dokument transformieren zu könnnen

Hauptanwendung: Transformation von xml (Daten) nach html (Darstellung)

XSL-FO: **XSL-F**ormatting **O**bjects, um ein XML-Dokument in eine Präsentationsform wie z.B. pdf oder PostScript umwandeln zu können.

Die drei Bestandteile von XSL

Im Detail:

XPath: um Teile eines XML-Dokuments auswählen zu könnnen

XSLT: **XSL-T**ransformations, um ein XML-Dokument in ein anderes (XML)-Dokument transformieren zu könnnen

Hauptanwendung: Transformation von xml (Daten) nach html (Darstellung)

XSL-FO: **XSL-F**ormatting **O**bjects, um ein XML-Dokument in eine Präsentationsform wie z.B. pdf oder PostScript umwandeln zu können.

Die drei Bestandteile von XSL

Im Detail:

XPath: um Teile eines XML-Dokuments auswählen zu könnnen

XSLT: **XSL-T**ransformations, um ein XML-Dokument in ein anderes (XML)-Dokument transformieren zu könnnen

Hauptanwendung: Transformation von xml (Daten) nach html (Darstellung)

XSL-FO: **XSL-F**ormatting **O**bjects, um ein XML-Dokument in eine Präsentationsform wie z.B. pdf oder PostScript umwandeln zu können.

Die drei Bestandteile von XSL

Im Detail:

XPath: um Teile eines XML-Dokuments auswählen zu könnnen

XSLT: **XSL-T**ransformations, um ein XML-Dokument in ein anderes (XML)-Dokument transformieren zu könnnen

Hauptanwendung: Transformation von xml (Daten) nach html (Darstellung)

XSL-FO: **XSL-F**ormatting **O**bjects, um ein XML-Dokument in eine Präsentationsform wie z.B. pdf oder PostScript umwandeln zu können.

Die drei Bestandteile von XSL

Im Detail:

XPath: um Teile eines XML-Dokuments auswählen zu könnnen

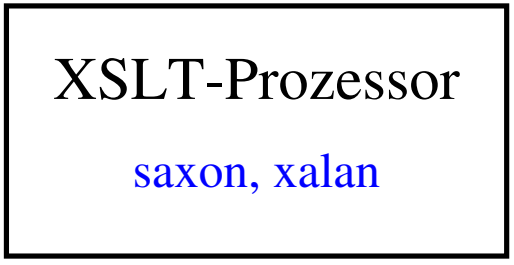
XSLT: **XSL-T**ransformations, um ein XML-Dokument in ein anderes (XML)-Dokument transformieren zu könnnen

Hauptanwendung: Transformation von xml (Daten) nach html (Darstellung)

XSL-FO: **XSL-F**ormatting **O**bjects, um ein XML-Dokument in eine Präsentationsform wie z.B. pdf oder PostScript umwandeln zu können.

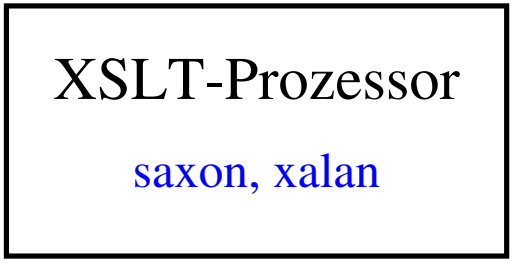
Der Transformationsvorgang

Der Transformationsvorgang



Der Transformationsvorgang

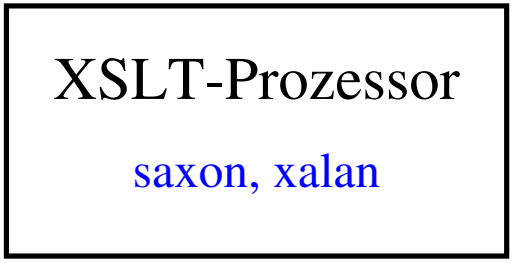
XSLT-Stylesheet



Der Transformationsvorgang

XSLT-Stylesheet

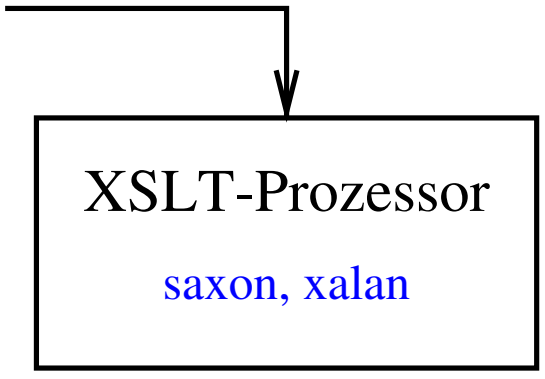
spielplan.xsl



Der Transformationsvorgang

XSLT-Stylesheet

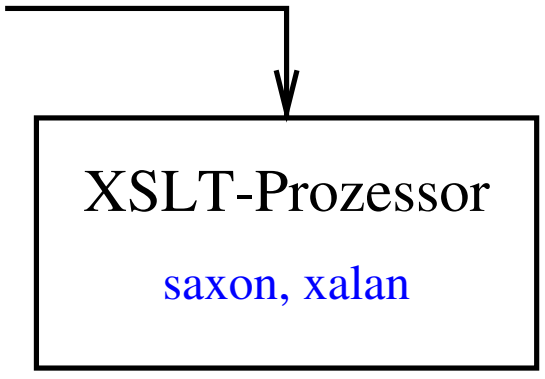
spielplan.xsl



Der Transformationsvorgang

XSLT-Stylesheet

spielplan.xsl

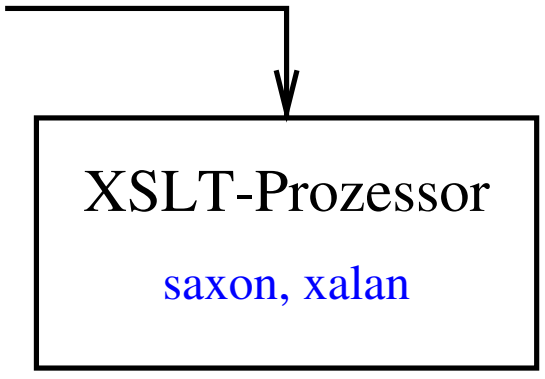


XML-Datei

Der Transformationsvorgang

XSLT-Stylesheet

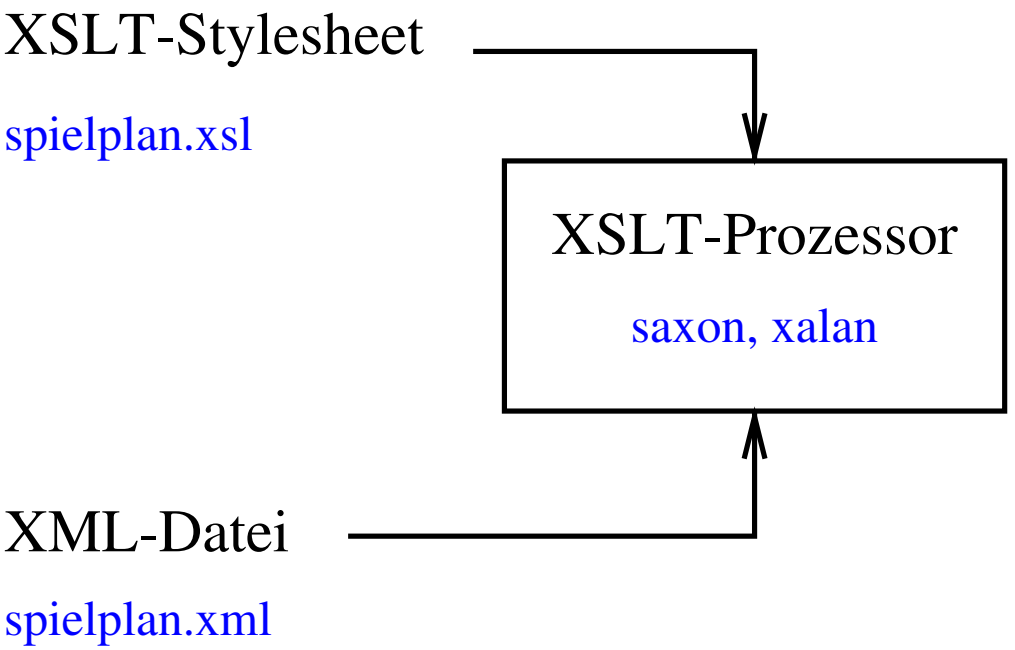
spielplan.xsl



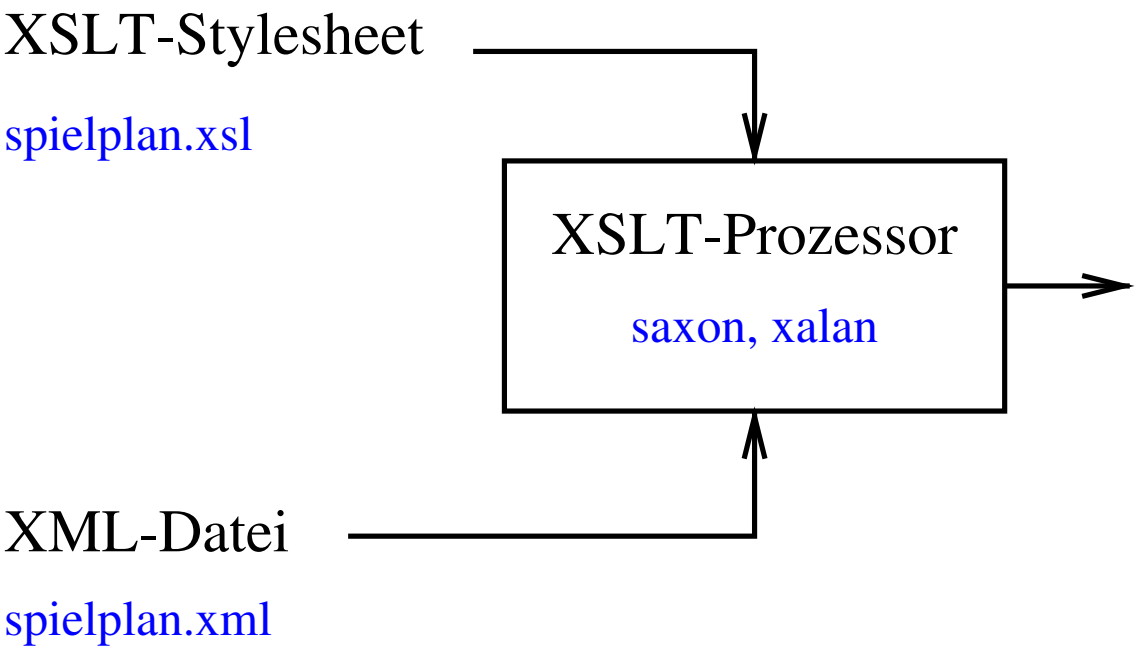
XML-Datei

spielplan.xml

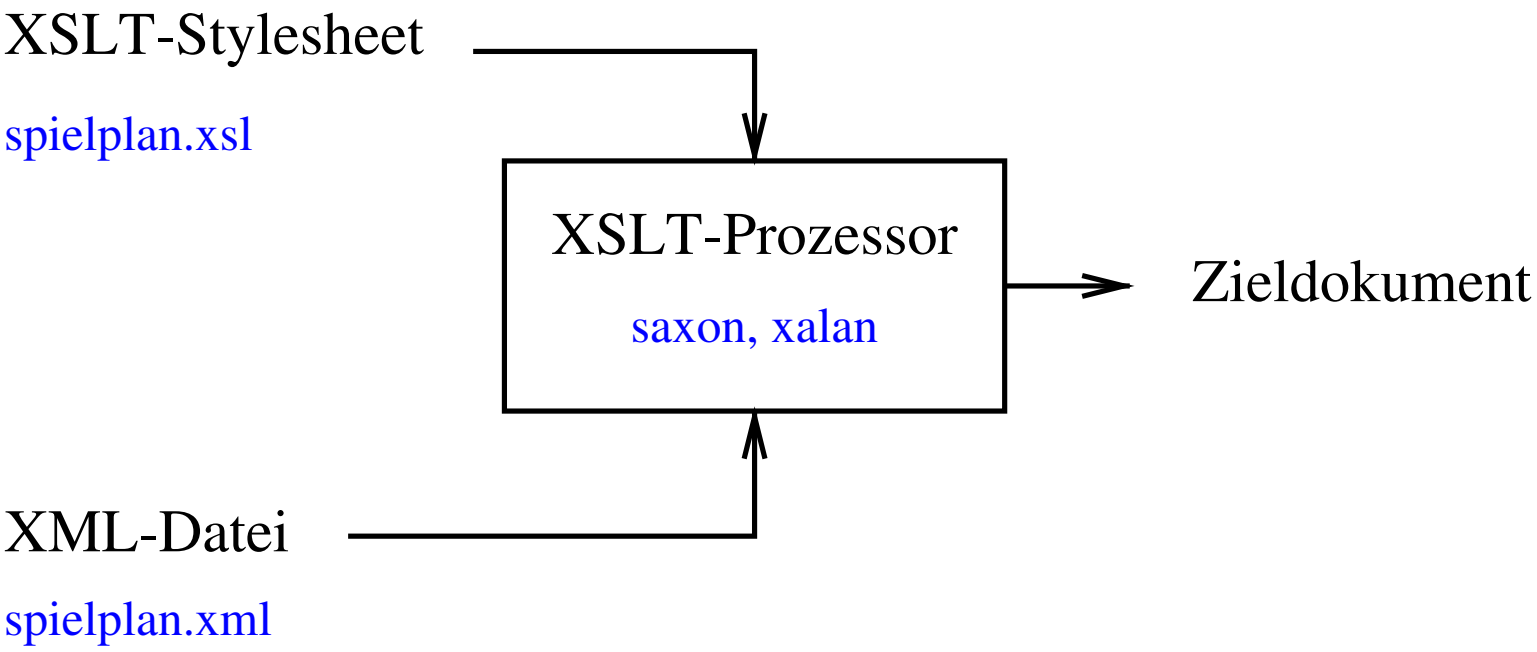
Der Transformationsvorgang



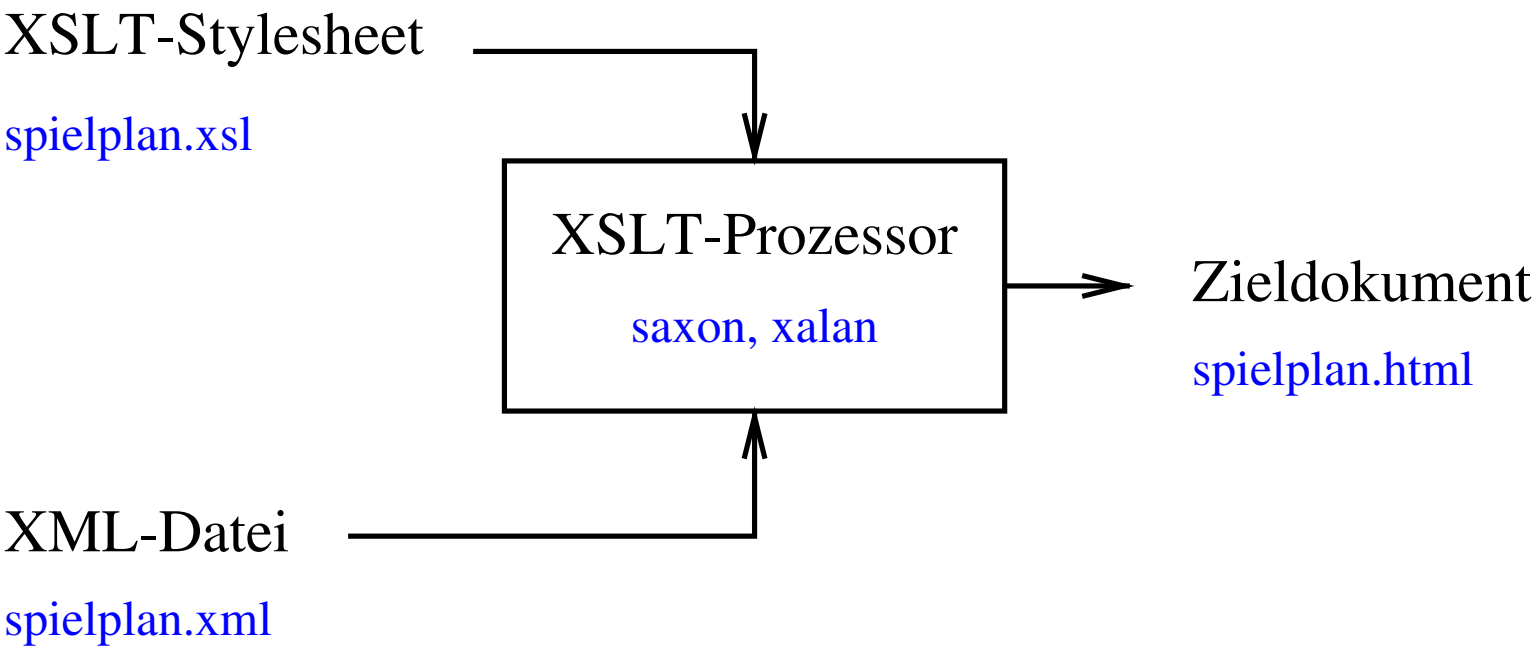
Der Transformationsvorgang



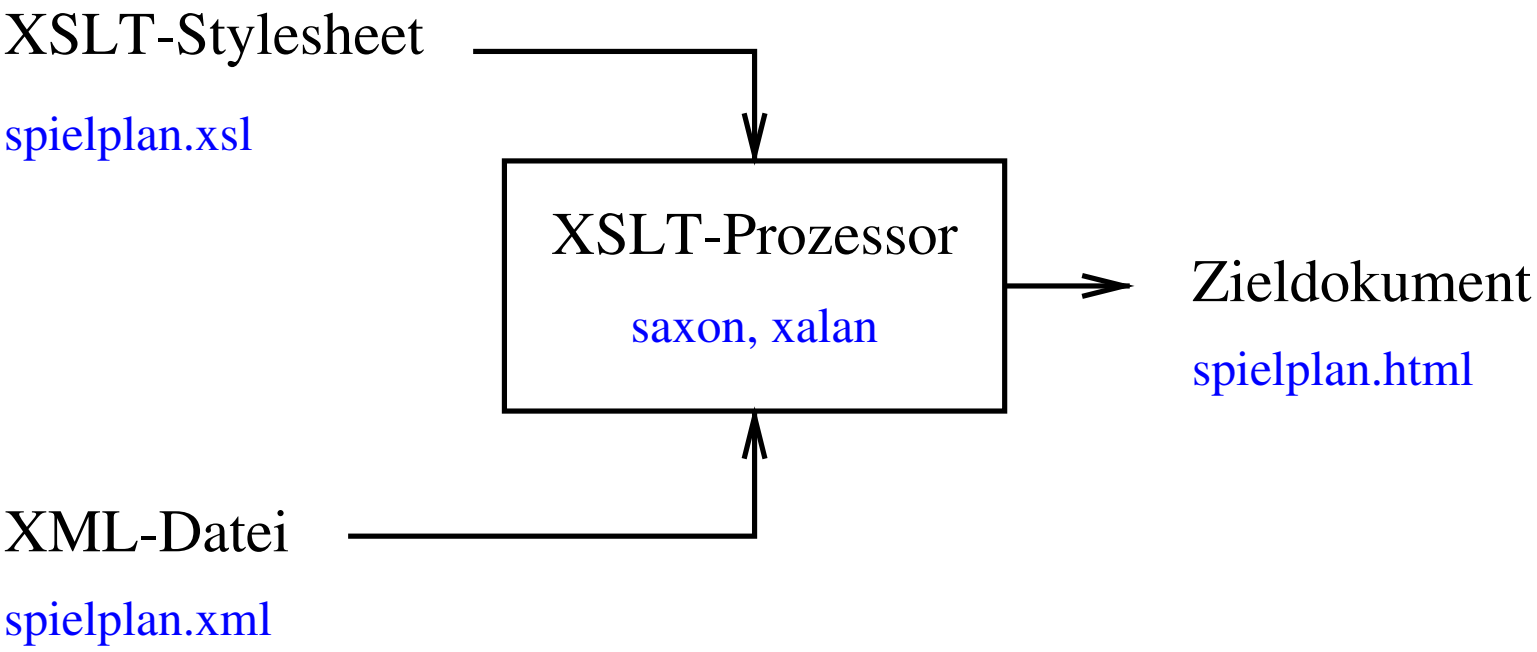
Der Transformationsvorgang



Der Transformationsvorgang

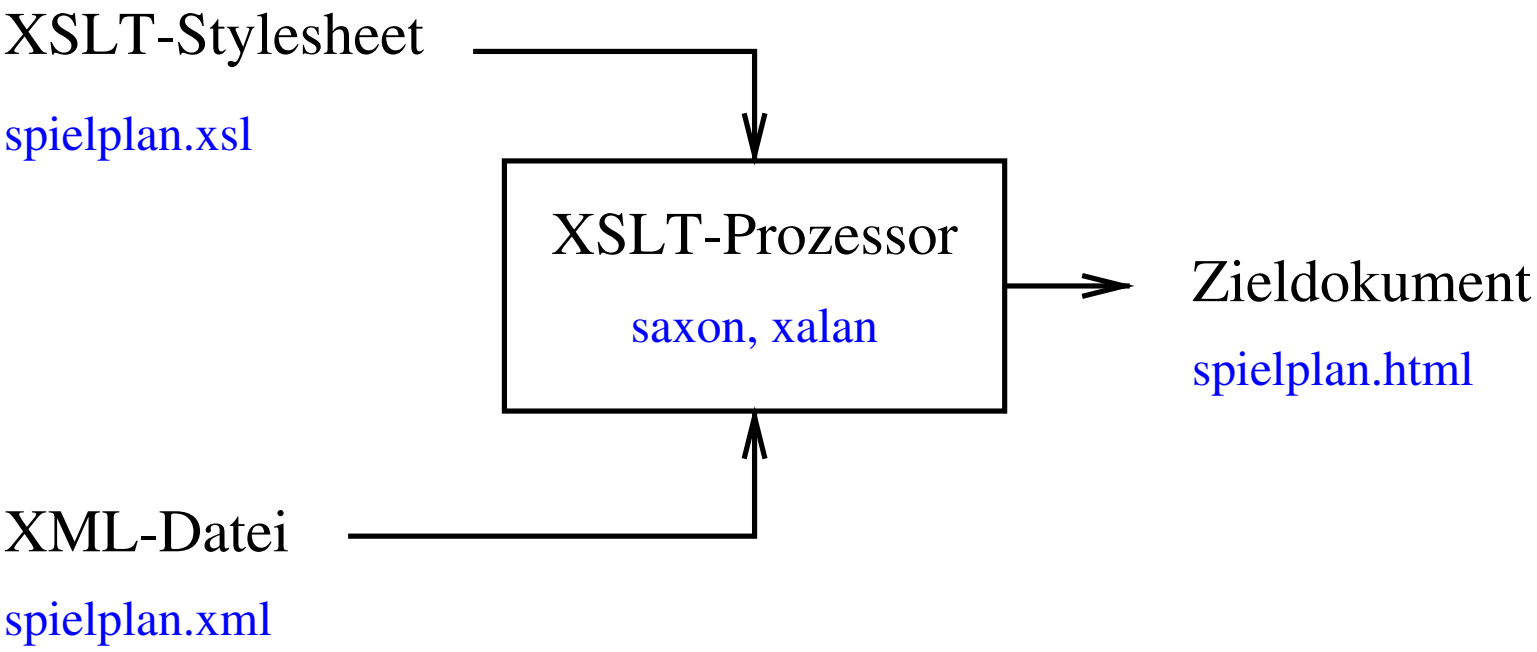


Der Transformationsvorgang



- Aufruf von saxon:

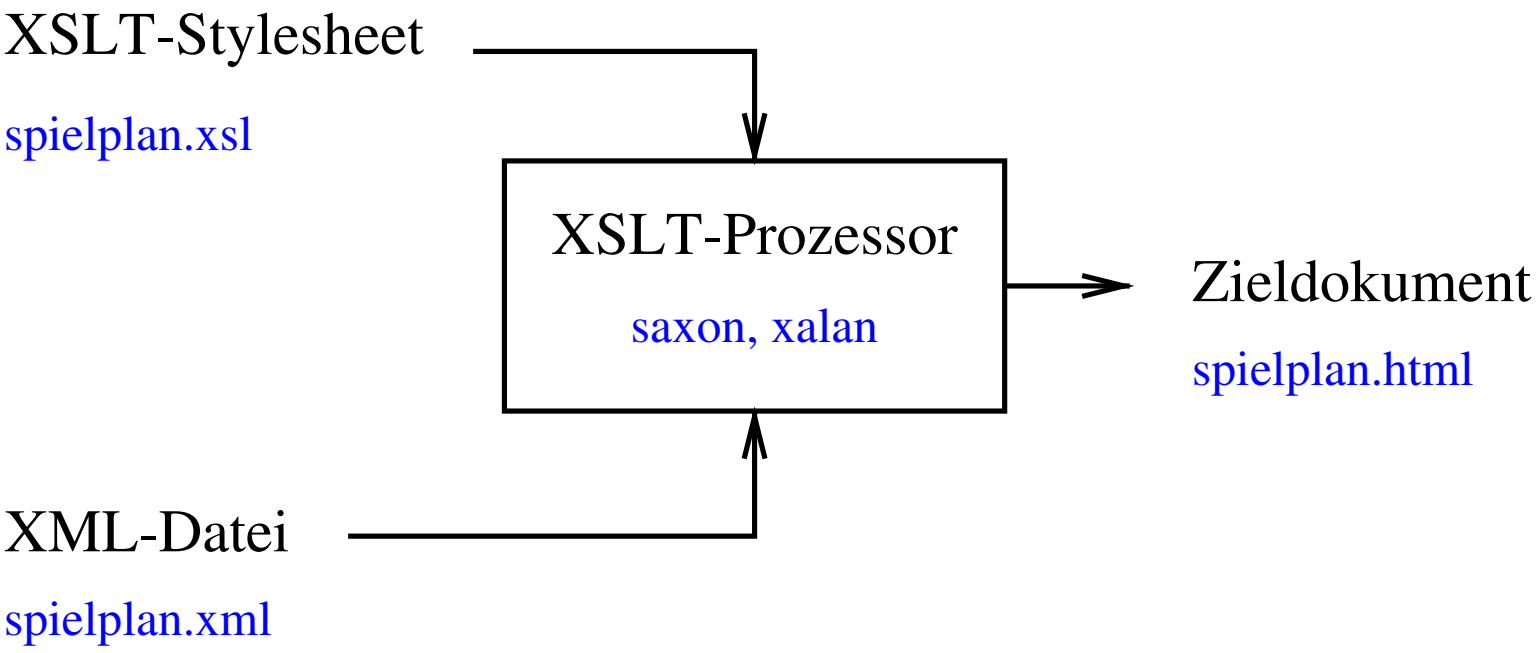
Der Transformationsvorgang



- Aufruf von saxon:

```
java -jar saxon.jar spielplan.xml spielplan.xsl
```

Der Transformationsvorgang

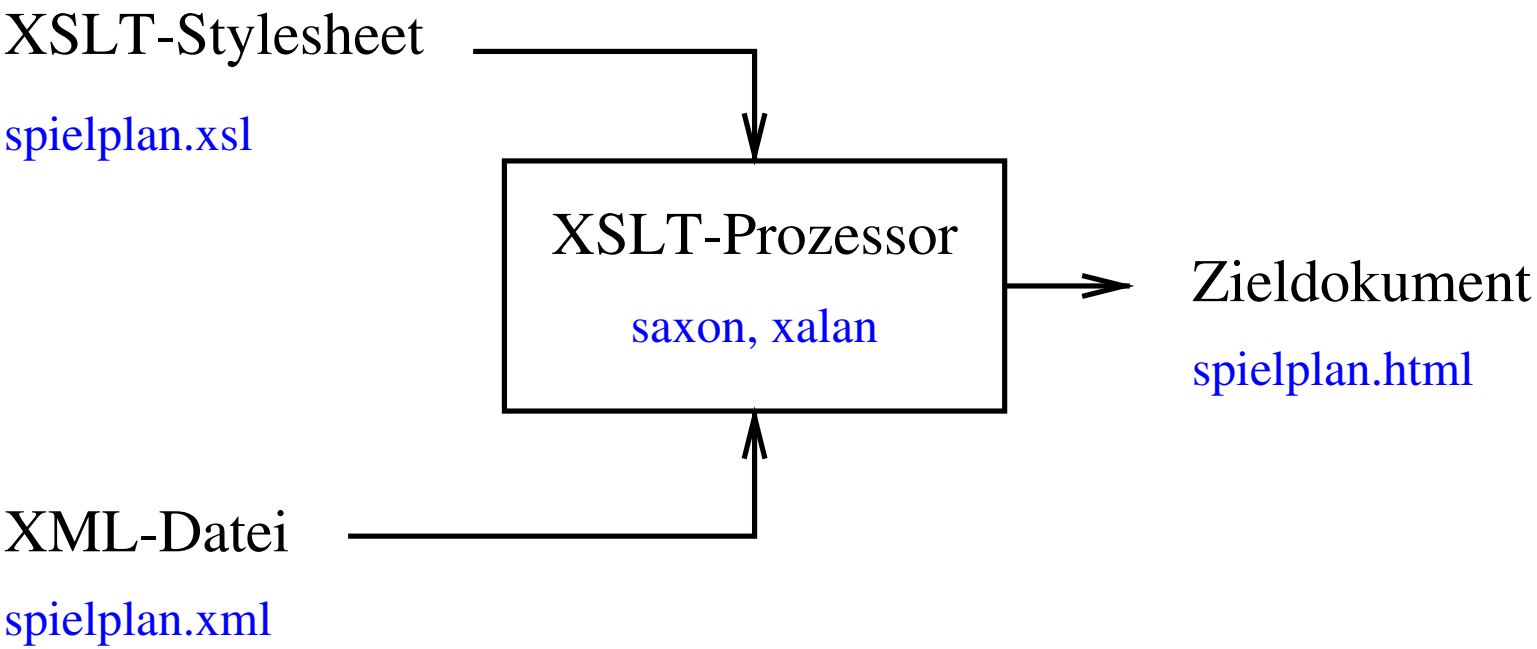


- Aufruf von saxon:

```
java -jar saxon.jar spielplan.xml spielplan.xsl
```

- Aufruf von xalan (C++ - Version):

Der Transformationsvorgang



- Aufruf von saxon:

```
java -jar saxon.jar spielplan.xml spielplan.xsl
```

- Aufruf von xalan (C++ - Version):

```
xalan -in europe.xml -xsl europe.xsl -out test.dxf
```

Zusammenspiel der drei Komponenten

Zusammenspiel von XPath, XSLT und XSL-FO:

XPath: wird hauptsächlich **innerhalb** von XSLT-Dokumenten eingesetzt.

XSL-FO: XSL-FO-Dokumente werden von einem XSL-FO-Formatierer in z.b. pdf umgewandelt. Beispiel für ein solches Programm:

Apache Formatting Objects Processor **fop**

XSLT: XSL-FO-Dokumente werden nicht von Hand geschrieben, sondern mit XSLT aus einem XML-Dokument erzeugt.

konsequente Trennung von Daten (xml) und Darstellung (xsl)

Zusammenspiel der drei Komponenten

Zusammenspiel von XPath, XSLT und XSL-FO:

XPath: wird hauptsächlich **innerhalb** von XSLT-Dokumenten eingesetzt.

XSL-FO: XSL-FO-Dokumente werden von einem XSL-FO-Formatierer in z.b. pdf umgewandelt. Beispiel für ein solches Programm:

Apache Formatting Objects Processor **fop**

XSLT: XSL-FO-Dokumente werden nicht von Hand geschrieben, sondern mit XSLT aus einem XML-Dokument erzeugt.

konsequente Trennung von Daten (xml) und Darstellung (xsl)

Zusammenspiel der drei Komponenten

Zusammenspiel von XPath, XSLT und XSL-FO:

XPath: wird hauptsächlich **innerhalb** von XSLT-Dokumenten eingesetzt.

XSL-FO: XSL-FO-Dokumente werden von einem XSL-FO-Formatierer in z.b. pdf umgewandelt.
Beispiel für ein solches Programm:

Apache Formatting Objects Processor **fop**

XSLT: XSL-FO-Dokumente werden nicht von Hand geschrieben, sondern mit XSLT aus einem XML-Dokument erzeugt.

konsequente Trennung von Daten (xml) und Darstellung (xsl)

Zusammenspiel der drei Komponenten

Zusammenspiel von XPath, XSLT und XSL-FO:

XPath: wird hauptsächlich **innerhalb** von XSLT-Dokumenten eingesetzt.

XSL-FO: XSL-FO-Dokumente werden von einem XSL-FO-Formatierer in z.b. pdf umgewandelt. Beispiel für ein solches Programm:

Apache Formatting Objects Processor **fop**

XSLT: XSL-FO-Dokumente werden nicht von Hand geschrieben, sondern mit XSLT aus einem XML-Dokument erzeugt.

konsequente Trennung von Daten (xml) und Darstellung (xsl)

Zusammenspiel der drei Komponenten

Zusammenspiel von XPath, XSLT und XSL-FO:

XPath: wird hauptsächlich **innerhalb** von XSLT-Dokumenten eingesetzt.

XSL-FO: XSL-FO-Dokumente werden von einem XSL-FO-Formatierer in z.b. pdf umgewandelt. Beispiel für ein solches Programm:

Apache Formatting Objects Processor **fop**

XSLT: XSL-FO-Dokumente werden nicht von Hand geschrieben, sondern mit XSLT aus einem XML-Dokument erzeugt.

konsequente Trennung von Daten (xml) und Darstellung (xsl)

Zusammenspiel der drei Komponenten

Zusammenspiel von XPath, XSLT und XSL-FO:

XPath: wird hauptsächlich **innerhalb** von XSLT-Dokumenten eingesetzt.

XSL-FO: XSL-FO-Dokumente werden von einem XSL-FO-Formatierer in z.b. pdf umgewandelt. Beispiel für ein solches Programm:

Apache Formatting Objects Processor **fop**

XSLT: XSL-FO-Dokumente werden nicht von Hand geschrieben, sondern mit XSLT aus einem XML-Dokument erzeugt.

konsequente Trennung von Daten (xml) und Darstellung (xsl)

Zusammenspiel der drei Komponenten

Zusammenspiel von XPath, XSLT und XSL-FO:

XPath: wird hauptsächlich **innerhalb** von XSLT-Dokumenten eingesetzt.

XSL-FO: XSL-FO-Dokumente werden von einem XSL-FO-Formatierer in z.b. pdf umgewandelt. Beispiel für ein solches Programm:

Apache Formatting Objects Processor **fop**

XSLT: XSL-FO-Dokumente werden nicht von Hand geschrieben, sondern mit XSLT aus einem XML-Dokument erzeugt.

konsequente Trennung von Daten (xml) und Darstellung (xsl)

Anwendungsbeispiel: **Ganz alte** Dateiformate erzeugen

- XSLT erzeugt **beliebige** Zieldokumente
- Zum Beispiel: *Data Interchange Format*: quelloffenes CAD-Austauschformat der Firma *Autodesk*
- Gegeben sind eine Reihe von x-y-Koordinaten (Spantenriss)
- Daten liegen bereits in einem xml-Format vor
- Aufgabe: die Daten in ein CAD-Format bringen
- Der Einfachheit halber Version 12 gewählt: 1988 !

Anwendungsbeispiel: **Ganz alte** Dateiformate erzeugen

- XSLT erzeugt **beliebige** Zieldokumente
- Zum Beispiel: *Data Interchange Format*: quelloffenes CAD-Austauschformat der Firma *Autodesk*
- Gegeben sind eine Reihe von x-y-Koordinaten (Spantenriss)
- Daten liegen bereits in einem xml-Format vor
- Aufgabe: die Daten in ein CAD-Format bringen
- Der Einfachheit halber Version 12 gewählt: 1988 !

Anwendungsbeispiel: **Ganz alte** Dateiformate erzeugen

- XSLT erzeugt **beliebige** Zieldokumente
- Zum Beispiel: *Data Interchange Format*: quelloffenes CAD-Austauschformat der Firma *Autodesk*
- Gegeben sind eine Reihe von x-y-Koordinaten (Spantenriss)
- Daten liegen bereits in einem xml-Format vor
- Aufgabe: die Daten in ein CAD-Format bringen
- Der Einfachheit halber Version 12 gewählt: 1988 !

Anwendungsbeispiel: **Ganz alte** Dateiformate erzeugen

- XSLT erzeugt **beliebige** Zieldokumente
- Zum Beispiel: *Data Interchange Format*: quelloffenes CAD-Austauschformat der Firma *Autodesk*
- Gegeben sind eine Reihe von x-y-Koordinaten (Spantenriss)
- Daten liegen bereits in einem xml-Format vor
- Aufgabe: die Daten in ein CAD-Format bringen
- Der Einfachheit halber Version 12 gewählt: 1988 !

Anwendungsbeispiel: **Ganz alte** Dateiformate erzeugen

- XSLT erzeugt **beliebige** Zieldokumente
- Zum Beispiel: *Data Interchange Format*: quelloffenes CAD-Austauschformat der Firma *Autodesk*
- Gegeben sind eine Reihe von x-y-Koordinaten (Spantenriss)
- Daten liegen bereits in einem xml-Format vor
- Aufgabe: die Daten in ein CAD-Format bringen
- Der Einfachheit halber Version 12 gewählt: 1988 !

Anwendungsbeispiel: **Ganz alte** Dateiformate erzeugen

- XSLT erzeugt **beliebige** Zieldokumente
- Zum Beispiel: *Data Interchange Format*: quelloffenes CAD-Austauschformat der Firma *Autodesk*
- Gegeben sind eine Reihe von x-y-Koordinaten (Spantenriss)
- Daten liegen bereits in einem xml-Format vor
- Aufgabe: die Daten in ein CAD-Format bringen
- Der Einfachheit halber Version 12 gewählt: 1988 !

Anwendungsbeispiel: **Ganz alte** Dateiformate erzeugen

- XSLT erzeugt **beliebige** Zieldokumente
- Zum Beispiel: *Data Interchange Format*: quelloffenes CAD-Austauschformat der Firma *Autodesk*
- Gegeben sind eine Reihe von x-y-Koordinaten (Spantenriss)
- Daten liegen bereits in einem xml-Format vor
- Aufgabe: die Daten in ein CAD-Format bringen
- Der Einfachheit halber Version 12 gewählt: 1988 !

Anwendungsbeispiel: **Ganz alte** Dateiformate erzeugen

- XSLT erzeugt **beliebige** Zieldokumente
- Zum Beispiel: *Data Interchange Format*: quelloffenes CAD-Austauschformat der Firma *Autodesk*
- Gegeben sind eine Reihe von x-y-Koordinaten (Spantenriss)
- Daten liegen bereits in einem xml-Format vor
- Aufgabe: die Daten in ein CAD-Format bringen
- Der Einfachheit halber Version 12 gewählt: 1988 !

Transformatin von xy-Kooordinaten

Rohdaten im xml-Format

```
<spant nr="1">
   <vertex>
      <x>81.00000000</x>
      <y>5.50000000</y>
   </vertex>
   <vertex>
      <x>80.26183187</x>
      <y>14.15232648</y>
   </vertex>
   <vertex>
      <x>79.40402646</x>
      <y>28.90070211</y>
   </vertex>
    ...
</spant>
```

Zeichnungsdaten im dxf-Format

```
    ...
0
POLYLINE
8
0
70
4
0
VERTEX
10
81.00000000
20
5.50000000
0
VERTEX
10
80.26183187
20
14.15232648
    ...
```

Transformatin von xy-Kooordinaten

Rohdaten im xml-Format

```
<spant nr="1">
   <vertex>
      <x>81.00000000</x>
      <y>5.50000000</y>
   </vertex>
   <vertex>
      <x>80.26183187</x>
      <y>14.15232648</y>
   </vertex>
   <vertex>
      <x>79.40402646</x>
      <y>28.90070211</y>
   </vertex>
    ...
</spant>
```

Zeichnungsdaten im dxf-Format

```
    ...
0
POLYLINE
8
0
70
4
0
VERTEX
10
81.00000000
20
5.50000000
0
VERTEX
10
80.26183187
20
14.15232648
    ...
```

Transformatin von xy-Kooordinaten

Rohdaten im xml-Format

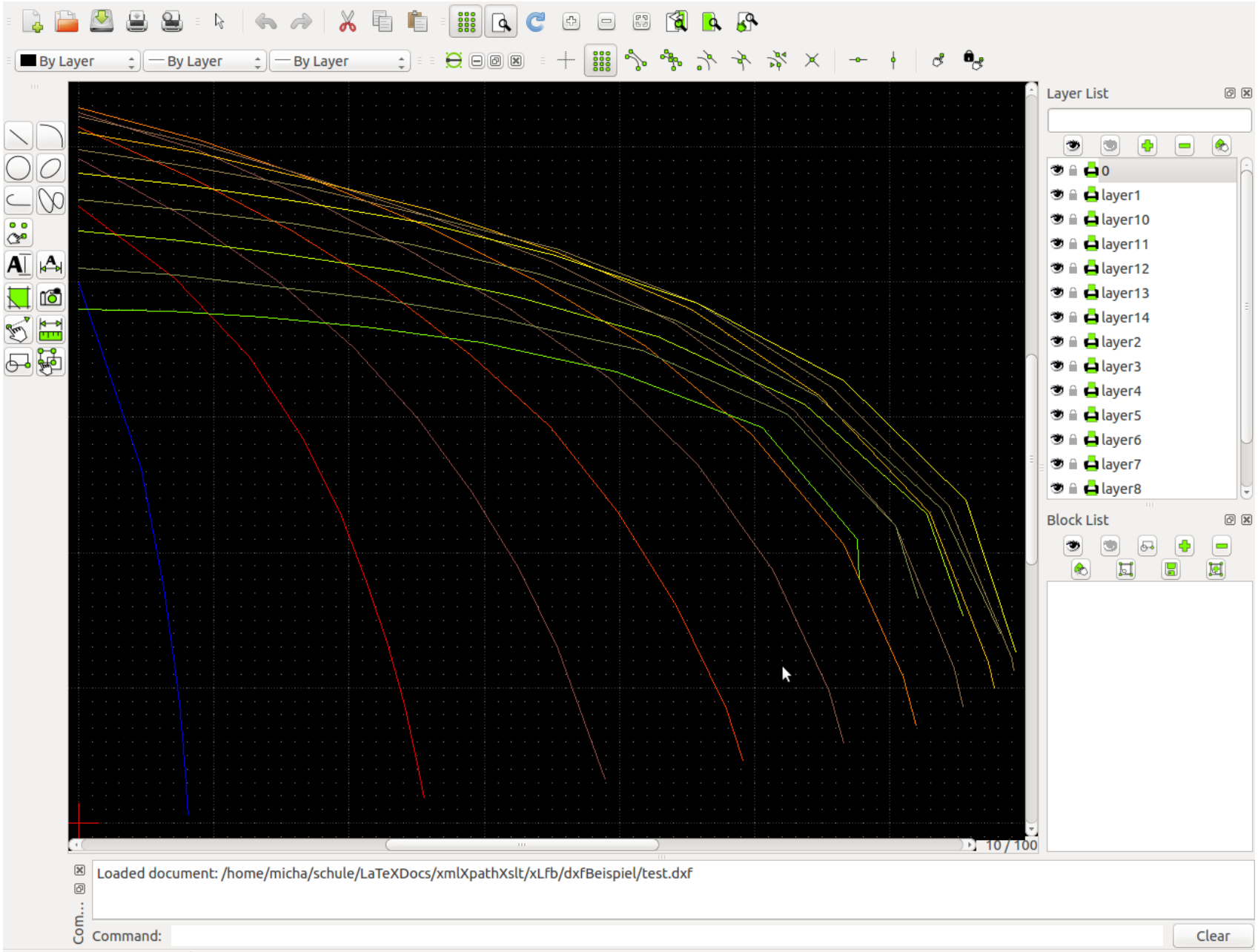
```
<spant nr="1">
   <vertex>
      <x>81.00000000</x>
      <y>5.50000000</y>
   </vertex>
   <vertex>
      <x>80.26183187</x>
      <y>14.15232648</y>
   </vertex>
   <vertex>
      <x>79.40402646</x>
      <y>28.90070211</y>
   </vertex>
    ...
</spant>
```

Zeichnungsdaten im dxf-Format

```
    ...
0
POLYLINE
8
0
70
4
0
VERTEX
10
81.00000000
20
5.50000000
0
VERTEX
10
80.26183187
20
14.15232648
    ...
```

Ergebnis der Transformation im CAD-Programm

Ergebnis der Transformation im CAD-Programm



Inhalt

Datenhaltung: XML-Datei mit Theaterspielplänen

spielplan.xml

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

<spielplan>
   <eintrag>
      <titel>Die Raueber</titel>
      <autor>Friedrich Schiller</autor>
      <premiere>2014-11-11</premiere>
   </eintrag>
   <eintrag>
      <titel>Faust</titel>
      <autor>Johann Wolfgang Goethe</autor>
      <premiere>2014-12-06</premiere>
   </eintrag>
      <eintrag>
      <titel>Der zerbrochene Krug</titel>
      <autor>Heinrich von Kleist</autor>
      <premiere>2015-01-01</premiere>
   </eintrag>
</spielplan>
```

Datenhaltung: XML-Datei mit Theaterspielplänen

Aufgabe:

- Die xsl-Datei soll den xml-Spielplan so transformieren, dass der Plan als **html-Tabelle** ausgegeben wird.
- Dabei muss man sich immer wieder vergegenwärtigen, dass eine xml-Datei eine **Baumstruktur** besitzt.
- schrittweise soll ein Stylesheet, besser wäre der Name **Transformationssheet**, für die Transformation entwickelt werden.

Datenhaltung: XML-Datei mit Theaterspielplänen

Aufgabe:

- Die xsl-Datei soll den xml-Spielplan so transformieren, dass der Plan als **html-Tabelle** ausgegeben wird.
- Dabei muss man sich immer wieder vergegenwärtigen, dass eine xml-Datei eine **Baumstruktur** besitzt.
- schrittweise soll ein Stylesheet, besser wäre der Name **Transformationssheet**, für die Transformation entwickelt werden.

Datenhaltung: XML-Datei mit Theaterspielplänen

Aufgabe:

- Die xsl-Datei soll den xml-Spielplan so transformieren, dass der Plan als **html-Tabelle** ausgegeben wird.
- Dabei muss man sich immer wieder vergegenwärtigen, dass eine xml-Datei eine **Baumstruktur** besitzt.
- schrittweise soll ein Stylesheet, besser wäre der Name **Transformationssheet**, für die Transformation entwickelt werden.

Datenhaltung: XML-Datei mit Theaterspielplänen

Aufgabe:

- Die xsl-Datei soll den xml-Spielplan so transformieren, dass der Plan als **html-Tabelle** ausgegeben wird.
- Dabei muss man sich immer wieder vergegenwärtigen, dass eine xml-Datei eine **Baumstruktur** besitzt.
- schrittweise soll ein Stylesheet, besser wäre der Name **Transformationssheet**, für die Transformation entwickelt werden.

Datenhaltung: XML-Datei mit Theaterspielplänen

Aufgabe:

- Die xsl-Datei soll den xml-Spielplan so transformieren, dass der Plan als **html-Tabelle** ausgegeben wird.
- Dabei muss man sich immer wieder vergegenwärtigen, dass eine xml-Datei eine **Baumstruktur** besitzt.
- schrittweise soll ein Stylesheet, besser wäre der Name **Transformationssheet**, für die Transformation entwickelt werden.

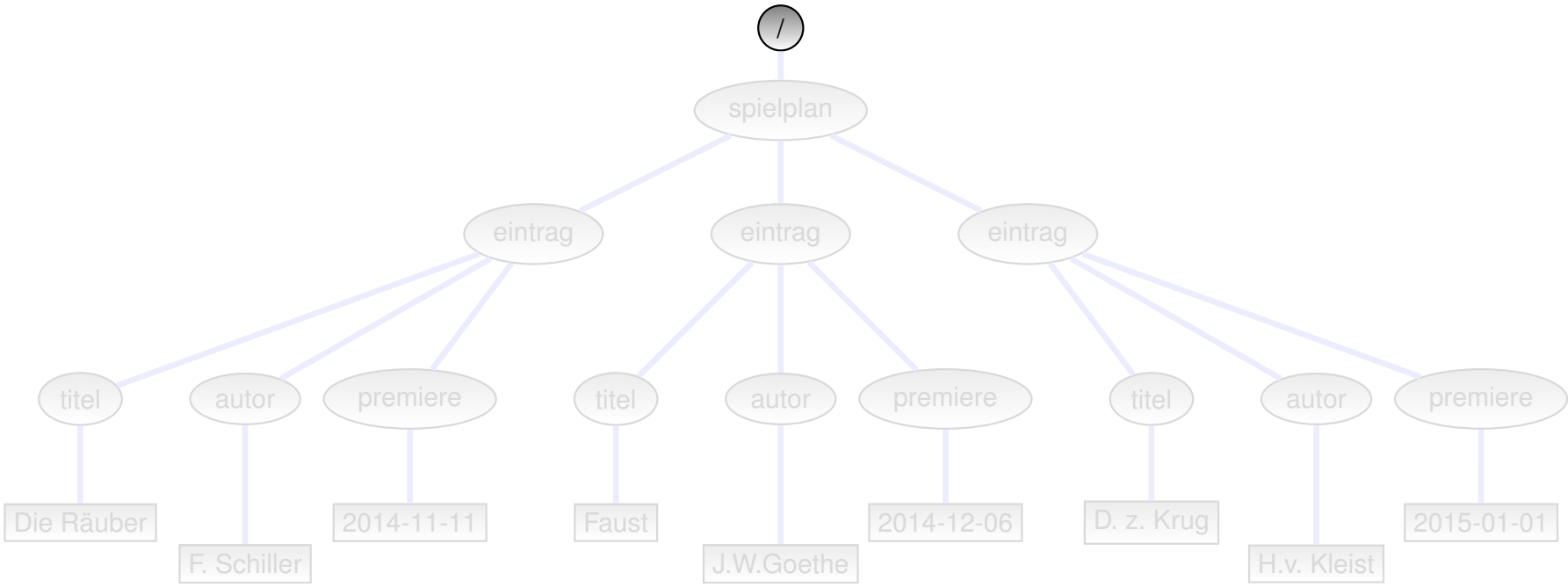
Datenhaltung: XML-Datei mit Theaterspielplänen

Aufgabe:

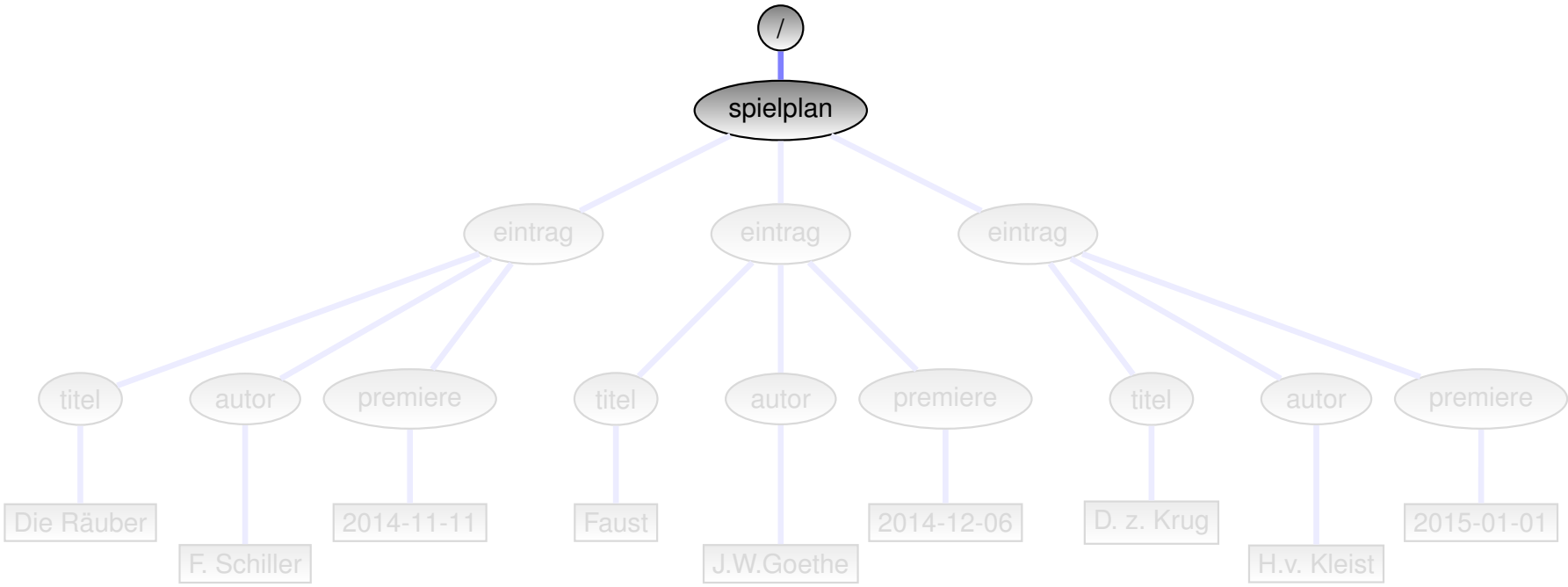
- Die xsl-Datei soll den xml-Spielplan so transformieren, dass der Plan als **html-Tabelle** ausgegeben wird.
- Dabei muss man sich immer wieder vergegenwärtigen, dass eine xml-Datei eine **Baumstruktur** besitzt.
- schrittweise soll ein Stylesheet, besser wäre der Name **Transformationssheet**, für die Transformation entwickelt werden.

Baumdarstellung von spielplan.xml

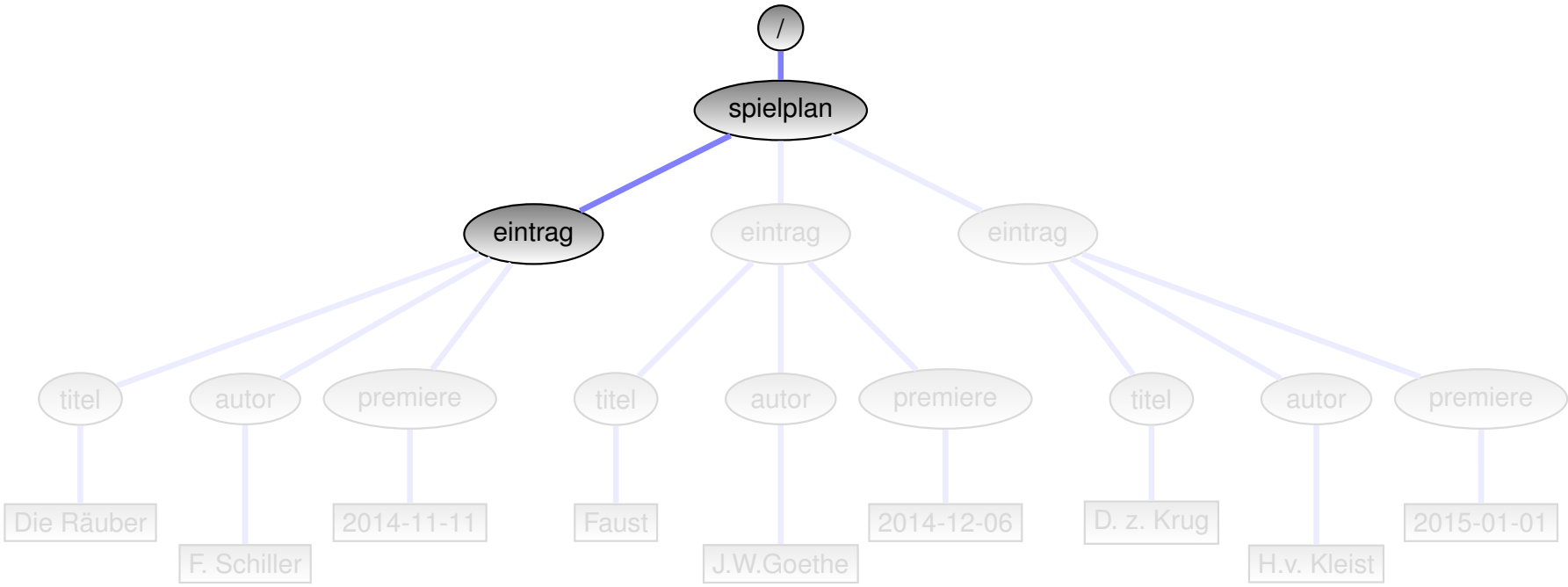
Baumdarstellung von spielplan.xml



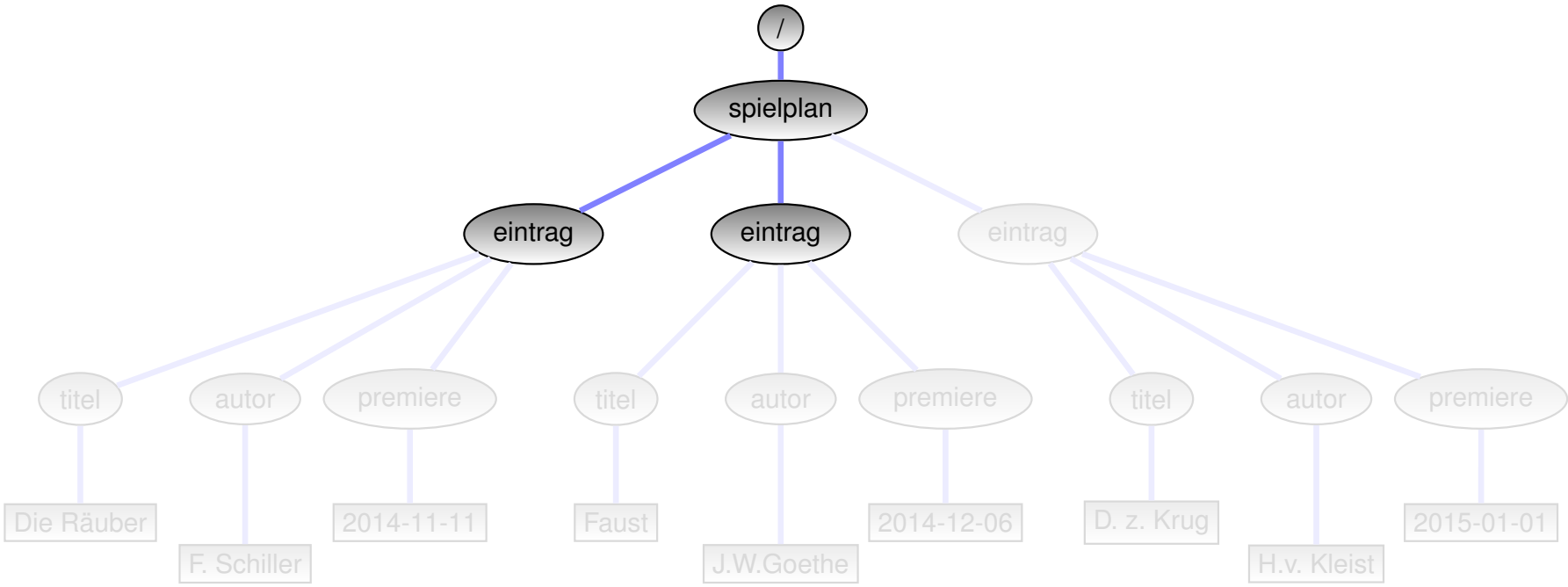
Baumdarstellung von spielplan.xml



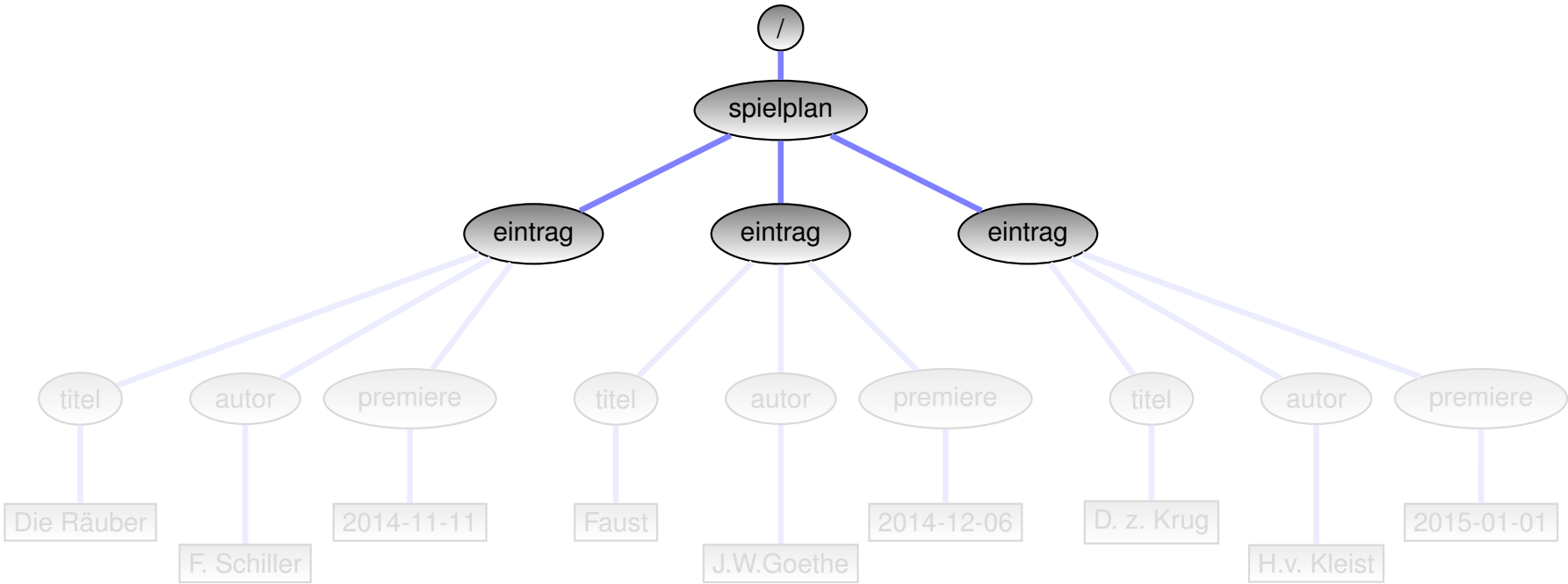
Baumdarstellung von spielplan.xml



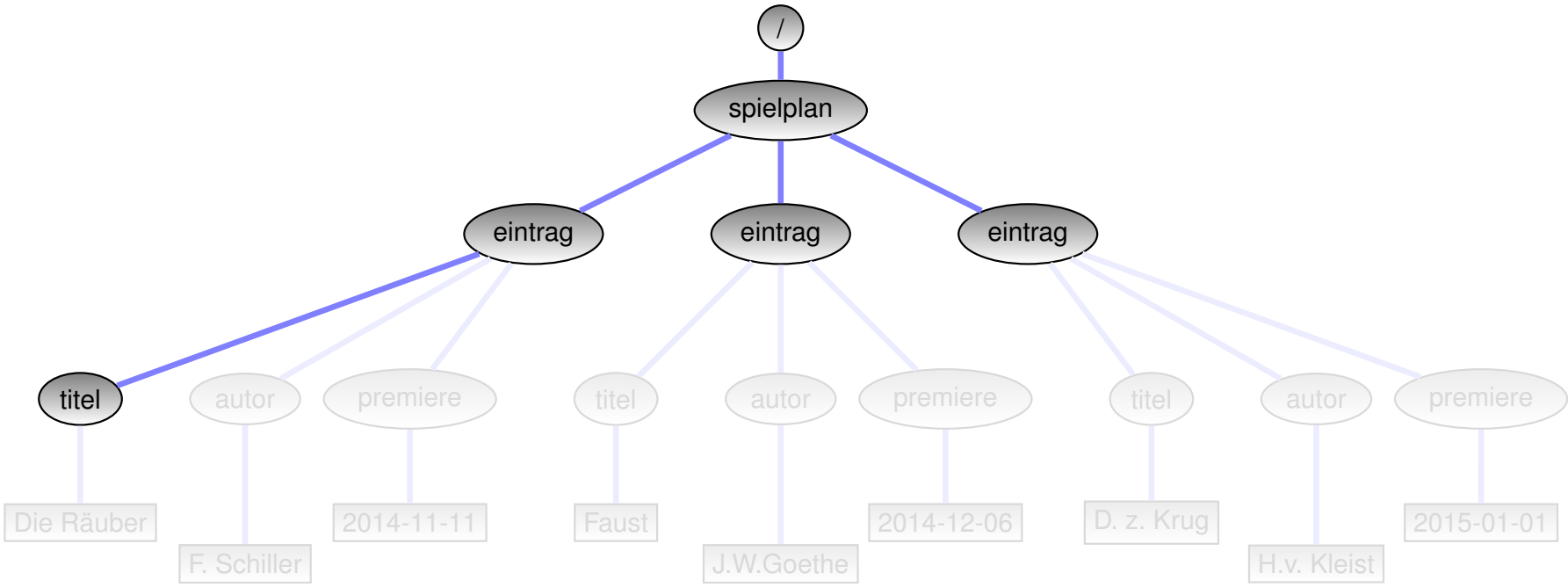
Baumdarstellung von spielplan.xml



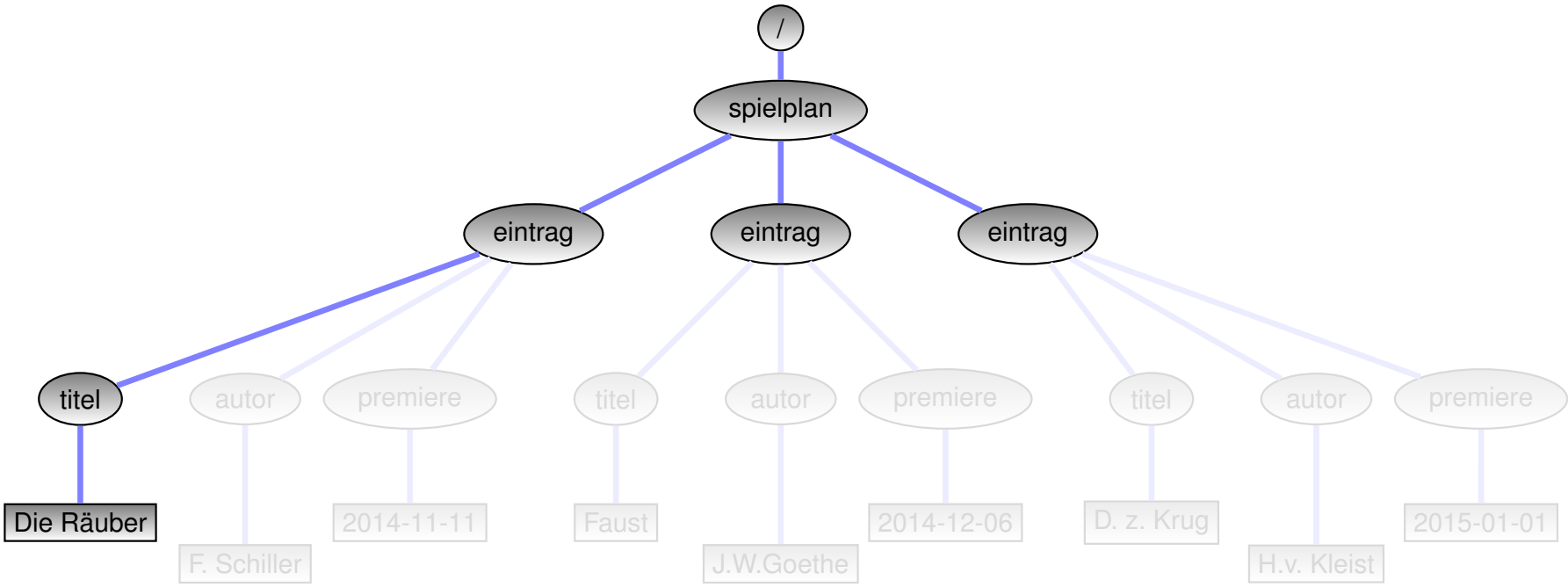
Baumdarstellung von spielplan.xml



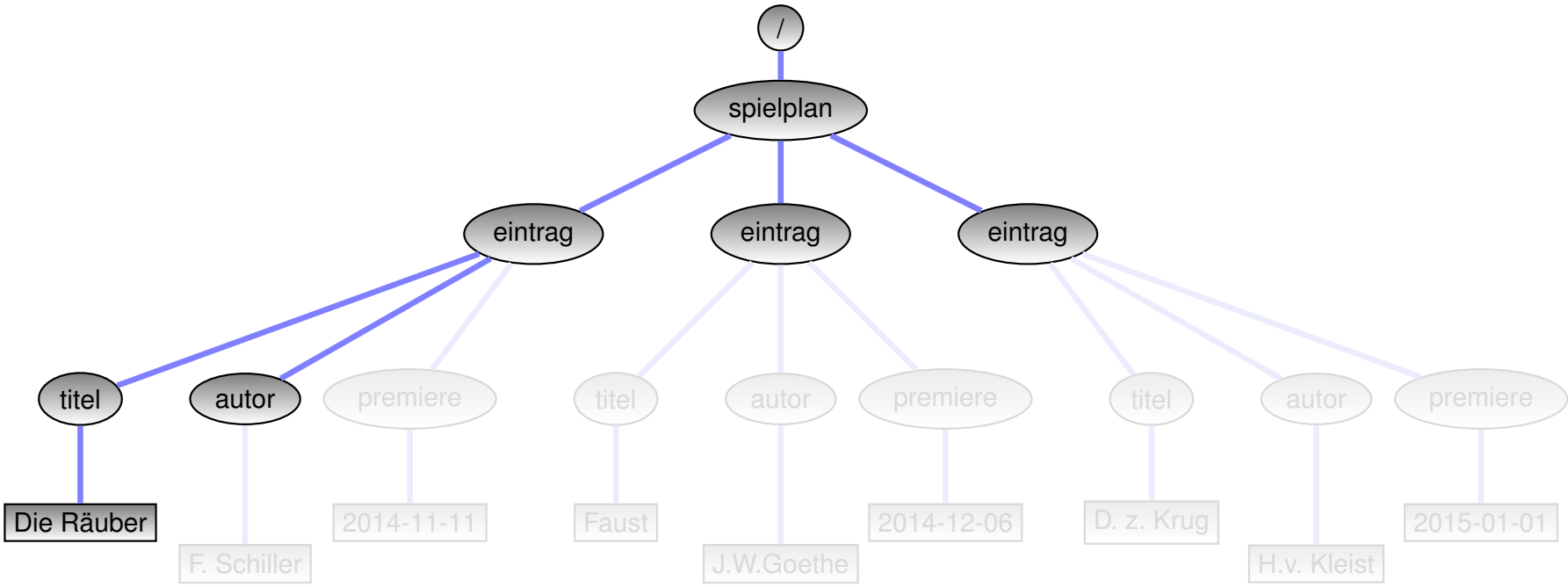
Baumdarstellung von spielplan.xml



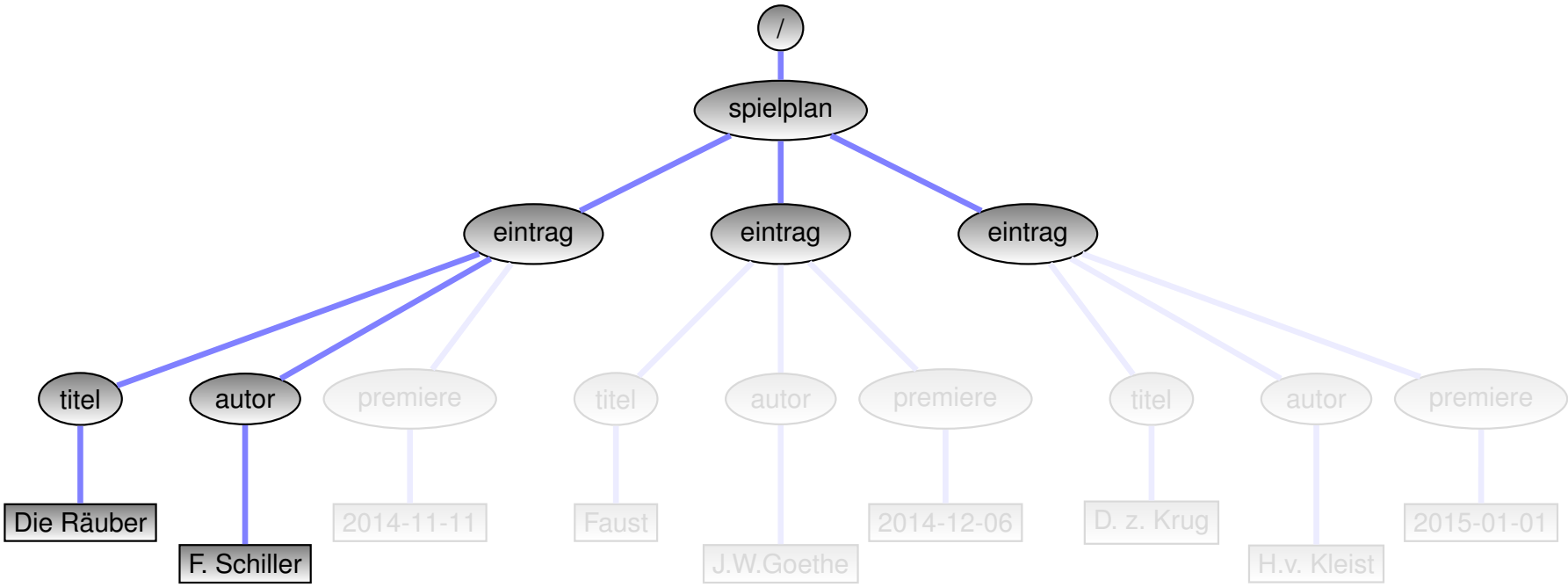
Baumdarstellung von spielplan.xml



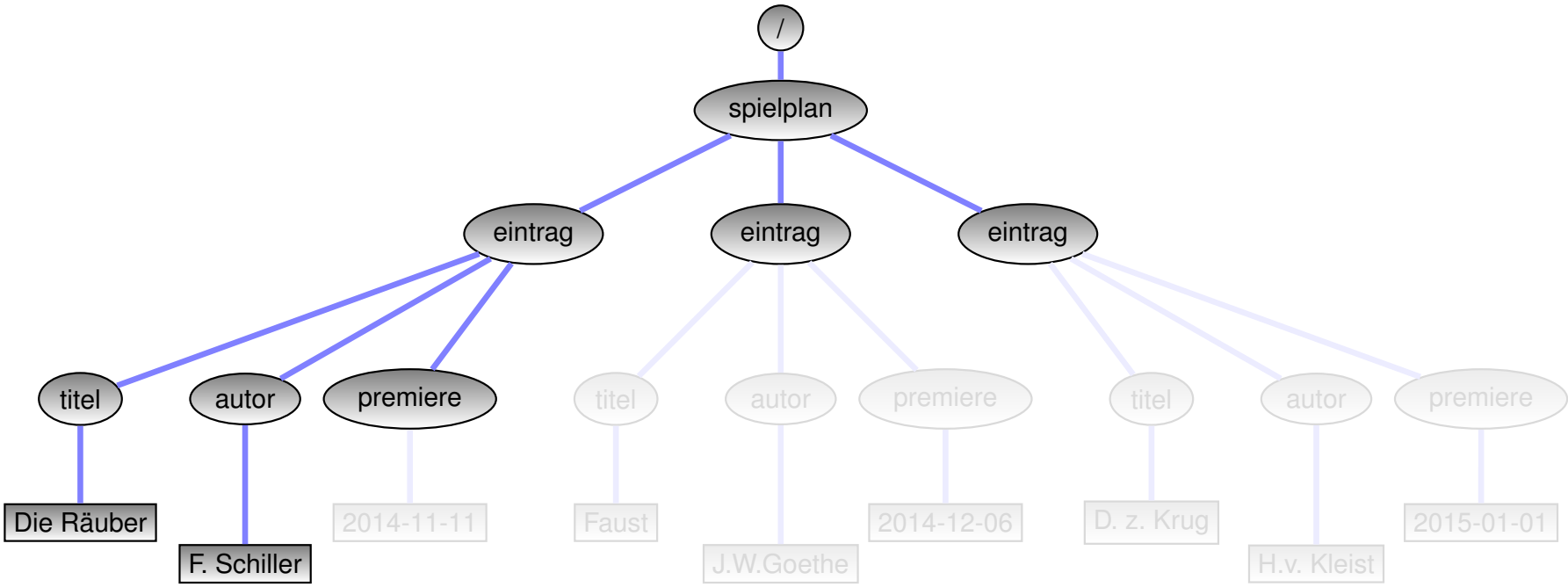
Baumdarstellung von spielplan.xml



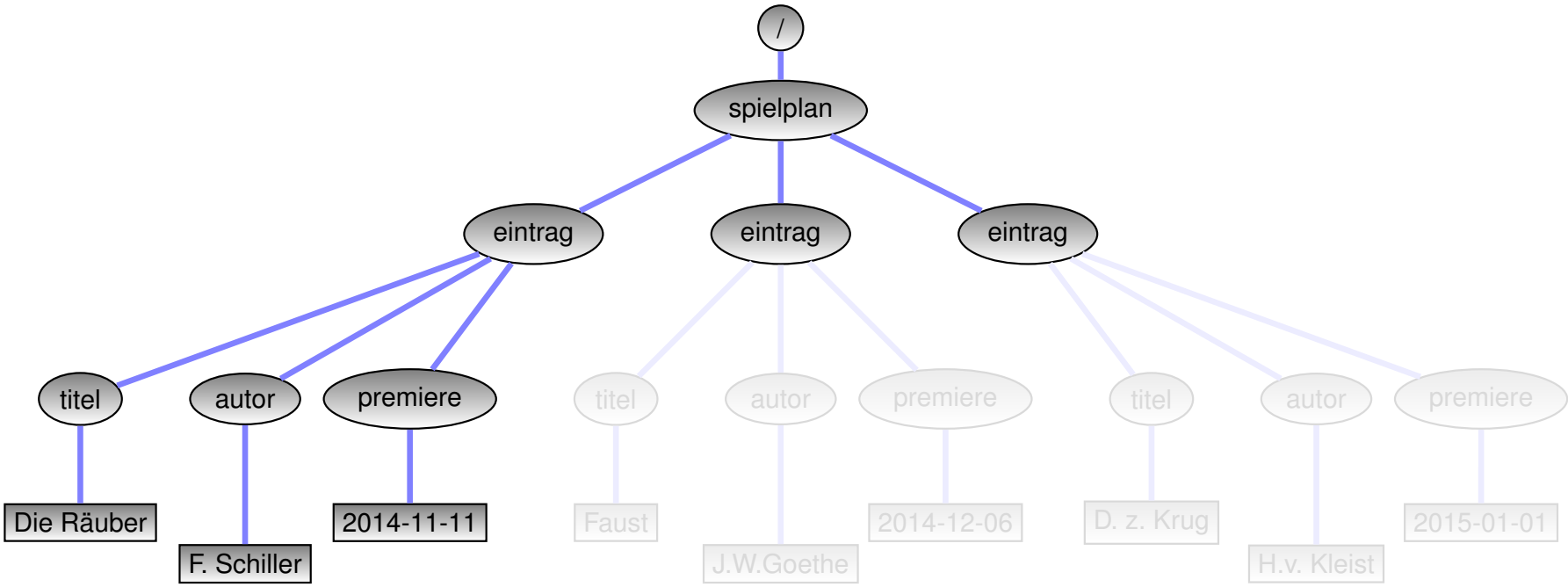
Baumdarstellung von spielplan.xml



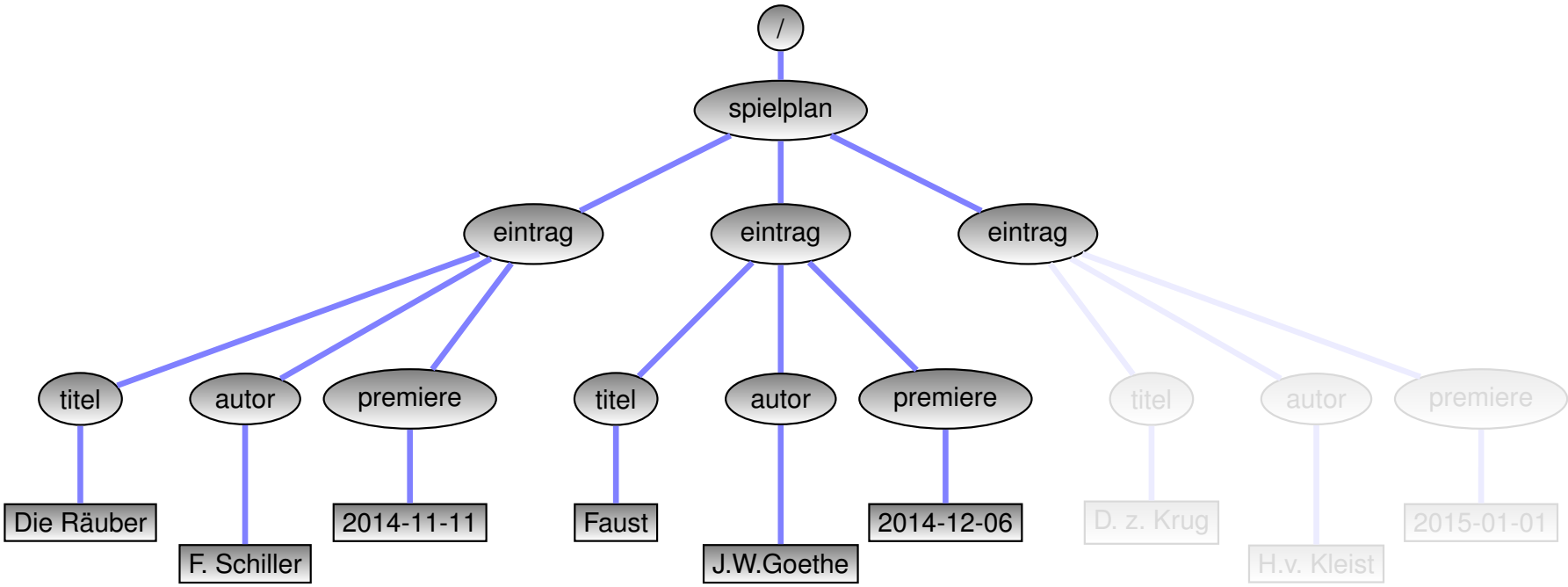
Baumdarstellung von spielplan.xml



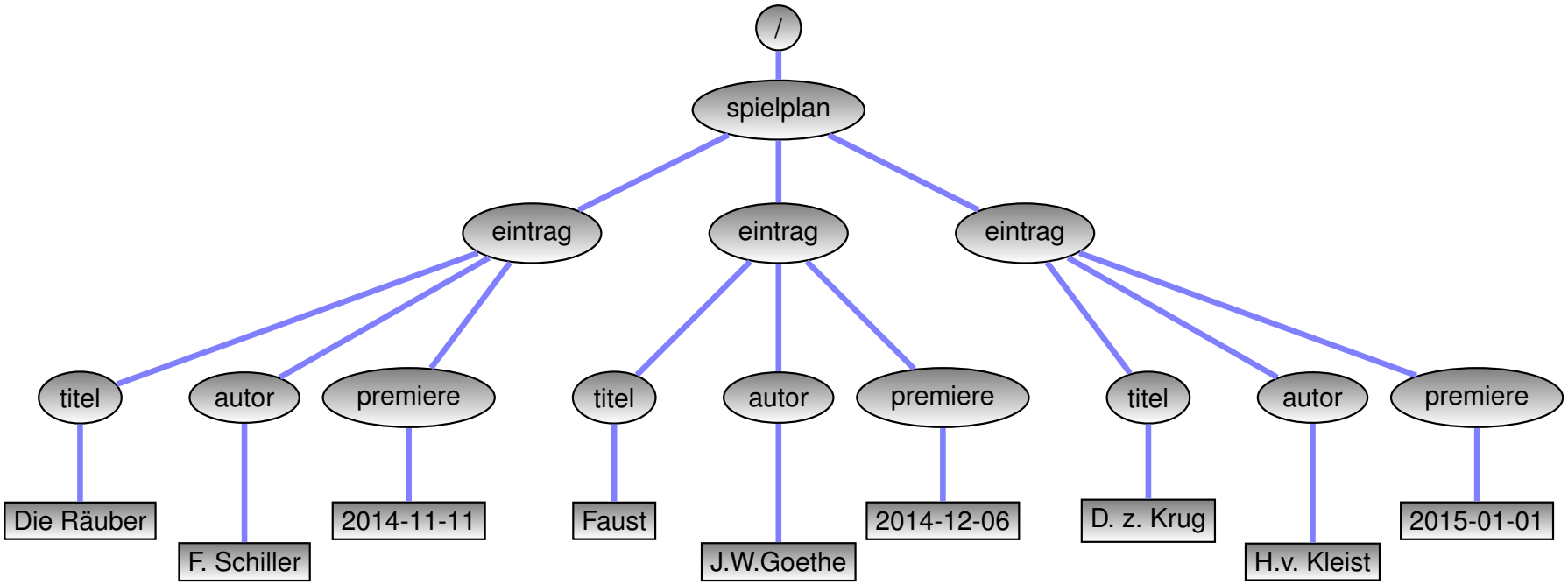
Baumdarstellung von spielplan.xml



Baumdarstellung von spielplan.xml



Baumdarstellung von spielplan.xml



Inhalt

Template Rules

Der wichtigste Bestandteil einer xslt-Datei sind die sog. **Template Rules**. Eine Template Rule hat folgende Syntax:

```
<xsl:template match="xpath-Lokalisierungsschritt">
...
</xsl:template>
```

Der Namensraumpräfix muss nicht **xsl** heissen, dies ist jedoch allgemeine Praxis und sinnvoll.
Der xpath-Lokalisierungsschritt wählt dabei den Teil des Originaldokuments aus, für den die Template Rule gelten soll.

Template Rules

Der wichtigste Bestandteil einer xslt-Datei sind die sog. **Template Rules**. Eine Template Rule hat folgende Syntax:

```
<xsl:template match="xpath-Lokalisierungsschritt">
...
</xsl:template>
```

Der Namensraumpräfix muss nicht **xsl** heissen, dies ist jedoch allgemeine Praxis und sinnvoll.
Der xpath-Lokalisierungsschritt wählt dabei den Teil des Originaldokuments aus, für den die Template Rule gelten soll.

Template Rules

Der wichtigste Bestandteil einer xslt-Datei sind die sog. **Template Rules**. Eine Template Rule hat folgende Syntax:

```
<xsl:template match="xpath-Lokalisierungsschritt">
...
</xsl:template>
```

Der Namensraumpräfix muss nicht **xsl** heissen, dies ist jedoch allgemeine Praxis und sinnvoll.
Der xpath-Lokalisierungsschritt wählt dabei den Teil des Originaldokuments aus, für den die Template Rule gelten soll.

Template Rules

Der wichtigste Bestandteil einer xslt-Datei sind die sog. **Template Rules**. Eine Template Rule hat folgende Syntax:

```
<xsl:template match="xpath-Lokalisierungsschritt">
...
</xsl:template>
```

Der Namensraumpräfix muss nicht **xsl** heissen, dies ist jedoch allgemeine Praxis und sinnvoll.

Der xpath-Lokalisierungsschritt wählt dabei den Teil des Originaldokuments aus, für den die Template Rule gelten soll.

Template Rules

Der wichtigste Bestandteil einer xslt-Datei sind die sog. **Template Rules**. Eine Template Rule hat folgende Syntax:

```
<xsl:template match="xpath-Lokalisierungsschritt">
...
</xsl:template>
```

Der Namensraumpräfix muss nicht **xsl** heissen, dies ist jedoch allgemeine Praxis und sinnvoll.
Der xpath-Lokalisierungsschritt wählt dabei den Teil des Originaldokuments aus, für den die Template Rule gelten soll.

Wie eine Template Rule funktioniert

Innerhalb der Template Rule gelten nun folgende, einfache, **ganz wichtige** Regeln:

1. Führe alle Anweisungen aus, deren Markup zum xsl-Namensraum gehört
2. Gib den Rest ins Ergebnisdokument aus.

Wie eine Template Rule funktioniert

Innerhalb der Template Rule gelten nun folgende, einfache, **ganz wichtige** Regeln:

1. Führe alle Anweisungen aus, deren Markup zum xsl-Namensraum gehört
2. Gib den Rest ins Ergebnisdokument aus.

Wie eine Template Rule funktioniert

Innerhalb der Template Rule gelten nun folgende, einfache, **ganz wichtige** Regeln:

1. Führe alle Anweisungen aus, deren Markup zum xsl-Namensraum gehört
2. Gib den Rest ins Ergebnisdokument aus.

Werte aus der Quelle ins Ziel übernehmen

Nun möchte man evtl. Daten aus dem Quelldokument ins Zieldokument übernehmen. Das macht man mit folgendem Ausdruck:

```
<xsl:value-of select="xpath-Ausdruck"/>
```

Werte aus der Quelle ins Ziel übernehmen

Nun möchte man evtl. Daten aus dem Quelldokument ins Zieldokument übernehmen. Das macht man mit folgendem Ausdruck:

```
<xsl:value-of select="xpath-Ausdruck"/>
```

Ausführen der Template-Rules

Eine Template-Rule lässt sich mit foldenden Ausdrücken aufrufen:

```
<xsl:apply-templates/>
```

Die Wirkung dieser Anweisung ist einfach:

- selektiere **alle Kinder** des aktuellen Knotens; man erhält eine $\Rightarrow$ Knotenmenge (Node-Set)
- suche für jedes Kind eine passende (gemäss match-Attribut) *Template-Rule* und führe diese aus.

Ausführen der Template-Rules

Eine Template-Rule lässt sich mit foldenden Ausdrücken aufrufen:

```
<xsl:apply-templates/>
```

Die Wirkung dieser Anweisung ist einfach:

- selektiere **alle Kinder** des aktuellen Knotens; man erhält eine $\Rightarrow$ Knotenmenge (Node-Set)
- suche für jedes Kind eine passende (gemäss match-Attribut) *Template-Rule* und führe diese aus.

Ausführen der Template-Rules

Eine Template-Rule lässt sich mit foldenden Ausdrücken aufrufen:

```
<xsl:apply-templates/>
```

Die Wirkung dieser Anweisung ist einfach:

- selektiere **alle Kinder** des aktuellen Knotens; man erhält eine $\Rightarrow$ Knotenmenge (Node-Set)
- suche für jedes Kind eine passende (gemäss match-Attribut) *Template-Rule* und führe diese aus.

Ausführen der Template-Rules

Eine Template-Rule lässt sich mit foldenden Ausdrücken aufrufen:

```
<xsl:apply-templates/>
```

Die Wirkung dieser Anweisung ist einfach:

- selektiere **alle Kinder** des aktuellen Knotens; man erhält eine $\Rightarrow$ Knotenmenge (Node-Set)
- suche für jedes Kind eine passende (gemäss match-Attribut) *Template-Rule* und führe diese aus.

Ausführen der Template-Rules

Eine Template-Rule lässt sich mit foldenden Ausdrücken aufrufen:

```
<xsl:apply-templates/>
```

Die Wirkung dieser Anweisung ist einfach:

- selektiere **alle Kinder** des aktuellen Knotens; man erhält eine $\Rightarrow$ Knotenmenge (Node-Set)
- suche für jedes Kind eine passende (gemäss match-Attribut) *Template-Rule* und führe diese aus.

Ausführen der Template-Rules

Eine Template-Rule lässt sich mit foldenden Ausdrücken aufrufen:

```
<xsl:apply-templates/>
```

Die Wirkung dieser Anweisung ist einfach:

- selektiere **alle Kinder** des aktuellen Knotens; man erhält eine ⇒ Knotenmenge (Node-Set)
- suche für jedes Kind eine passende (gemäss match-Attribut) *Template-Rule* und führe diese aus.

Ausführen der Template-Rules

Eine Template-Rule lässt sich mit foldenden Ausdrücken aufrufen:

```
<xsl:apply-templates/>
```

Die Wirkung dieser Anweisung ist einfach:

- selektiere **alle Kinder** des aktuellen Knotens; man erhält eine $\Rightarrow$ Knotenmenge (Node-Set)
- suche für jedes Kind eine passende (gemäss match-Attribut) *Template-Rule* und führe diese aus.

Ausführen der Template-Rules

Eine Template-Rule lässt sich mit foldenden Ausdrücken aufrufen:

```
<xsl:apply-templates/>
```

Die Wirkung dieser Anweisung ist einfach:

- selektiere **alle Kinder** des aktuellen Knotens; man erhält eine $\Rightarrow$ Knotenmenge (Node-Set)
- suche für jedes Kind eine passende (gemäss match-Attribut) *Template-Rule* und führe diese aus.

Beispiele für Knotenmengen (node-sets)

```
<xsl:template match="/spielplan">
   <xsl:apply-templates/>
</xsl:template>
```



```
<xsl:template match="eintrag">
  <xsl:apply-templates/>
</xsl:template>
```

Beispiele für Knotenmengen (node-sets)

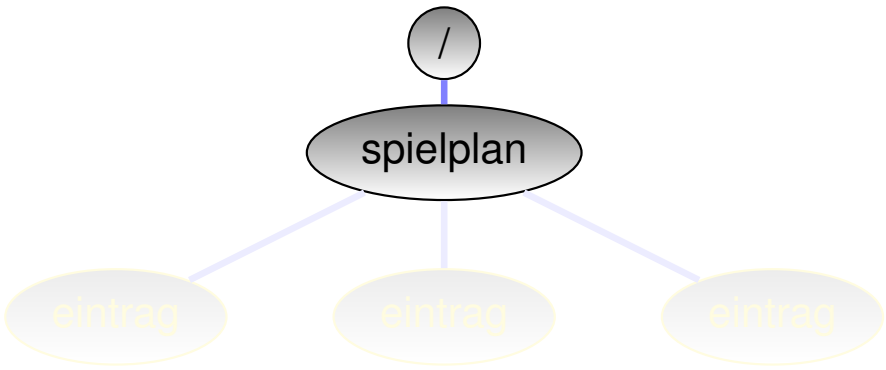
```
<xsl:template match="/spielplan">
   <xsl:apply-templates/>
</xsl:template>
```



```
<xsl:template match="eintrag">
  <xsl:apply-templates/>
</xsl:template>
```

Beispiele für Knotenmengen (node-sets)

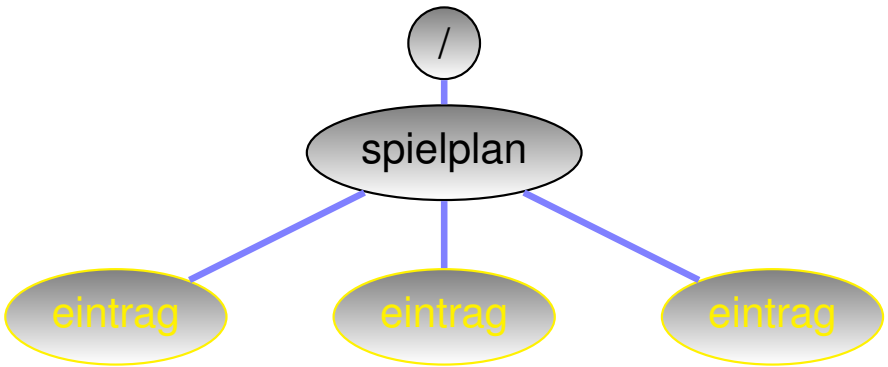
```
<xsl:template match="/spielplan">
   <xsl:apply-templates/>
</xsl:template>
```




```
<xsl:template match="eintrag">
  <xsl:apply-templates/>
</xsl:template>
```

Beispiele für Knotenmengen (node-sets)

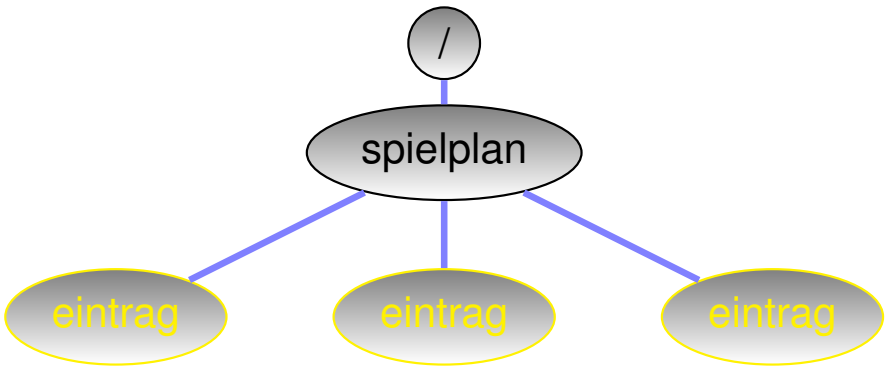
```
<xsl:template match="/spielplan">
   <xsl:apply-templates/>
</xsl:template>
```




```
<xsl:template match="eintrag">
  <xsl:apply-templates/>
</xsl:template>
```

Beispiele für Knotenmengen (node-sets)

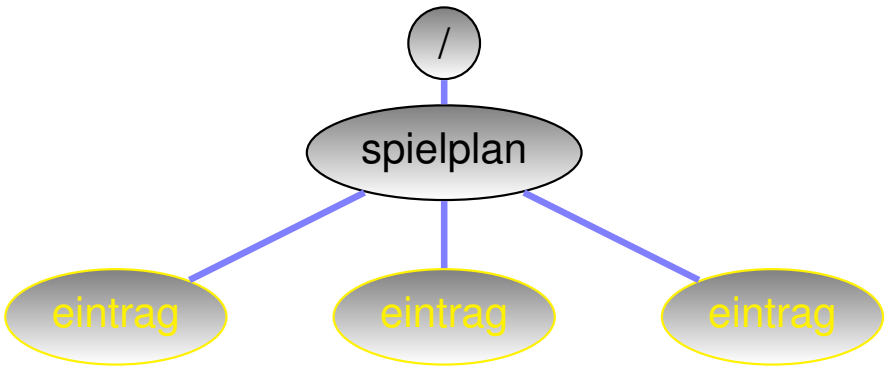
```
<xsl:template match="/spielplan">
   <xsl:apply-templates/>
</xsl:template>
```



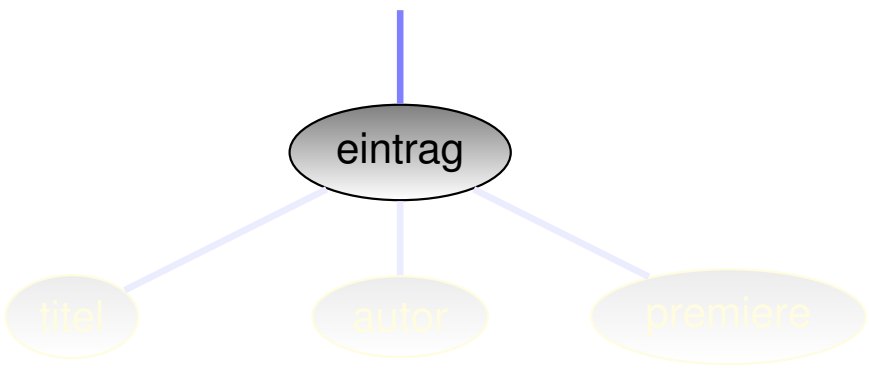

```
<xsl:template match="eintrag">
  <xsl:apply-templates/>
</xsl:template>
```

Beispiele für Knotenmengen (node-sets)

```
<xsl:template match="/spielplan">
   <xsl:apply-templates/>
</xsl:template>
```

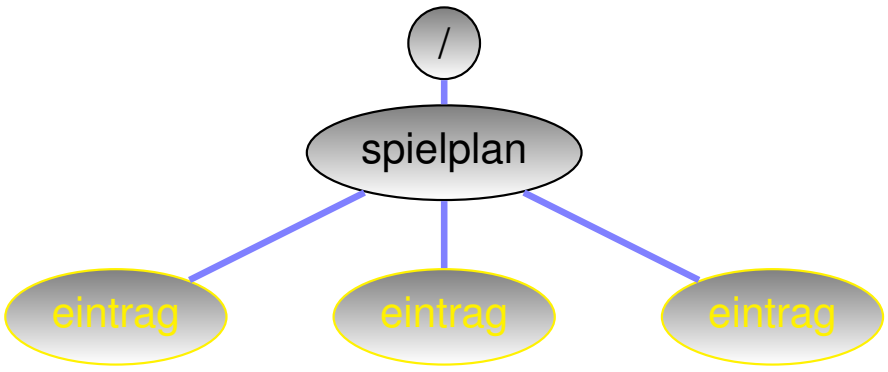



```
<xsl:template match="eintrag">
  <xsl:apply-templates/>
</xsl:template>
```

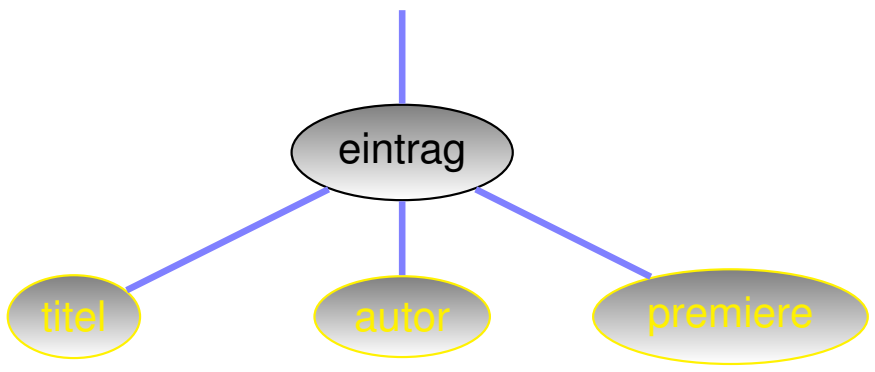


Beispiele für Knotenmengen (node-sets)

```
<xsl:template match="/spielplan">
   <xsl:apply-templates/>
</xsl:template>
```




```
<xsl:template match="eintrag">
  <xsl:apply-templates/>
</xsl:template>
```



Direktes Ausführen einer Template-Rule

Mehr Kontrolle über den Transformationsvorgang hat man mit dem optionalen Attribut select :

```
<xsl:apply-templates select="xpath-Ausdruck"/>
```

Mit dem optionalen xpath-Ausdruck lässt sich ein Kindsknoten gezielt ansteuern. Lässt man ihn weg, werden die Template Rules *aller* Kinder ausgeführt.

Direktes Ausführen einer Template-Rule

Mehr Kontrolle über den Transformationsvorgang hat man mit dem optionalen Attribut select :

```
<xsl:apply-templates select="xpath-Ausdruck"/>
```

Mit dem optionalen xpath-Ausdruck lässt sich ein Kindsknoten gezielt ansteuern. Lässt man ihn weg, werden die Template Rules *aller* Kinder ausgeführt.

Direktes Ausführen einer Template-Rule

Mehr Kontrolle über den Transformationsvorgang hat man mit dem optionalen Attribut select :

```
<xsl:apply-templates select="xpath-Ausdruck"/>
```

Mit dem optionalen xpath-Ausdruck lässt sich ein Kindsknoten gezielt ansteuern. Lässt man ihn weg, werden die Template Rules *aller* Kinder ausgeführt.

Direktes Ausführen einer Template-Rule

Mehr Kontrolle über den Transformationsvorgang hat man mit dem optionalen Attribut select :

```
<xsl:apply-templates select="xpath-Ausdruck"/>
```

Mit dem optionalen xpath-Ausdruck lässt sich ein Kindsknoten gezielt ansteuern. Lässt man ihn weg, werden die Template Rules *aller* Kinder ausgeführt.

Rekursives Abarbeiten des Quellbaums

Da das zu transformierende Dokument eine Baumstruktur besitzt (es ist ein xml-Dokument), wird bevorzugt **rekursiv** geaerbeitet.

```
<xsl:template match="eintrag">
   <tr>
      <xsl:apply-templates/>
   </tr>
</xsl:template>
```

Rekursives Abarbeiten des Quellbaums

Da das zu transformierende Dokument eine Baumstruktur besitzt (es ist ein xml-Dokument), wird bevorzugt **rekursiv** geaerbeitet.

```
<xsl:template match="eintrag">
   <tr>
      <xsl:apply-templates/>
   </tr>
</xsl:template>
```

Zurück zum Spielplan-Beispiel

Es soll die xml-Datei mit dem Theater-Spielplan als html-Tabelle ausgegeben werden.

Zurück zum Spielplan-Beispiel

Zuerst ein bisschen Bürokratie:

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

<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" encoding="iso-8859-1"/>

      ....

</xsl:stylesheet>
```

Da xsl-Stylesheets selbst wieder wohlgeformte xml-Dokumente sind, muss es ein Wurzelelement `<xsl:stylesheet>` geben.

Zurück zum Spielplan-Beispiel

Zuerst ein bisschen Bürokratie:

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

<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html" encoding="iso-8859-1"/>

      ....

</xsl:stylesheet>
```

Da xsl-Stylesheets selbst wieder wohlgeformte xml-Dokumente sind, muss es ein Wurzelelement `<xsl:stylesheet>` geben.

Zurück zum Spielplan-Beispiel

Zuerst ein bisschen Bürokratie:

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

<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html" encoding="iso-8859-1"/>

      ....

</xsl:stylesheet>
```

Da xsl-Stylesheets selbst wieder wohlgeformte xml-Dokumente sind, muss es ein Wurzelelement `<xsl:stylesheet>` geben.

Zurück zum Spielplan-Beispiel

Zuerst ein bisschen Bürokratie:

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

<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html" encoding="iso-8859-1"/>

      ....

</xsl:stylesheet>
```

Da xsl-Stylesheets selbst wieder wohlgeformte xml-Dokumente sind, muss es ein Wurzelelement `<xsl:stylesheet>` geben.

Zurück zum Spielplan-Beispiel

Das erste Template gilt für den Wurzelknoten (”/”), das gesamte Dokument (document node) selbst:

```
                    ...

<xsl:template match="/">
   <xsl:apply-templates/>
</xsl:template>

<xsl:template match="spielplan">
                    ...
   <xsl:apply-templates/>
                    ...
</xsl:template>
                    ...
```

Zurück zum Spielplan-Beispiel

Das erste Template gilt für den Wurzelknoten (”/”), das gesamte Dokument (document node) selbst:

```
                    ...

<xsl:template match="/">
   <xsl:apply-templates/>
</xsl:template>

<xsl:template match="spielplan">
                    ...
   <xsl:apply-templates/>
                    ...
</xsl:template>
                    ...
```

Zurück zum Spielplan-Beispiel

Da das Wurzel**element** keine Nachbarn hat, kann man die Templates zusammenfassen:

```
                    ...

<xsl:template match="/spielplan">
                    ...
   <xsl:apply-templates/>
                    ...
</xsl:template>
                    ...
```

Zurück zum Spielplan-Beispiel

Da das Wurzel**element** keine Nachbarn hat, kann man die Templates zusammenfassen:

```
                    ...

<xsl:template match="/spielplan">
                    ...
   <xsl:apply-templates/>
                    ...
</xsl:template>
                    ...
```

Zurück zum Spielplan-Beispiel

Die nächsten Elemente im Baum und eine logische Verknüpfung:

```
                    ...

<xsl:template match="/spielplan"> ...
</xsl:template>

<xsl:template match="eintrag"> ...
</xsl:template>

<xsl:template match="titel | autor | premiere"> ...
</xsl:template>
                    ...
```

Zurück zum Spielplan-Beispiel

Die nächsten Elemente im Baum und eine logische Verknüpfung:

```
                    ...

<xsl:template match="/spielplan"> ...
</xsl:template>

<xsl:template match="eintrag"> ...
</xsl:template>

<xsl:template match="titel | autor | premiere"> ...
</xsl:template>
                    ...
```

Wie soll das Ergebnis der Transformation aussehen?

```
<html xmlns="http://www.w3.org/1999/xhtml">
   <head><title>Eine erste xslt-Uebung</title></head>
   <body>
      <h1>Spielplan des Klassischen Theaters</h1>
      <table border="1">
         <tr>
            <th>Titel</th>
            <th>Autor</th>
            <th>Premiere</th>
         </tr>
         <tr>
            <td>Die Raueber</td>
            <td>Friedrich Schiller</td>
            <td>8. Juli 2010</td>
        </tr>
        <tr> ...  </tr>
        <tr> ...  </tr>
     </table>
   </body>
</html>
```

Wie kommt der html-Markup ins Ergebnis?

Im Template für `/spielplan`: html-head und -body deklarieren, Tabelle mit Kopfzeile einleiten, **pro Tabellenzeile nächstes Template aufrufen**

Im Template für `eintrag`: eine neue Tabellenzeile einleiten, nächstes Template aufrufen

In den Templates für `titel, autor, premiere`: eine Tabellenzeile einleiten und mit den Daten aus dem xml-Dokument füllen

Zusammenfassen gleicher Aktionen: um Redundanz zu vermeiden, können Templates mit gleichen Aktionen zusammengefasst werden

Wie kommt der html-Markup ins Ergebnis?

Im Template für `/spielplan`: html-head und -body deklarieren, Tabelle mit Kopfzeile einleiten, **pro Tabellenzeile nächstes Template aufrufen**

Im Template für `eintrag`: eine neue Tabellenzeile einleiten, nächstes Template aufrufen

In den Templates für `titel, autor, premiere`: eine Tabellenzelle einleiten und mit den Daten aus dem xml-Dokument füllen

Zusammenfassen gleicher Aktionen: um Redundanz zu vermeiden, können Templates mit gleichen Aktionen zusammengefasst werden

Wie kommt der html-Markup ins Ergebnis?

Im Template für `/spielplan`: html-head und -body deklarieren, Tabelle mit Kopfzeile einleiten, **pro Tabellenzeile nächstes Template aufrufen**

Im Template für `eintrag`: eine neue Tabellenzeile einleiten, nächstes Template aufrufen

In den Templates für `titel, autor, premiere`: eine Tabellenzelle einleiten und mit den Daten aus dem xml-Dokument füllen

Zusammenfassen gleicher Aktionen: um Redundanz zu vermeiden, können Templates mit gleichen Aktionen zusammengefasst werden

Wie kommt der html-Markup ins Ergebnis?

Im Template für `/spielplan`: html-head und -body deklarieren, Tabelle mit Kopfzeile einleiten, **pro Tabellenzeile nächstes Template aufrufen**

Im Template für `eintrag`: eine neue Tabellenzeile einleiten, nächstes Template aufrufen

In den Templates für `titel, autor, premiere`: eine Tabellenzelle einleiten und mit den Daten aus dem xml-Dokument füllen

Zusammenfassen gleicher Aktionen: um Redundanz zu vermeiden, können Templates mit gleichen Aktionen zusammengefasst werden

Wie kommt der html-Markup ins Ergebnis?

Im Template für `/spielplan`: html-head und -body deklarieren, Tabelle mit Kopfzeile einleiten, **pro Tabellenzeile nächstes Template aufrufen**

Im Template für `eintrag`: eine neue Tabellenzeile einleiten, nächstes Template aufrufen

In den Templates für `titel, autor, premiere`: eine Tabellenzelle einleiten und mit den Daten aus dem xml-Dokument füllen

Zusammenfassen gleicher Aktionen: um Redundanz zu vermeiden, können Templates mit gleichen Aktionen zusammengefasst werden

Erinnerung: wie arbeitet der XSLT-Prozessor?

- Führe alle Anweisungen aus, deren Markup zum xsl-Namensraum gehört
- Gib den Rest ins Ergebnisdokument aus.
- Daten aus dem Quelldokument ins Zieldokument übernehmen:

```
<xsl:value-of select="xpath-Ausdruck"/>
```

Erinnerung: wie arbeitet der XSLT-Prozessor?

- Führe alle Anweisungen aus, deren Markup zum xsl-Namensraum gehört
- Gib den Rest ins Ergebnisdokument aus.
- Daten aus dem Quelldokument ins Zieldokument übernehmen:

```
<xsl:value-of select="xpath-Ausdruck"/>
```

Erinnerung: wie arbeitet der XSLT-Prozessor?

- Führe alle Anweisungen aus, deren Markup zum xsl-Namensraum gehört
- Gib den Rest ins Ergebnisdokument aus.
- Daten aus dem Quelldokument ins Zieldokument übernehmen:

```
<xsl:value-of select="xpath-Ausdruck"/>
```

Erinnerung: wie arbeitet der XSLT-Prozessor?

- Führe alle Anweisungen aus, deren Markup zum xsl-Namensraum gehört
- Gib den Rest ins Ergebnisdokument aus.
- Daten aus dem Quelldokument ins Zieldokument übernehmen:

```
<xsl:value-of select="xpath-Ausdruck"/>
```

Das ”/spielplan”-Template

```
<xsl:template match="/spielplan">
   <html>
      <head><title>Eine erste xslt-Uebung</title></head>
      <body>
         <h1>Spielplan des Klassischen Theaters</h1>
         <table border="1">
            <tr>
               <th>Titel</th>
               <th>Autor</th>
               <th>Premiere</th>
            </tr>

            <xsl:apply-templates/>

         </table>
      </body>
   </html>
</xsl:template>
```

Die restlichen Templates

```
<xsl:template match="eintrag">
   <tr>
      <xsl:apply-templates/>
   </tr>
</xsl:template>

<xsl:template match="titel | autor | premiere">
   <td><xsl:value-of select="./text()"/></td>
</xsl:template>
```

Inhalt

Eine Web-Applikation als komfortable Testumgebung

- Um eine xsl-Datei zu testen, muss man sie zusammen mit der zugehörigen xml-Datei mit einem xsl-Prozessor wie *saxon* oder *xalan* aufrufen. Der Ausgabestrom des xsl-Prozessors kann dann wiederum z.B. mit einem html-Browser angezeigt werden.
- Um diesen Vorgang zu automatisieren und das Transformationsergebnis direkt in einem Browser anzeigen zu können, wurde eine kleine Java-Webanwendung erstellt.
- Benötigte Programme: Netbeans IDE mit Tomcat (Java EE - Download), html-Browser

Eine Web-Applikation als komfortable Testumgebung

- Um eine xsl-Datei zu testen, muss man sie zusammen mit der zugehörigen xml-Datei mit einem xsl-Prozessor wie *saxon* oder *xalan* aufrufen. Der Ausgabestrom des xsl-Prozessors kann dann wiederum z.B. mit einem html-Browser angezeigt werden.
- Um diesen Vorgang zu automatisieren und das Transformationsergebnis direkt in einem Browser anzeigen zu können, wurde eine kleine Java-Webanwendung erstellt.
- Benötigte Programme: Netbeans IDE mit Tomcat (Java EE - Download), html-Browser

Eine Web-Applikation als komfortable Testumgebung

- Um eine xsl-Datei zu testen, muss man sie zusammen mit der zugehörigen xml-Datei mit einem xsl-Prozessor wie *saxon* oder *xalan* aufrufen. Der Ausgabestrom des xsl-Prozessors kann dann wiederum z.B. mit einem html-Browser angezeigt werden.
- Um diesen Vorgang zu automatisieren und das Transformationsergebnis direkt in einem Browser anzeigen zu können, wurde eine kleine Java-Webanwendung erstellt.
- Benötigte Programme: Netbeans IDE mit Tomcat (Java EE - Download), html-Browser

Eine Web-Applikation als komfortable Testumgebung

- Um eine xsl-Datei zu testen, muss man sie zusammen mit der zugehörigen xml-Datei mit einem xsl-Prozessor wie *saxon* oder *xalan* aufrufen. Der Ausgabestrom des xsl-Prozessors kann dann wiederum z.B. mit einem html-Browser angezeigt werden.
- Um diesen Vorgang zu automatisieren und das Transformationsergebnis direkt in einem Browser anzeigen zu können, wurde eine kleine Java-Webanwendung erstellt.
- Benötigte Programme: Netbeans IDE mit Tomcat (Java EE - Download), html-Browser

Eine Web-Applikation als komfortable Testumgebung

- Anleitung:
- Laden Sie foldende ZIP-Datei herunter:
  `http://dt.wara.de/xml/xsltester.zip`
- Starten Sie `NetBeans` und importieren Sie die heruntergeladene Datei (File-Menue → Import Project → From ZIP)
- Starten Sie die Anwendung
- Erstellen Sie das gewünschte Stylesheet in der noch unvollständigen Datei `spielplan.xsl` (zum Editieren mit Doppelklick öffnen):

Eine Web-Applikation als komfortable Testumgebung

- Anleitung:
- Laden Sie foldende ZIP-Datei herunter: `http://dt.wara.de/xml/xsltester.zip`
- Starten Sie `NetBeans` und importieren Sie die heruntergeladene Datei (File-Menue → Import Project → From ZIP)
- Starten Sie die Anwendung
- Erstellen Sie das gewünschte Stylesheet in der noch unvollständigen Datei `spielplan.xsl` (zum Editieren mit Doppelklick öffnen):

Eine Web-Applikation als komfortable Testumgebung

- Anleitung:
- Laden Sie foldende ZIP-Datei herunter:
  `http://dt.wara.de/xml/xsltester.zip`
- Starten Sie `NetBeans` und importieren Sie die heruntergeladene Datei (File-Menue → Import Project → From ZIP)
- Starten Sie die Anwendung
- Erstellen Sie das gewünschte Stylesheet in der noch unvollständigen Datei `spielplan.xsl` (zum Editieren mit Doppelklick öffnen):

Eine Web-Applikation als komfortable Testumgebung

- Anleitung:
- Laden Sie foldende ZIP-Datei herunter:
  `http://dt.wara.de/xml/xsltester.zip`
- Starten Sie `NetBeans` und importieren Sie die heruntergeladene Datei (File-Menue → Import Project → From ZIP)
- Starten Sie die Anwendung
- Erstellen Sie das gewünschte Stylesheet in der noch unvollständigen Datei `spielplan.xsl` (zum Editieren mit Doppelklick öffnen):

Eine Web-Applikation als komfortable Testumgebung

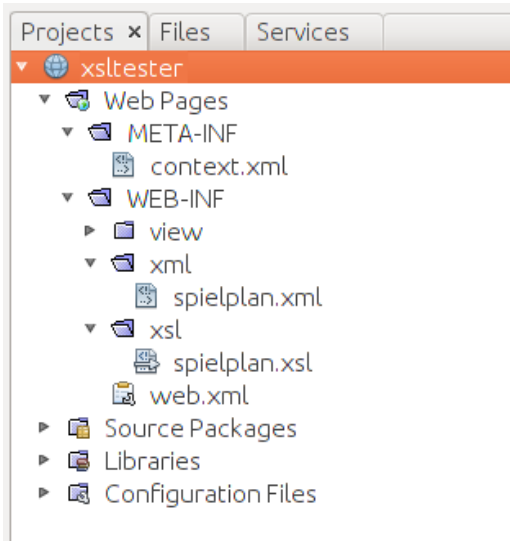
- Anleitung:
- Laden Sie foldende ZIP-Datei herunter:
  `http://dt.wara.de/xml/xsltester.zip`
- Starten Sie `NetBeans` und importieren Sie die heruntergeladene Datei (File-Menue → Import Project → From ZIP)
- Starten Sie die Anwendung
- Erstellen Sie das gewünschte Stylesheet in der noch unvollständigen Datei `spielplan.xsl` (zum Editieren mit Doppelklick öffnen):

Eine Web-Applikation als komfortable Testumgebung

- Anleitung:
- Laden Sie foldende ZIP-Datei herunter:
  `http://dt.wara.de/xml/xsltester.zip`
- Starten Sie `NetBeans` und importieren Sie die heruntergeladene Datei (File-Menue $\rightarrow$ Import Project $\rightarrow$ From ZIP)
- Starten Sie die Anwendung
- Erstellen Sie das gewünschte Stylesheet in der noch unvollständigen Datei `spielplan.xsl` (zum Editieren mit Doppelklick öffnen):

Eine Web-Applikation als komfortable Testumgebung

- Anleitung:
- Laden Sie foldende ZIP-Datei herunter:
  `http://dt.wara.de/xml/xsltester.zip`
- Starten Sie `NetBeans` und importieren Sie die heruntergeladene Datei (File-Menue → Import Project → From ZIP)
- Starten Sie die Anwendung
- Erstellen Sie das gewünschte Stylesheet in der noch unvollständigen Datei `spielplan.xsl` (zum Editieren mit Doppelklick öffnen):



Wie funktioniert die Web-Applikation

- Java-Web-Applikationen verwenden einen in Java geschriebenen Server, der direkt Java-Code ausführen kann und der das http-Protokoll implementiert.
- Bekannteste Java-Webserver:
  - Tomcat (apache): unterstützt Servlets und JSP-Standards, Vorteil: leichtgewichtig
  - Glassfish (Sun/Oracle): fast komplette Unterstützung des Java-EE-Standards, Nachteil: sehr resourcenhungrig
- Als xsl-Prozessor werden Klassen aus der Java-Bibliothek `javax.xml.transform` verwendet. Hiermit ist eine xsl-Transformation direkt in der Web-Applikation möglich. Das Ergebnis wird im Browser angezeigt.

Wie funktioniert die Web-Applikation

- Java-Web-Applikationen verwenden einen in Java geschriebenen Server, der direkt Java-Code ausführen kann und der das http-Protokoll implementiert.
- Bekannteste Java-Webserver:
  - Tomcat (apache): unterstützt Servlets und JSP-Standards, Vorteil: leichtgewichtig
  - Glassfish (Sun/Oracle): fast komplette Unterstützung des Java-EE-Standards, Nachteil: sehr resourcenhungrig
- Als xsl-Prozessor werden Klassen aus der Java-Bibliothek `javax.xml.transform` verwendet. Hiermit ist eine xsl-Transformation direkt in der Web-Applikation möglich. Das Ergebnis wird im Browser angezeigt.

Wie funktioniert die Web-Applikation

- Java-Web-Applikationen verwenden einen in Java geschriebenen Server, der direkt Java-Code ausführen kann und der das http-Protokoll implementiert.
- Bekannteste Java-Webserver:
  - Tomcat (apache): unterstützt Servlets und JSP-Standards, Vorteil: leichtgewichtig
  - Glassfish (Sun/Oracle): fast komplette Unterstützung des Java-EE-Standards, Nachteil: sehr resourcenhungrig
- Als xsl-Prozessor werden Klassen aus der Java-Bibliothek `javax.xml.transform` verwendet. Hiermit ist eine xsl-Transformation direkt in der Web-Applikation möglich. Das Ergebnis wird im Browser angezeigt.

Wie funktioniert die Web-Applikation

- Java-Web-Applikationen verwenden einen in Java geschriebenen Server, der direkt Java-Code ausführen kann und der das http-Protokoll implementiert.
- Bekannteste Java-Webserver:
  - Tomcat (apache): unterstützt Servlets und JSP-Standards, Vorteil: leichtgewichtig
  - Glassfish (Sun/Oracle): fast komplette Unterstützung des Java-EE-Standards, Nachteil: sehr resourcenhungrig
- Als xsl-Prozessor werden Klassen aus der Java-Bibliothek `javax.xml.transform` verwendet. Hiermit ist eine xsl-Transformation direkt in der Web-Applikation möglich. Das Ergebnis wird im Browser angezeigt.

Wie funktioniert die Web-Applikation

- Java-Web-Applikationen verwenden einen in Java geschriebenen Server, der direkt Java-Code ausführen kann und der das http-Protokoll implementiert.
- Bekannteste Java-Webserver:
  - Tomcat (apache): unterstützt Servlets und JSP-Standards, Vorteil: leichtgewichtig
  - Glassfish (Sun/Oracle): fast komplette Unterstützung des Java-EE-Standards, Nachteil: sehr resourcenhungrig
- Als xsl-Prozessor werden Klassen aus der Java-Bibliothek `javax.xml.transform` verwendet. Hiermit ist eine xsl-Transformation direkt in der Web-Applikation möglich. Das Ergebnis wird im Browser angezeigt.

Wie funktioniert die Web-Applikation

- Java-Web-Applikationen verwenden einen in Java geschriebenen Server, der direkt Java-Code ausführen kann und der das http-Protokoll implementiert.
- Bekannteste Java-Webserver:
  - Tomcat (apache): unterstützt Servlets und JSP-Standards, Vorteil: leichtgewichtig
  - Glassfish (Sun/Oracle): fast komplette Unterstützung des Java-EE-Standards, Nachteil: sehr resourcenhungrig
- Als xsl-Prozessor werden Klassen aus der Java-Bibliothek `javax.xml.transform` verwendet. Hiermit ist eine xsl-Transformation direkt in der Web-Applikation möglich. Das Ergebnis wird im Browser angezeigt.

Übungen 1

- Erstellen Sie ein xsl-Stylesheet, das die Theaterspielplan-xml-Datei in eine html-Tabelle überführt.
- Je nach Arbeitstempo kann die Tabelle zunehmend aufwändig gestaltet und sortiert werden:

Übungen 1

- Erstellen Sie ein xsl-Stylesheet, das die Theaterspielplan-xml-Datei in eine html-Tabelle überführt.
- Je nach Arbeitstempo kann die Tabelle zunehmend aufwändig gestaltet und sortiert werden:

Übungen 1

- Erstellen Sie ein xsl-Stylesheet, das die Theaterspielplan-xml-Datei in eine html-Tabelle überführt.
- Je nach Arbeitstempo kann die Tabelle zunehmend aufwändig gestaltet und sortiert werden:

Übungen 1

- Erstellen Sie ein xsl-Stylesheet, das die Theaterspielplan-xml-Datei in eine html-Tabelle überführt.
- Je nach Arbeitstempo kann die Tabelle zunehmend aufwändig gestaltet und sortiert werden:

XSLT macht Spass!

Titel	Autor	Premiere
Die Räuber	Friedrich Schiller	2014-10-11
Faust	Johann Wolfgang Goethe	2014-11-12
Der zerbrochene Krug	Heinrich von Kleist	2014-12-13
Tod eines Handlungsreisenden	Henry Miller	2015-02-02
Der Watzmann	Ambros/Tauchen/Prokopetz	2014-11-22
Der Vorname	Delaporte/Patellière	2014-12-13

Übungen 1

- Erstellen Sie ein xsl-Stylesheet, das die Theaterspielplan-xml-Datei in eine html-Tabelle überführt.
- Je nach Arbeitstempo kann die Tabelle zunehmend aufwändig gestaltet und sortiert werden:

XSLT macht Spass!

Titel	Autor	Premiere
Die Räuber	Friedrich Schiller	2014-10-11
Faust	Johann Wolfgang Goethe	2014-11-12
Der zerbrochene Krug	Heinrich von Kleist	2014-12-13
Tod eines Handlungsreisenden	Henry Miller	2015-02-02
Der Watzmann	Ambros/Tauchen/Prokopetz	2014-11-22
Der Vorname	Delaporte/Patellière	2014-12-13

XSLT macht Spass!

Nr.	Titel	Autor	Premiere
1	Die Räuber	Friedrich Schiller	2014-10-11
2	Faust	Johann Wolfgang Goethe	2014-11-12
3	Der zerbrochene Krug	Heinrich von Kleist	2014-12-13
4	Tod eines Handlungsreisenden	Henry Miller	2015-02-02
5	Der Watzmann	Ambros/Tauchen/Prokopetz	2014-11-22
6	Der Vorname	Delaporte/Patellière	2014-12-13

Hinweise zu den Übungen

- Und schliesslich mit Sortierung:

- die fortlaufende Nummer wird mit der Funktion position() erzeugt. position() enthält die Nummer des gegenwärtigen Knotens.
- für die Hintergrundfarben der Zeilen benötigen Sie:

Hinweise zu den Übungen

- Und schliesslich mit Sortierung:

- die fortlaufende Nummer wird mit der Funktion position() erzeugt. position() enthält die Nummer des gegenwärtigen Knotens.
- für die Hintergrundfarben der Zeilen benötigen Sie:

Hinweise zu den Übungen

- Und schliesslich mit Sortierung:

XSLT macht Spass!

Nr.	Titel	Autor	Premiere
1	Die Räuber	Friedrich Schiller	2014-10-11
2	Faust	Johann Wolfgang Goethe	2014-11-12
3	Der Watzmann	Ambros/Tauchen/Prokopetz	2014-11-22
4	Der Vorname	Delaporte/Patellière	2014-12-13
5	Der zerbrochene Krug	Heinrich von Kleist	2014-12-13
6	Tod eines Handlungsreisenden	Henry Miller	2015-02-02

- die fortlaufende Nummer wird mit der Funktion position() erzeugt. position() enthält die Nummer des gegenwärtigen Knotens.
- für die Hintergrundfarben der Zeilen benötigen Sie:

Hinweise zu den Übungen

- Und schliesslich mit Sortierung:

XSLT macht Spass!

Nr.	Titel	Autor	Premiere
1	Die Räuber	Friedrich Schiller	2014-10-11
2	Faust	Johann Wolfgang Goethe	2014-11-12
3	Der Watzmann	Ambros/Tauchen/Prokopetz	2014-11-22
4	Der Vorname	Delaporte/Patellière	2014-12-13
5	Der zerbrochene Krug	Heinrich von Kleist	2014-12-13
6	Tod eines Handlungsreisenden	Henry Miller	2015-02-02

- die fortlaufende Nummer wird mit der Funktion position() erzeugt. position() enthält die Nummer des gegenwärtigen Knotens.
- für die Hintergrundfarben der Zeilen benötigen Sie:

Hinweise zu den Übungen

- Und schliesslich mit Sortierung:

XSLT macht Spass!

Nr.	Titel	Autor	Premiere
1	Die Räuber	Friedrich Schiller	2014-10-11
2	Faust	Johann Wolfgang Goethe	2014-11-12
3	Der Watzmann	Ambros/Tauchen/Prokopetz	2014-11-22
4	Der Vorname	Delaporte/Patellière	2014-12-13
5	Der zerbrochene Krug	Heinrich von Kleist	2014-12-13
6	Tod eines Handlungsreisenden	Henry Miller	2015-02-02

- die fortlaufende Nummer wird mit der Funktion position() erzeugt. position() enthält die Nummer des gegenwärtigen Knotens.
- für die Hintergrundfarben der Zeilen benötigen Sie:

Hinweise zu den Übungen

xsl:variable : Erzeugen Sie eine Variable mit Namen `"farbe"` und weisen Sie dieser die Hintergrundfarbe zu.

```
http://www.w3schools.com/xsl/el_variable.asp
```

css-style : Das Attribut `style` wird verwendet, um die Hintergrundfarbe einer html-Tabellenzeile zu setzen:

```
<tr style="background-color:{$farbe};">
```

Hinweise zu den Übungen

xsl:variable : Erzeugen Sie eine Variable mit Namen `"farbe"` und weisen Sie dieser die Hintergrundfarbe zu.

```
http://www.w3schools.com/xsl/el_variable.asp
```

css-style : Das Attribut `style` wird verwendet, um die Hintergrundfarbe einer html-Tabellenzeile zu setzen:

```
<tr style="background-color:{$farbe};">
```

Hinweise zu den Übungen

xsl:variable : Erzeugen Sie eine Variable mit Namen `"farbe"` und weisen Sie dieser die Hintergrundfarbe zu.

```
http://www.w3schools.com/xsl/el_variable.asp
```

css-style : Das Attribut `style` wird verwendet, um die Hintergrundfarbe einer html-Tabellenzeile zu setzen:

```
<tr style="background-color:{$farbe};">
```

Hinweise zu den Übungen

xsl:variable : Erzeugen Sie eine Variable mit Namen `"farbe"` und weisen Sie dieser die Hintergrundfarbe zu.

```
http://www.w3schools.com/xsl/el_variable.asp
```

css-style : Das Attribut `style` wird verwendet, um die Hintergrundfarbe einer html-Tabellenzeile zu setzen:

```
<tr style="background-color:{$farbe};">
```

Hinweise zu den Übungen

xsl:variable : Erzeugen Sie eine Variable mit Namen `"farbe"` und weisen Sie dieser die Hintergrundfarbe zu.

```
http://www.w3schools.com/xsl/el_variable.asp
```

css-style : Das Attribut `style` wird verwendet, um die Hintergrundfarbe einer html-Tabellenzeile zu setzen:

```
<tr style="background-color:{$farbe};">
```

Hinweise zu den Übungen

xsl:choose : Unterschiedliche Werte für die Farbvariable wählen (gerade/ungerade Zeilennummern);

```
http://www.w3schools.com/xsl/xsl_choose.asp
```

Zu Vergleichs- und Modulo-Operatoren siehe hier:

```
http://www.w3schools.com/xpath/xpath_operators.asp
```

xsl:sort : das <xsl:sort>-Element darf nur als *erstes* Kindelement innerhalb des <xsl:apply-templates>- oder des <xsl:for-each>-Elements (siehe unten) stehen. Genaueres hier:

```
http://www.w3schools.com/xsl/el_sort.asp
```

xsl:if : Der Vollständigkeit halber:

```
http://www.w3schools.com/xsl/xsl_if.asp
```

Hinweise zu den Übungen

xsl:choose : Unterschiedliche Werte für die Farbvariable wählen (gerade/ungerade Zeilennummern);

```
http://www.w3schools.com/xsl/xsl_choose.asp
```

Zu Vergleichs- und Modulo-Operatoren siehe hier:

```
http://www.w3schools.com/xpath/xpath_operators.asp
```

xsl:sort : das <xsl:sort>-Element darf nur als *erstes* Kindelement innerhalb des <xsl:apply-templates>- oder des <xsl:for-each>-Elements (siehe unten) stehen. Genaueres hier:

```
http://www.w3schools.com/xsl/el_sort.asp
```

xsl:if : Der Vollständigkeit halber:

```
http://www.w3schools.com/xsl/xsl_if.asp
```

Hinweise zu den Übungen

xsl:choose : Unterschiedliche Werte für die Farbvariable wählen (gerade/ungerade Zeilennummern);

```
http://www.w3schools.com/xsl/xsl_choose.asp
```

Zu Vergleichs- und Modulo-Operatoren siehe hier:

```
http://www.w3schools.com/xpath/xpath_operators.asp
```

xsl:sort : das <xsl:sort>-Element darf nur als *erstes* Kindelement innerhalb des <xsl:apply-templates>- oder des <xsl:for-each>-Elements (siehe unten) stehen. Genaueres hier:

```
http://www.w3schools.com/xsl/el_sort.asp
```

xsl:if : Der Vollständigkeit halber:

```
http://www.w3schools.com/xsl/xsl_if.asp
```

Hinweise zu den Übungen

xsl:choose : Unterschiedliche Werte für die Farbvariable wählen (gerade/ungerade Zeilennummern);

```
http://www.w3schools.com/xsl/xsl_choose.asp
```

Zu Vergleichs- und Modulo-Operatoren siehe hier:

```
http://www.w3schools.com/xpath/xpath_operators.asp
```

xsl:sort : das <xsl:sort>-Element darf nur als *erstes* Kindelement innerhalb des <xsl:apply-templates>- oder des <xsl:for-each>-Elements (siehe unten) stehen. Genaueres hier:

```
http://www.w3schools.com/xsl/el_sort.asp
```

xsl:if : Der Vollständigkeit halber:

```
http://www.w3schools.com/xsl/xsl_if.asp
```

Hinweise zu den Übungen

xsl:choose : Unterschiedliche Werte für die Farbvariable wählen (gerade/ungerade Zeilennummern);

```
http://www.w3schools.com/xsl/xsl_choose.asp
```

Zu Vergleichs- und Modulo-Operatoren siehe hier:

```
http://www.w3schools.com/xpath/xpath_operators.asp
```

xsl:sort : das <xsl:sort>-Element darf nur als *erstes* Kindelement innerhalb des <xsl:apply-templates>- oder des <xsl:for-each>-Elements (siehe unten) stehen. Genaueres hier:

```
http://www.w3schools.com/xsl/el_sort.asp
```

xsl:if : Der Vollständigkeit halber:

```
http://www.w3schools.com/xsl/xsl_if.asp
```

Hinweise zu den Übungen

xsl:choose : Unterschiedliche Werte für die Farbvariable wählen (gerade/ungerade Zeilennummern);

```
http://www.w3schools.com/xsl/xsl_choose.asp
```

Zu Vergleichs- und Modulo-Operatoren siehe hier:

```
http://www.w3schools.com/xpath/xpath_operators.asp
```

xsl:sort : das <xsl:sort>-Element darf nur als *erstes* Kindelement innerhalb des <xsl:apply-templates>- oder des <xsl:for-each>-Elements (siehe unten) stehen. Genaueres hier:

```
http://www.w3schools.com/xsl/el_sort.asp
```

xsl:if : Der Vollständigkeit halber:

```
http://www.w3schools.com/xsl/xsl_if.asp
```

Hinweise zu den Übungen

xsl:choose : Unterschiedliche Werte für die Farbvariable wählen (gerade/ungerade Zeilennummern);

```
http://www.w3schools.com/xsl/xsl_choose.asp
```

Zu Vergleichs- und Modulo-Operatoren siehe hier:

```
http://www.w3schools.com/xpath/xpath_operators.asp
```

xsl:sort : das <xsl:sort>-Element darf nur als *erstes* Kindelement innerhalb des <xsl:apply-templates>- oder des <xsl:for-each>-Elements (siehe unten) stehen. Genaueres hier:

```
http://www.w3schools.com/xsl/el_sort.asp
```

xsl:if : Der Vollständigkeit halber:

```
http://www.w3schools.com/xsl/xsl_if.asp
```

Inhalt

Nachtrag: Eine Alternative zu <xsl:apply-templates>

- XSLT-Schleifen:

```
<xsl: for-each select="xpath-Ausdruck"
```

- der select-Ausdruck wählt eine Knotenmenge aus
- über die Knotenmenge wird mit for-each iteriert
- $\Rightarrow$ `<xsl:for-each>` bildet eine Alternative zu `<xsl:apply-templates>`
- man *zieht* die ausgewählten Knoten in ein Template hinein
- man nennt diesen Ansatz: *Pull Processing*

Nachtrag: Eine Alternative zu <xsl:apply-templates>

- XSLT-Schleifen:

```
<xsl: for-each select="xpath-Ausdruck"
```

- der select-Ausdruck wählt eine Knotenmenge aus
- über die Knotenmenge wird mit for-each iteriert
- $\Rightarrow$ `<xsl:for-each>` bildet eine Alternative zu `<xsl:apply-templates>`
- man *zieht* die ausgewählten Knoten in ein Template hinein
- man nennt diesen Ansatz: *Pull Processing*

Nachtrag: Eine Alternative zu <xsl:apply-templates>

- XSLT-Schleifen:

```
<xsl: for-each select="xpath-Ausdruck"
```

- der select-Ausdruck wählt eine Knotenmenge aus
- über die Knotenmenge wird mit for-each iteriert
- $\Rightarrow$ `<xsl:for-each>` bildet eine Alternative zu `<xsl:apply-templates>`
- man *zieht* die ausgewählten Knoten in ein Template hinein
- man nennt diesen Ansatz: *Pull Processing*

Nachtrag: Eine Alternative zu <xsl:apply-templates>

- XSLT-Schleifen:

```
<xsl: for-each select="xpath-Ausdruck"
```

- der select-Ausdruck wählt eine Knotenmenge aus
- über die Knotenmenge wird mit for-each iteriert
- $\Rightarrow$ `<xsl:for-each>` bildet eine Alternative zu `<xsl:apply-templates>`
- man *zieht* die ausgewählten Knoten in ein Template hinein
- man nennt diesen Ansatz: *Pull Processing*

Nachtrag: Eine Alternative zu <xsl:apply-templates>

- XSLT-Schleifen:

```
<xsl: for-each select="xpath-Ausdruck"
```

- der select-Ausdruck wählt eine Knotenmenge aus
- über die Knotenmenge wird mit for-each iteriert
- $\Rightarrow$ `<xsl:for-each>` bildet eine Alternative zu `<xsl:apply-templates>`
- man *zieht* die ausgewählten Knoten in ein Template hinein
- man nennt diesen Ansatz: *Pull Processing*

Nachtrag: Eine Alternative zu <xsl:apply-templates>

- XSLT-Schleifen:

```
<xsl: for-each select="xpath-Ausdruck"
```

- der select-Ausdruck wählt eine Knotenmenge aus
- über die Knotenmenge wird mit for-each iteriert
- $\Rightarrow$ `<xsl:for-each>` bildet eine Alternative zu `<xsl:apply-templates>`
- man *zieht* die ausgewählten Knoten in ein Template hinein
- man nennt diesen Ansatz: *Pull Processing*

Nachtrag: Eine Alternative zu <xsl:apply-templates>

- XSLT-Schleifen:

```
<xsl: for-each select="xpath-Ausdruck"
```

- der select-Ausdruck wählt eine Knotenmenge aus
- über die Knotenmenge wird mit for-each iteriert
- $\Rightarrow$ `<xsl:for-each>` bildet eine Alternative zu `<xsl:apply-templates>`
- man *zieht* die ausgewählten Knoten in ein Template hinein
- man nennt diesen Ansatz: *Pull Processing*

Nachtrag: Eine Alternative zu <xsl:apply-templates>

- XSLT-Schleifen:

```
<xsl: for-each select="xpath-Ausdruck"
```

- der select-Ausdruck wählt eine Knotenmenge aus
- über die Knotenmenge wird mit for-each iteriert
- $\Rightarrow$ `<xsl:for-each>` bildet eine Alternative zu `<xsl:apply-templates>`
- man *zieht* die ausgewählten Knoten in ein Template hinein
- man nennt diesen Ansatz: *Pull Processing*

Ein Beispiel mit <xsl:for-each>

```
<xsl:template match="/spielplan">
   ...
   <xsl:for-each select="eintrag">
      <tr><td>
            <xsl:value-of select="titel"/>
         <td></td>
            <xsl:value-of select="autor"/>
         <td></td>
            <xsl:value-of select="premiere"/>
      </td></tr>
   </xsl:for-each>
   ...
</xsl:template>
```

⇒ Die Knoten `titel`, `autor` und `premiere` werden ins Template von `/spielplan` *hineingezogen*.

Ein Beispiel mit <xsl:for-each>

```
<xsl:template match="/spielplan">
   ...
   <xsl:for-each select="eintrag">
      <tr><td>
            <xsl:value-of select="titel"/>
          <td></td>
            <xsl:value-of select="autor"/>
          <td></td>
            <xsl:value-of select="premiere"/>
      </td></tr>
   </xsl:for-each>
   ...
</xsl:template>
```

$\Rightarrow$ Die Knoten `titel`, `autor` und `premiere` werden ins Template von `/spielplan` *hineingezogen*.

Ein Beispiel mit <xsl:for-each>

```
<xsl:template match="/spielplan">
   ...
   <xsl:for-each select="eintrag">
      <tr><td>
            <xsl:value-of select="titel"/>
          <td></td>
            <xsl:value-of select="autor"/>
          <td></td>
            <xsl:value-of select="premiere"/>
      </td></tr>
   </xsl:for-each>
   ...
</xsl:template>
```

⇒ Die Knoten `titel`, `autor` und `premiere` werden ins Template von `/spielplan` *hineingezogen*.

Erzeugen von Attributen und Elementen

- Problem: wir möchten html-Elemente mit Attributen erzeugen. Die Werte der Attribute sollen aus der xml-Datei stammen.
- Beispiel-Daten (xml): die Spielplan-Datei wird um `<szenenfotos>`-Elemente erweitert:

```
        ...
<eintrag>
   <titel>Die Räuber</titel>
   <autor>Friedrich Schiller</autor>
   <premiere>2014-10-11</premiere>
   <szenenfotos>raueber.jpg</szenenfotos>
</eintrag>
        ...
```

Erzeugen von Attributen und Elementen

- Problem: wir möchten html-Elemente mit Attributen erzeugen. Die Werte der Attribute sollen aus der xml-Datei stammen.
- Beispiel-Daten (xml): die Spielplan-Datei wird um `<szenenfotos>`-Elemente erweitert:

```
        ...
<eintrag>
   <titel>Die Räuber</titel>
   <autor>Friedrich Schiller</autor>
   <premiere>2014-10-11</premiere>
   <szenenfotos>raueber.jpg</szenenfotos>
</eintrag>
        ...
```

Erzeugen von Attributen und Elementen

- Problem: wir möchten html-Elemente mit Attributen erzeugen. Die Werte der Attribute sollen aus der xml-Datei stammen.
- Beispiel-Daten (xml): die Spielplan-Datei wird um `<szenenfotos>`-Elemente erweitert:

```
        ...
<eintrag>
   <titel>Die Räuber</titel>
   <autor>Friedrich Schiller</autor>
   <premiere>2014-10-11</premiere>
   <szenenfotos>raueber.jpg</szenenfotos>
</eintrag>
        ...
```

Erzeugen von Attributen und Elementen

- Problem: wir möchten html-Elemente mit Attributen erzeugen. Die Werte der Attribute sollen aus der xml-Datei stammen.
- Beispiel-Daten (xml): die Spielplan-Datei wird um `<szenenfotos>`-Elemente erweitert:

```
        ...
<eintrag>
   <titel>Die Räuber</titel>
   <autor>Friedrich Schiller</autor>
   <premiere>2014-10-11</premiere>
   <szenenfotos>raueber.jpg</szenenfotos>
</eintrag>
        ...
```

2 Möglichkeiten

1. **Attribute Value Templates**

```
<a href="{xpath-Ausdruck}"> Linkname </a>
```

Die geschweifen Klammern weisen den XSLT-Prozessor an, den enthaltenen xpath-Ausdruck auszuwerten und das Ergebnis dort in den Ausgabestrom zu schreiben.

2. `<xsl:attribute>`

```
<xsl:attribute name="href"><xsl:value-of
select="xpath-ausdruck"/> | text></xsl:attribute>
```

- Vorsicht: keine Zeilenumbrüche in die Attributwerte einbauen!
- Vorteil: kann mit Kontrollstrukturen verwendet werden
- Siehe auch:

```
http://www.w3schools.com/xsl/el_attribute.asp
```

2 Möglichkeiten

1. **Attribute Value Templates**

```
<a href="{xpath-Ausdruck}"> Linkname </a>
```

Die geschweifen Klammern weisen den XSLT-Prozessor an, den enthaltenen xpath-Ausdruck auszuwerten und das Ergebnis dort in den Ausgabestrom zu schreiben.

2. `<xsl:attribute>`

```
<xsl:attribute name="href"><xsl:value-of
select="xpath-ausdruck"/> | text></xsl:attribute>
```

- Vorsicht: keine Zeilenumbrüche in die Attributwerte einbauen!
- Vorteil: kann mit Kontrollstrukturen verwendet werden
- Siehe auch:

```
http://www.w3schools.com/xsl/el_attribute.asp
```

2 Möglichkeiten

1. **Attribute Value Templates**

```
<a href="{xpath-Ausdruck}"> Linkname </a>
```

Die geschweifen Klammern weisen den XSLT-Prozessor an, den enthaltenen xpath-Ausdruck auszuwerten und das Ergebnis dort in den Ausgabestrom zu schreiben.

2. `<xsl:attribute>`

```
<xsl:attribute name="href"><xsl:value-of
select="xpath-ausdruck"/> | text></xsl:attribute>
```

- Vorsicht: keine Zeilenumbrüche in die Attributwerte einbauen!
- Vorteil: kann mit Kontrollstrukturen verwendet werden
- Siehe auch:

```
http://www.w3schools.com/xsl/el_attribute.asp
```

2 Möglichkeiten

1. **Attribute Value Templates**

```
<a href="{xpath-Ausdruck}"> Linkname </a>
```

Die geschweifen Klammern weisen den XSLT-Prozessor an, den enthaltenen xpath-Ausdruck auszuwerten und das Ergebnis dort in den Ausgabestrom zu schreiben.

2. `<xsl:attribute>`

```
<xsl:attribute name="href"><xsl:value-of
select="xpath-ausdruck"/> | text></xsl:attribute>
```

- Vorsicht: keine Zeilenumbrüche in die Attributwerte einbauen!
- Vorteil: kann mit Kontrollstrukturen verwendet werden
- Siehe auch:

```
http://www.w3schools.com/xsl/el_attribute.asp
```

2 Möglichkeiten

1. **Attribute Value Templates**

```
<a href="{xpath-Ausdruck}"> Linkname </a>
```

Die geschweifen Klammern weisen den XSLT-Prozessor an, den enthaltenen xpath-Ausdruck auszuwerten und das Ergebnis dort in den Ausgabestrom zu schreiben.

2. `<xsl:attribute>`

```
<xsl:attribute name="href"><xsl:value-of
select="xpath-ausdruck"/> | text></xsl:attribute>
```

- Vorsicht: keine Zeilenumbrüche in die Attributwerte einbauen!
- Vorteil: kann mit Kontrollstrukturen verwendet werden
- Siehe auch:

```
http://www.w3schools.com/xsl/el_attribute.asp
```

2 Möglichkeiten

1. **Attribute Value Templates**

```
<a href="{xpath-Ausdruck}"> Linkname </a>
```

Die geschweifen Klammern weisen den XSLT-Prozessor an, den enthaltenen xpath-Ausdruck auszuwerten und das Ergebnis dort in den Ausgabestrom zu schreiben.

2. `<xsl:attribute>`

```
<xsl:attribute name="href"><xsl:value-of
select="xpath-ausdruck"/> | text></xsl:attribute>
```

- Vorsicht: keine Zeilenumbrüche in die Attributwerte einbauen!
- Vorteil: kann mit Kontrollstrukturen verwendet werden
- Siehe auch:

```
http://www.w3schools.com/xsl/el_attribute.asp
```

2 Möglichkeiten

1. **Attribute Value Templates**

```
<a href="{xpath-Ausdruck}"> Linkname </a>
```

Die geschweifen Klammern weisen den XSLT-Prozessor an, den enthaltenen xpath-Ausdruck auszuwerten und das Ergebnis dort in den Ausgabestrom zu schreiben.

2. `<xsl:attribute>`

```
<xsl:attribute name="href"><xsl:value-of
select="xpath-ausdruck"/> | text></xsl:attribute>
```

- Vorsicht: keine Zeilenumbrüche in die Attributwerte einbauen!
- Vorteil: kann mit Kontrollstrukturen verwendet werden
- Siehe auch:

```
http://www.w3schools.com/xsl/el_attribute.asp
```

2 Möglichkeiten

1. **Attribute Value Templates**

```
<a href="{xpath-Ausdruck}"> Linkname </a>
```

Die geschweifen Klammern weisen den XSLT-Prozessor an, den enthaltenen xpath-Ausdruck auszuwerten und das Ergebnis dort in den Ausgabestrom zu schreiben.

2. `<xsl:attribute>`

```
<xsl:attribute name="href"><xsl:value-of
select="xpath-ausdruck"/> | text></xsl:attribute>
```

- Vorsicht: keine Zeilenumbrüche in die Attributwerte einbauen!
- Vorteil: kann mit Kontrollstrukturen verwendet werden
- Siehe auch:

```
http://www.w3schools.com/xsl/el_attribute.asp
```

2 Möglichkeiten

1. **Attribute Value Templates**

```
<a href="{xpath-Ausdruck}"> Linkname </a>
```

Die geschweifen Klammern weisen den XSLT-Prozessor an, den enthaltenen xpath-Ausdruck auszuwerten und das Ergebnis dort in den Ausgabestrom zu schreiben.

2. `<xsl:attribute>`

```
<xsl:attribute name="href"><xsl:value-of
select="xpath-ausdruck"/> | text></xsl:attribute>
```

- Vorsicht: keine Zeilenumbrüche in die Attributwerte einbauen!
- Vorteil: kann mit Kontrollstrukturen verwendet werden
- Siehe auch:

```
http://www.w3schools.com/xsl/el_attribute.asp
```

Übungen zur Attributerzeugung

1. Erzeugen Sie Links in der html-Tabelle mit `Attribute Value Templates`
2. Erzeugen Sie die Links nur, wenn die xml-Datei die nötigen Dateipfade enthält. Geben Sie andernfalls einen Hinweis aus (mit <xsl:attribute> und <xsl:choose>):

Übungen zur Attributerzeugung

1. Erzeugen Sie Links in der html-Tabelle mit `Attribute Value Templates`
2. Erzeugen Sie die Links nur, wenn die xml-Datei die nötigen Dateipfade enthält. Geben Sie andernfalls einen Hinweis aus (mit <xsl:attribute> und <xsl:choose>):

Übungen zur Attributerzeugung

1. Erzeugen Sie Links in der html-Tabelle mit `Attribute Value Templates`
2. Erzeugen Sie die Links nur, wenn die xml-Datei die nötigen Dateipfade enthält. Geben Sie andernfalls einen Hinweis aus (mit <xsl:attribute> und <xsl:choose>):

Übungen zur Attributerzeugung

1. Erzeugen Sie Links in der html-Tabelle mit `Attribute Value Templates`
2. Erzeugen Sie die Links nur, wenn die xml-Datei die nötigen Dateipfade enthält. Geben Sie andernfalls einen Hinweis aus (mit <xsl:attribute> und <xsl:choose>):

XSLT macht Spass!

Nr.	Titel	Autor	Premiere	Szenenfotos
1	Die Räuber	Friedrich Schiller	2014-10-11	klick: Szenefotos
2	Faust	Johann Wolfgang Goethe	2014-11-12	keine aktuellen Bilder vorhanden
3	Der Watzmann	Ambros/Tauchen/Prokopetz	2014-11-22	keine aktuellen Bilder vorhanden
4	Der Vorname	Delaporte/Patellière	2014-12-13	klick: Szenefotos
5	Der zerbrochene Krug	Heinrich von Kleist	2014-12-13	keine aktuellen Bilder vorhanden
6	Tod eines Handlungsreisenden	Henry Miller	2015-02-02	keine aktuellen Bilder vorhanden

Elemente erzeugen

- Im Ausgabedokument sollen (xml)-Elemente vorkommen, deren Namen erst zur Laufzeit des xsl-Transformationsvorgangs festgelegt werden.
- Erzeugen mit `<xsl:element>` und ggfs `<xsl:attribute-set>` :

```
<xsl:attribute-set name="attrSetName">
   <xsl:attribute name="name">
      attribut-wert
   </xsl:attribute>
</xsl:attribute>

<xsl:element name="elementname"
             use-attribute-sets="attrSetName1, ...">
   ...
</xsl:element>
```

Elemente erzeugen

- Im Ausgabedokument sollen (xml)-Elemente vorkommen, deren Namen erst zur Laufzeit des xsl-Transformationsvorgangs festgelegt werden.
- Erzeugen mit `<xsl:element>` und ggfs `<xsl:attribute-set>` :

```
<xsl:attribute-set name="attrSetName">
   <xsl:attribute name="name">
      attribut-wert
   </xsl:attribute>
</xsl:attribute>

<xsl:element name="elementname"
             use-attribute-sets="attrSetName1, ...">
   ...
</xsl:element>
```

Elemente erzeugen

- Im Ausgabedokument sollen (xml)-Elemente vorkommen, deren Namen erst zur Laufzeit des xsl-Transformationsvorgangs festgelegt werden.
- Erzeugen mit `<xsl:element>` und ggfs `<xsl:attribute-set>`:

```
<xsl:attribute-set name="attrSetName">
   <xsl:attribute name="name">
      attribut-wert
   </xsl:attribute>
</xsl:attribute>

<xsl:element name="elementname"
             use-attribute-sets="attrSetName1, ...">
   ...
</xsl:element>
```

Elemente erzeugen

- Im Ausgabedokument sollen (xml)-Elemente vorkommen, deren Namen erst zur Laufzeit des xsl-Transformationsvorgangs festgelegt werden.
- Erzeugen mit `<xsl:element>` und ggfs `<xsl:attribute-set>`:

```
<xsl:attribute-set name="attrSetName">
   <xsl:attribute name="name">
      attribut-wert
   </xsl:attribute>
</xsl:attribute>

<xsl:element name="elementname"
             use-attribute-sets="attrSetName1, ...">
   ...
</xsl:element>
```

Elemente erzeugen

- Im Ausgabedokument sollen (xml)-Elemente vorkommen, deren Namen erst zur Laufzeit des xsl-Transformationsvorgangs festgelegt werden.
- Erzeugen mit `<xsl:element>` und ggfs `<xsl:attribute-set>` :

```
<xsl:attribute-set name="attrSetName">
   <xsl:attribute name="name">
      attribut-wert
   </xsl:attribute>
</xsl:attribute>

<xsl:element name="elementname"
             use-attribute-sets="attrSetName1, ...">
   ...
</xsl:element>
```

Elemente erzeugen

- Im Ausgabedokument sollen (xml)-Elemente vorkommen, deren Namen erst zur Laufzeit des xsl-Transformationsvorgangs festgelegt werden.
- Erzeugen mit `<xsl:element>` und ggfs `<xsl:attribute-set>` :

```
<xsl:attribute-set name="attrSetName">
   <xsl:attribute name="name">
      attribut-wert
   </xsl:attribute>
</xsl:attribute>

<xsl:element name="elementname"
             use-attribute-sets="attrSetName1, ...">
   ...
</xsl:element>
```

Elemente erzeugen

- Anwendung 1: Erzeugen von xsl-Dateien mit xsl-Transformationen
- Anwendung 2: Verwendung von Attribute-Sets
- Übung (optional): die Tabellenzeilen abwechselnd einfärben. Dabei `<xsl:attribute-set>` für die Farben und `<xsl:element>` für die Table-Row (tr) verwenden.

Elemente erzeugen

- Anwendung 1: Erzeugen von xsl-Dateien mit xsl-Transformationen
- Anwendung 2: Verwendung von Attribute-Sets
- Übung (optional): die Tabellenzeilen abwechselnd einfärben. Dabei `<xsl:attribute-set>` für die Farben und `<xsl:element>` für die Table-Row (tr) verwenden.

Elemente erzeugen

- Anwendung 1: Erzeugen von xsl-Dateien mit xsl-Transformationen
- Anwendung 2: Verwendung von Attribute-Sets
- Übung (optional): die Tabellenzeilen abwechselnd einfärben. Dabei `<xsl:attribute-set>` für die Farben und `<xsl:element>` für die Table-Row (tr) verwenden.

Elemente erzeugen

- Anwendung 1: Erzeugen von xsl-Dateien mit xsl-Transformationen
- Anwendung 2: Verwendung von Attribute-Sets
- Übung (optional): die Tabellenzeilen abwechselnd einfärben. Dabei `<xsl:attribute-set>` für die Farben und `<xsl:element>` für die Table-Row (tr) verwenden.

Inhalt

Funktionen

- Template-Rules können das Attribut *name* erhalten und können dann über den Namen aufgerufen werden.
  $\Rightarrow$ Template-Rules entsprechen somit Funktionen, Methoden.
- Template-Rules können auch Parameter erhalten
- Entsprechend kann eine Template-Rule mit Parametern aufgerufen werden
- Syntax und Verwendung folgen in einem Beispiel

Funktionen

- Template-Rules können das Attribut *name* erhalten und können dann über den Namen aufgerufen werden.
  $\Rightarrow$ Template-Rules entsprechen somit Funktionen, Methoden.
- Template-Rules können auch Parameter erhalten
- Entsprechend kann eine Template-Rule mit Parametern aufgerufen werden
- Syntax und Verwendung folgen in einem Beispiel

Funktionen

- Template-Rules können das Attribut *name* erhalten und können dann über den Namen aufgerufen werden.
  $\Rightarrow$ Template-Rules entsprechen somit Funktionen, Methoden.
- Template-Rules können auch Parameter erhalten
- Entsprechend kann eine Template-Rule mit Parametern aufgerufen werden
- Syntax und Verwendung folgen in einem Beispiel

Funktionen

- Template-Rules können das Attribut *name* erhalten und können dann über den Namen aufgerufen werden.
  $\Rightarrow$ Template-Rules entsprechen somit Funktionen, Methoden.
- Template-Rules können auch Parameter erhalten
- Entsprechend kann eine Template-Rule mit Parametern aufgerufen werden
- Syntax und Verwendung folgen in einem Beispiel

Funktionen

- Template-Rules können das Attribut *name* erhalten und können dann über den Namen aufgerufen werden.
  $\Rightarrow$ Template-Rules entsprechen somit Funktionen, Methoden.
- Template-Rules können auch Parameter erhalten
- Entsprechend kann eine Template-Rule mit Parametern aufgerufen werden
- Syntax und Verwendung folgen in einem Beispiel

Funktionen

- Template-Rules können das Attribut *name* erhalten und können dann über den Namen aufgerufen werden.
  $\Rightarrow$ Template-Rules entsprechen somit Funktionen, Methoden.
- Template-Rules können auch Parameter erhalten
- Entsprechend kann eine Template-Rule mit Parametern aufgerufen werden
- Syntax und Verwendung folgen in einem Beispiel

Variablen

- Definition einer Variablen, Variante1:

```
<xsl:variable name="name" select="ausdruck" />
```

- Definition, Variante2:

```
<xsl:variable name="name">
    anweisungen
</xsl:variable>
```

- Geltungsbereich: innerhalb des umschliessenden Elements, aber dort nur nach ihrer Definition
- Unglaublich aber wahr:

  Variablen können nach ihrer Definition nicht mehr verändert werden: Immutable Variables

- Unveränderbare “Variablen” sind eine grundlegende Eigenschaft von rein **funktionalen** Sprachen.

Variablen

- Definition einer Variablen, Variante1:

```
<xsl:variable name="name" select="ausdruck" />
```

- Definition, Variante2:

```
<xsl:variable name="name">
    anweisungen
</xsl:variable>
```

- Geltungsbereich: innerhalb des umschliessenden Elements, aber dort nur nach ihrer Definition
- Unglaublich aber wahr:

  Variablen können nach ihrer Definition nicht mehr verändert werden: Immutable Variables

- Unveränderbare “Variablen” sind eine grundlegende Eigenschaft von rein **funktionalen** Sprachen.

Variablen

- Definition einer Variablen, Variante1:

```
<xsl:variable name="name" select="ausdruck" />
```

- Definition, Variante2:

```
<xsl:variable name="name">
    anweisungen
</xsl:variable>
```

- Geltungsbereich: innerhalb des umschliessenden Elements, aber dort nur nach ihrer Definition
- Unglaublich aber wahr:

  Variablen können nach ihrer Definition nicht mehr verändert werden: Immutable Variables

- Unveränderbare “Variablen” sind eine grundlegende Eigenschaft von rein **funktionalen** Sprachen.

Variablen

- Definition einer Variablen, Variante1:

```
<xsl:variable name="name" select="ausdruck" />
```

- Definition, Variante2:

```
<xsl:variable name="name">
    anweisungen
</xsl:variable>
```

- Geltungsbereich: innerhalb des umschliessenden Elements, aber dort nur nach ihrer Definition
- Unglaublich aber wahr:

  Variablen können nach ihrer Definition nicht mehr verändert werden: Immutable Variables

- Unveränderbare “Variablen” sind eine grundlegende Eigenschaft von rein **funktionalen** Sprachen.

Variablen

- Definition einer Variablen, Variante1:

```
<xsl:variable name="name" select="ausdruck" />
```

- Definition, Variante2:

```
<xsl:variable name="name">
    anweisungen
</xsl:variable>
```

- Geltungsbereich: innerhalb des umschliessenden Elements, aber dort nur nach ihrer Definition
- Unglaublich aber wahr:

  Variablen können nach ihrer Definition nicht mehr verändert werden: Immutable Variables

- Unveränderbare “Variablen” sind eine grundlegende Eigenschaft von rein **funktionalen** Sprachen.

Variablen

- Definition einer Variablen, Variante1:

```
<xsl:variable name="name" select="ausdruck" />
```

- Definition, Variante2:

```
<xsl:variable name="name">
    anweisungen
</xsl:variable>
```

- Geltungsbereich: innerhalb des umschliessenden Elements, aber dort nur nach ihrer Definition
- Unglaublich aber wahr:

  Variablen können nach ihrer Definition nicht mehr verändert werden: Immutable Variables

- Unveränderbare “Variablen” sind eine grundlegende Eigenschaft von rein **funktionalen** Sprachen.

Variablen

- Definition einer Variablen, Variante1:

```
<xsl:variable name="name" select="ausdruck" />
```

- Definition, Variante2:

```
<xsl:variable name="name">
    anweisungen
</xsl:variable>
```

- Geltungsbereich: innerhalb des umschliessenden Elements, aber dort nur nach ihrer Definition
- Unglaublich aber wahr:

  Variablen können nach ihrer Definition nicht mehr verändert werden: Immutable Variables

- Unveränderbare “Variablen” sind eine grundlegende Eigenschaft von rein **funktionalen** Sprachen.

Rekursive Programmierung

- Problem: mit xslt soll der Inhalt eines Webshop-Warenkorbs tabellarisch dargestellt werden.
- In der letzten Tabellenzeile soll der Gesamtwarenwert angezeigt werden.
- Aufsummieren in einer nicht veränderbaren Variablen ist nicht möglich!
- Lösung: rekursive Programmierung.
- Pseudocode:

```
function total(liste) {
  if (liste != leer) {
    summe = total(liste ab element 2);
    return summe +
      (1. elem. der liste).preis *
      (1. elem. der liste).bestellmenge;
  }
  else
    return 0;
}
```

Rekursive Programmierung

- Problem: mit xslt soll der Inhalt eines Webshop-Warenkorbs tabellarisch dargestellt werden.
- In der letzten Tabellenzeile soll der Gesamtwarenwert angezeigt werden.
- Aufsummieren in einer nicht veränderbaren Variablen ist nicht möglich!
- Lösung: rekursive Programmierung.
- Pseudocode:

```
function total(liste) {
  if (liste != leer) {
    summe = total(liste ab element 2);
    return summe +
      (1. elem. der liste).preis *
      (1. elem. der liste).bestellmenge;
  }
  else
    return 0;
}
```

Rekursive Programmierung

- Problem: mit xslt soll der Inhalt eines Webshop-Warenkorbs tabellarisch dargestellt werden.
- In der letzten Tabellenzeile soll der Gesamtwarenwert angezeigt werden.
- Aufsummieren in einer nicht veränderbaren Variablen ist nicht möglich!
- Lösung: rekursive Programmierung.
- Pseudocode:

```
function total(liste) {
  if (liste != leer) {
    summe = total(liste ab element 2);
    return summe +
      (1. elem. der liste).preis *
      (1. elem. der liste).bestellmenge;
  }
  else
    return 0;
}
```

Rekursive Programmierung

- Problem: mit xslt soll der Inhalt eines Webshop-Warenkorbs tabellarisch dargestellt werden.
- In der letzten Tabellenzeile soll der Gesamtwarenwert angezeigt werden.
- Aufsummieren in einer nicht veränderbaren Variablen ist nicht möglich!
- Lösung: rekursive Programmierung.
- Pseudocode:

```
function total(liste) {
  if (liste != leer) {
    summe = total(liste ab element 2);
    return summe +
      (1. elem. der liste).preis *
      (1. elem. der liste).bestellmenge;
  }
  else
    return 0;
}
```

Rekursive Programmierung

- Problem: mit xslt soll der Inhalt eines Webshop-Warenkorbs tabellarisch dargestellt werden.
- In der letzten Tabellenzeile soll der Gesamtwarenwert angezeigt werden.
- Aufsummieren in einer nicht veränderbaren Variablen ist nicht möglich!
- Lösung: rekursive Programmierung.
- Pseudocode:

```
function total(liste) {
  if (liste != leer) {
    summe = total(liste ab element 2);
    return summe +
      (1. elem. der liste).preis *
      (1. elem. der liste).bestellmenge;
  }
  else
    return 0;
}
```

Rekursive Programmierung

- Problem: mit xslt soll der Inhalt eines Webshop-Warenkorbs tabellarisch dargestellt werden.
- In der letzten Tabellenzeile soll der Gesamtwarenwert angezeigt werden.
- Aufsummieren in einer nicht veränderbaren Variablen ist nicht möglich!
- Lösung: rekursive Programmierung.
- Pseudocode:

```
function total(liste) {
  if (liste != leer) {
    summe = total(liste ab element 2);
    return summe +
      (1. elem. der liste).preis *
      (1. elem. der liste).bestellmenge;
  }
  else
    return 0;
}
```

Rekursive Programmierung

```
1  <xsl:template name="total">
2   <xsl:param name="liste" />
3    <xsl:choose>
4     <xsl:when test="$liste">
5      <xsl:variable name="summe">
6       <xsl:call-template name="total">
7        <xsl:with-param name="liste" select="$liste[
             position() &gt; 1]" />
8       </xsl:call-template>
9      </xsl:variable>
10     <xsl:value-of select="$summe + $liste/preis * $
         liste/bestellmenge" />
11    </xsl:when>
12   <xsl:otherwise>0</xsl:otherwise>
13  </xsl:choose>
14 </xsl:template>
```

Rekursive Programmierung

```
1  <xsl:template name="total">
2    <xsl:param name="liste" />
3      <xsl:choose>
4        <xsl:when test="$liste">
5          <xsl:variable name="summe">
6            <xsl:call-template name="total">
7              <xsl:with-param name="liste" select="$liste[
                   position() &gt; 1]" />
8            </xsl:call-template>
9          </xsl:variable>
10         <xsl:value-of select="$summe + $liste/preis * $
               liste/bestellmenge" />
11       </xsl:when>
12     <xsl:otherwise>0</xsl:otherwise>
13   </xsl:choose>
14 </xsl:template>
```

Rekursive Programmierung

```
1  <xsl:template name="total">
2   <xsl:param name="liste" />
3    <xsl:choose>
4     <xsl:when test="$liste">
5      <xsl:variable name="summe">
6       <xsl:call-template name="total">
7        <xsl:with-param name="liste" select="$liste[
             position() &gt; 1]" />
8       </xsl:call-template>
9      </xsl:variable>
10     <xsl:value-of select="$summe + $liste/preis * $
           liste/bestellmenge" />
11    </xsl:when>
12   <xsl:otherwise>0</xsl:otherwise>
13  </xsl:choose>
14 </xsl:template>
```

Rekursive Programmierung

```
1  <xsl:template name="total">
2   <xsl:param name="liste" />
3    <xsl:choose>
4     <xsl:when test="$liste">
5      <xsl:variable name="summe">
6       <xsl:call-template name="total">
7        <xsl:with-param name="liste" select="$liste[
             position() &gt; 1]" />
8       </xsl:call-template>
9      </xsl:variable>
10     <xsl:value-of select="$summe + $liste/preis * $
         liste/bestellmenge" />
11    </xsl:when>
12   <xsl:otherwise>0</xsl:otherwise>
13  </xsl:choose>
14 </xsl:template>
```

Rekursive Programmierung

```
1  <xsl:template name="total">
2   <xsl:param name="liste" />
3    <xsl:choose>
4     <xsl:when test="$liste">
5      <xsl:variable name="summe">
6       <xsl:call-template name="total">
7        <xsl:with-param name="liste" select="$liste[
             position() &gt; 1]" />
8       </xsl:call-template>
9      </xsl:variable>
10     <xsl:value-of select="$summe + $liste/preis * $
         liste/bestellmenge" />
11    </xsl:when>
12   <xsl:otherwise>0</xsl:otherwise>
13  </xsl:choose>
14 </xsl:template>
```

Rekursive Programmierung

```
1  <xsl:template name="total">
2   <xsl:param name="liste" />
3    <xsl:choose>
4     <xsl:when test="$liste">
5      <xsl:variable name="summe">
6       <xsl:call-template name="total">
7        <xsl:with-param name="liste" select="$liste[
             position() &gt; 1]" />
8       </xsl:call-template>
9      </xsl:variable>
10     <xsl:value-of select="$summe + $liste/preis * $
         liste/bestellmenge" />
11    </xsl:when>
12   <xsl:otherwise>0</xsl:otherwise>
13  </xsl:choose>
14 </xsl:template>
```

Rekursive Programmierung

```
1  <xsl:template name="total">
2   <xsl:param name="liste" />
3    <xsl:choose>
4     <xsl:when test="$liste">
5      <xsl:variable name="summe">
6       <xsl:call-template name="total">
7        <xsl:with-param name="liste" select="$liste[
             position() &gt; 1]" />
8       </xsl:call-template>
9      </xsl:variable>
10     <xsl:value-of select="$summe + $liste/preis * $
         liste/bestellmenge" />
11    </xsl:when>
12   <xsl:otherwise>0</xsl:otherwise>
13  </xsl:choose>
14 </xsl:template>
```

Rekursive Programmierung

```
1  <xsl:template name="total">
2   <xsl:param name="liste" />
3    <xsl:choose>
4     <xsl:when test="$liste">
5      <xsl:variable name="summe">
6       <xsl:call-template name="total">
7        <xsl:with-param name="liste" select="$liste[
             position() &gt; 1]" />
8         </xsl:call-template>
9       </xsl:variable>
10      <xsl:value-of select="$summe + $liste/preis * $
           liste/bestellmenge" />
11    </xsl:when>
12   <xsl:otherwise>0</xsl:otherwise>
13  </xsl:choose>
14 </xsl:template>
```

Rekursive Programmierung

```
1  <xsl:template name="total">
2   <xsl:param name="liste" />
3    <xsl:choose>
4     <xsl:when test="$liste">
5      <xsl:variable name="summe">
6       <xsl:call-template name="total">
7        <xsl:with-param name="liste" select="$liste[
             position() &gt; 1]" />
8       </xsl:call-template>
9      </xsl:variable>
10     <xsl:value-of select="$summe + $liste/preis * $
           liste/bestellmenge" />
11    </xsl:when>
12   <xsl:otherwise>0</xsl:otherwise>
13  </xsl:choose>
14 </xsl:template>
```

Rekursive Programmierung

```
1  <xsl:template name="total">
2   <xsl:param name="liste" />
3    <xsl:choose>
4     <xsl:when test="$liste">
5      <xsl:variable name="summe">
6       <xsl:call-template name="total">
7        <xsl:with-param name="liste" select="$liste[
             position() &gt; 1]" />
8       </xsl:call-template>
9      </xsl:variable>
10     <xsl:value-of select="$summe + $liste/preis * $
           liste/bestellmenge" />
11    </xsl:when>
12   <xsl:otherwise>0</xsl:otherwise>
13  </xsl:choose>
14 </xsl:template>
```

Rekursive Programmierung

```
1  <xsl:template name="total">
2   <xsl:param name="liste" />
3    <xsl:choose>
4     <xsl:when test="$liste">
5      <xsl:variable name="summe">
6       <xsl:call-template name="total">
7        <xsl:with-param name="liste" select="$liste[
             position() &gt; 1]" />
8       </xsl:call-template>
9      </xsl:variable>
10     <xsl:value-of select="$summe + $liste/preis * $
           liste/bestellmenge" />
11    </xsl:when>
12    <xsl:otherwise>0</xsl:otherwise>
13   </xsl:choose>
14  </xsl:template>
```

Rekursive Programmierung

```
1  <xsl:template name="total">
2   <xsl:param name="liste" />
3    <xsl:choose>
4     <xsl:when test="$liste">
5      <xsl:variable name="summe">
6       <xsl:call-template name="total">
7        <xsl:with-param name="liste" select="$liste[
             position() &gt; 1]" />
8       </xsl:call-template>
9      </xsl:variable>
10     <xsl:value-of select="$summe + $liste/preis * $
           liste/bestellmenge" />
11    </xsl:when>
12   <xsl:otherwise>0</xsl:otherwise>
13  </xsl:choose>
14 </xsl:template>
```

Rekursive Programmierung

```
1  <xsl:template name="total">
2   <xsl:param name="liste" />
3    <xsl:choose>
4     <xsl:when test="$liste">
5      <xsl:variable name="summe">
6       <xsl:call-template name="total">
7        <xsl:with-param name="liste" select="$liste[
             position() &gt; 1]" />
8       </xsl:call-template>
9      </xsl:variable>
10     <xsl:value-of select="$summe + $liste/preis * $
           liste/bestellmenge" />
11    </xsl:when>
12   <xsl:otherwise>0</xsl:otherwise>
13  </xsl:choose>
14 </xsl:template>
```

Inhalt

Highly Verbose

- xml is slightly verbose but highly structured
- xsl-fo is **ultra** verbose
- xsl-fo basiert auf xml: xsl-fo-Dateien sind xml-Dateien
- xsl-fo ist weit besser als html geeignet, einen text milimetergenau für die Druckausgabe zu formatieren. D.h.:
  - html: Bildschirmausgabe
  - xsl-fo: Ausgabe von Druckerzeugnissen
- dabei ist xsl-fo weniger eine Seitenbeschreibungssprache, sondern wurde für grosse, komplexe Dokumente konzipiert.
- z.Zt. verstehen Drucker xsl-fo nicht direkt, daher wird xsl-fo nach pdf gewandelt (Formatting Objects Prozessor fop)

Highly Verbose

- xml is slightly verbose but highly structured
- xsl-fo is **ultra** verbose
- xsl-fo basiert auf xml: xsl-fo-Dateien sind xml-Dateien
- xsl-fo ist weit besser als html geeignet, einen text milimetergenau für die Druckausgabe zu formatieren. D.h.:
  - html: Bildschirmausgabe
  - xsl-fo: Ausgabe von Druckerzeugnissen
- dabei ist xsl-fo weniger eine Seitenbeschreibungssprache, sondern wurde für grosse, komplexe Dokumente konzipiert.
- z.Zt. verstehen Drucker xsl-fo nicht direkt, daher wird xsl-fo nach pdf gewandelt (Formatting Objects Prozessor fop)

Highly Verbose

- xml is slightly verbose but highly structured
- xsl-fo is **ultra** verbose
- xsl-fo basiert auf xml: xsl-fo-Dateien sind xml-Dateien
- xsl-fo ist weit besser als html geeignet, einen text milimetergenau für die Druckausgabe zu formatieren. D.h.:
  - html: Bildschirmausgabe
  - xsl-fo: Ausgabe von Druckerzeugnissen
- dabei ist xsl-fo weniger eine Seitenbeschreibungssprache, sondern wurde für grosse, komplexe Dokumente konzipiert.
- z.Zt. verstehen Drucker xsl-fo nicht direkt, daher wird xsl-fo nach pdf gewandelt (Formatting Objects Prozessor fop)

Highly Verbose

- xml is slightly verbose but highly structured
- xsl-fo is **ultra** verbose
- xsl-fo basiert auf xml: xsl-fo-Dateien sind xml-Dateien
- xsl-fo ist weit besser als html geeignet, einen text milimetergenau für die Druckausgabe zu formatieren. D.h.:
  - html: Bildschirmausgabe
  - xsl-fo: Ausgabe von Druckerzeugnissen
- dabei ist xsl-fo weniger eine Seitenbeschreibungssprache, sondern wurde für grosse, komplexe Dokumente konzipiert.
- z.Zt. verstehen Drucker xsl-fo nicht direkt, daher wird xsl-fo nach pdf gewandelt (Formatting Objects Prozessor fop)

Highly Verbose

- xml is slightly verbose but highly structured
- xsl-fo is **ultra** verbose
- xsl-fo basiert auf xml: xsl-fo-Dateien sind xml-Dateien
- xsl-fo ist weit besser als html geeignet, einen text milimetergenau für die Druckausgabe zu formatieren. D.h.:
  - html: Bildschirmausgabe
  - xsl-fo: Ausgabe von Druckerzeugnissen
- dabei ist xsl-fo weniger eine Seitenbeschreibungssprache, sondern wurde für grosse, komplexe Dokumente konzipiert.
- z.Zt. verstehen Drucker xsl-fo nicht direkt, daher wird xsl-fo nach pdf gewandelt (Formatting Objects Prozessor fop)

Highly Verbose

- xml is slightly verbose but highly structured
- xsl-fo is **ultra** verbose
- xsl-fo basiert auf xml: xsl-fo-Dateien sind xml-Dateien
- xsl-fo ist weit besser als html geeignet, einen text milimetergenau für die Druckausgabe zu formatieren. D.h.:
  - html: Bildschirmausgabe
  - xsl-fo: Ausgabe von Druckerzeugnissen
- dabei ist xsl-fo weniger eine Seitenbeschreibungssprache, sondern wurde für grosse, komplexe Dokumente konzipiert.
- z.Zt. verstehen Drucker xsl-fo nicht direkt, daher wird xsl-fo nach pdf gewandelt (Formatting Objects Prozessor fop)

Highly Verbose

- xml is slightly verbose but highly structured
- xsl-fo is **ultra** verbose
- xsl-fo basiert auf xml: xsl-fo-Dateien sind xml-Dateien
- xsl-fo ist weit besser als html geeignet, einen text milimetergenau für die Druckausgabe zu formatieren. D.h.:
  - html: Bildschirmausgabe
  - xsl-fo: Ausgabe von Druckerzeugnissen
- dabei ist xsl-fo weniger eine Seitenbeschreibungssprache, sondern wurde für grosse, komplexe Dokumente konzipiert.
- z.Zt. verstehen Drucker xsl-fo nicht direkt, daher wird xsl-fo nach pdf gewandelt (Formatting Objects Prozessor fop)

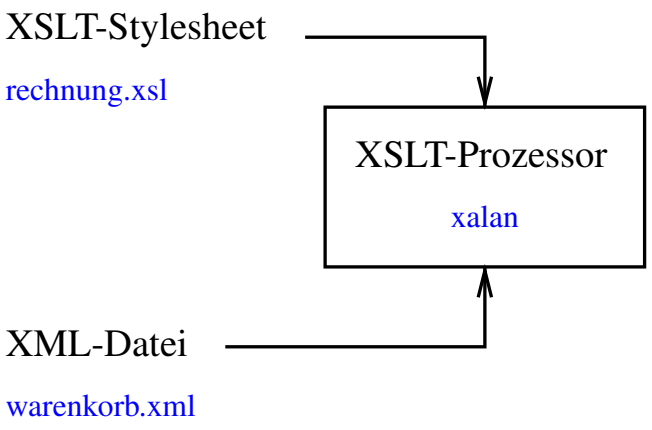
Highly Verbose

- xml is slightly verbose but highly structured
- xsl-fo is **ultra** verbose
- xsl-fo basiert auf xml: xsl-fo-Dateien sind xml-Dateien
- xsl-fo ist weit besser als html geeignet, einen text milimetergenau für die Druckausgabe zu formatieren. D.h.:
  - html: Bildschirmausgabe
  - xsl-fo: Ausgabe von Druckerzeugnissen
- dabei ist xsl-fo weniger eine Seitenbeschreibungssprache, sondern wurde für grosse, komplexe Dokumente konzipiert.
- z.Zt. verstehen Drucker xsl-fo nicht direkt, daher wird xsl-fo nach pdf gewandelt (Formatting Objects Prozessor fop)

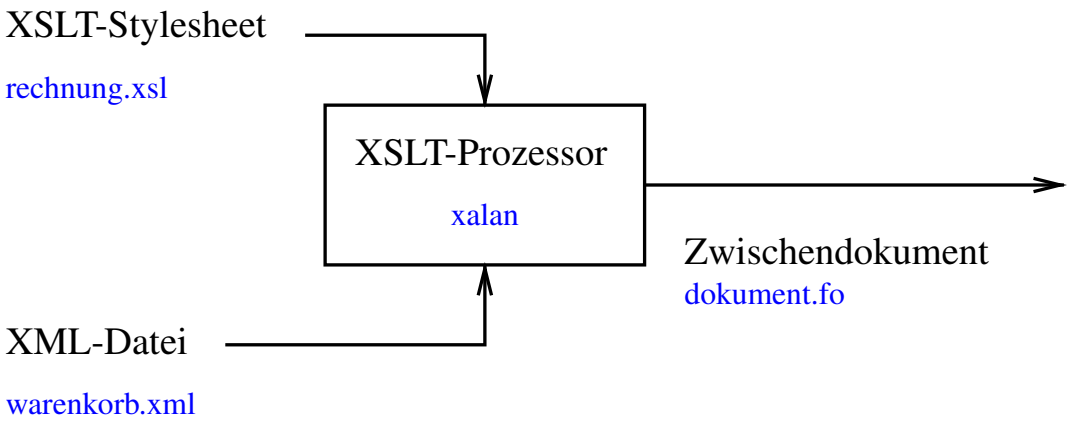
Highly Verbose

- xml is slightly verbose but highly structured
- xsl-fo is **ultra** verbose
- xsl-fo basiert auf xml: xsl-fo-Dateien sind xml-Dateien
- xsl-fo ist weit besser als html geeignet, einen text milimetergenau für die Druckausgabe zu formatieren. D.h.:
  - html: Bildschirmausgabe
  - xsl-fo: Ausgabe von Druckerzeugnissen
- dabei ist xsl-fo weniger eine Seitenbeschreibungssprache, sondern wurde für grosse, komplexe Dokumente konzipiert.
- z.Zt. verstehen Drucker xsl-fo nicht direkt, daher wird xsl-fo nach pdf gewandelt (Formatting Objects Prozessor fop)

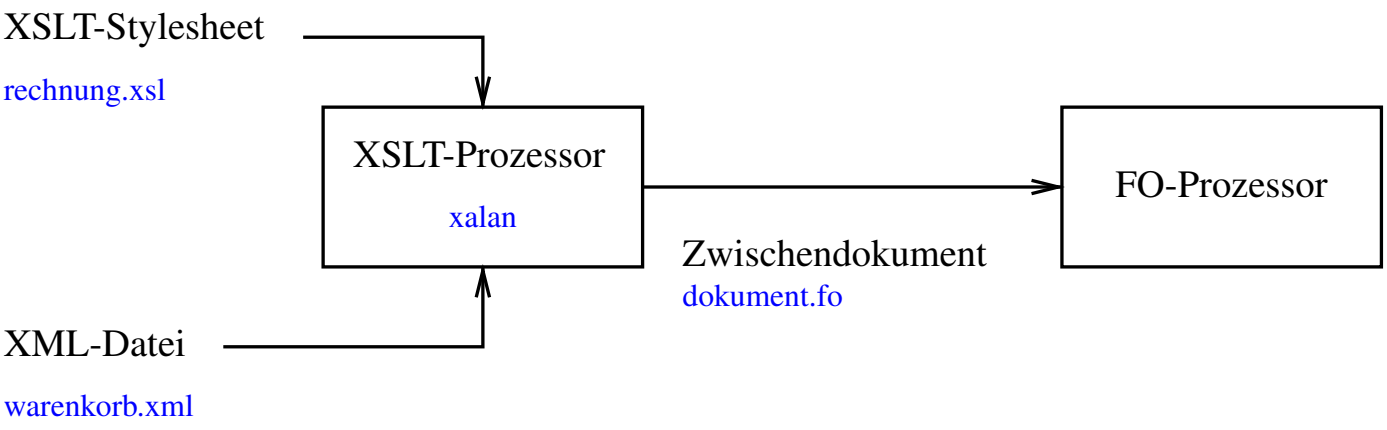
Der Transformationsvorgang am Beispiel von Apache fop



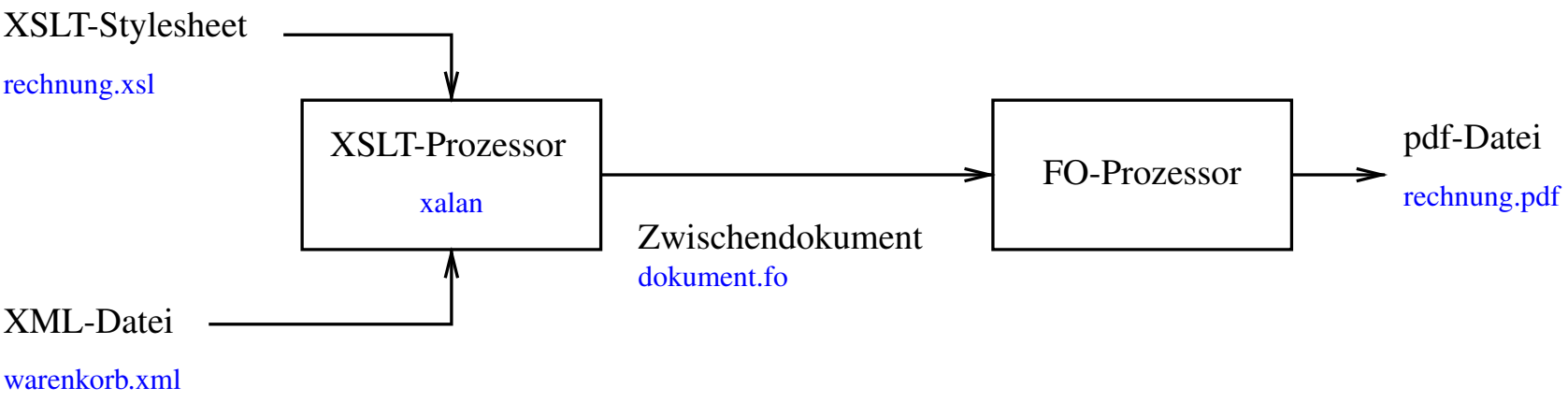
Der Transformationsvorgang am Beispiel von Apache fop



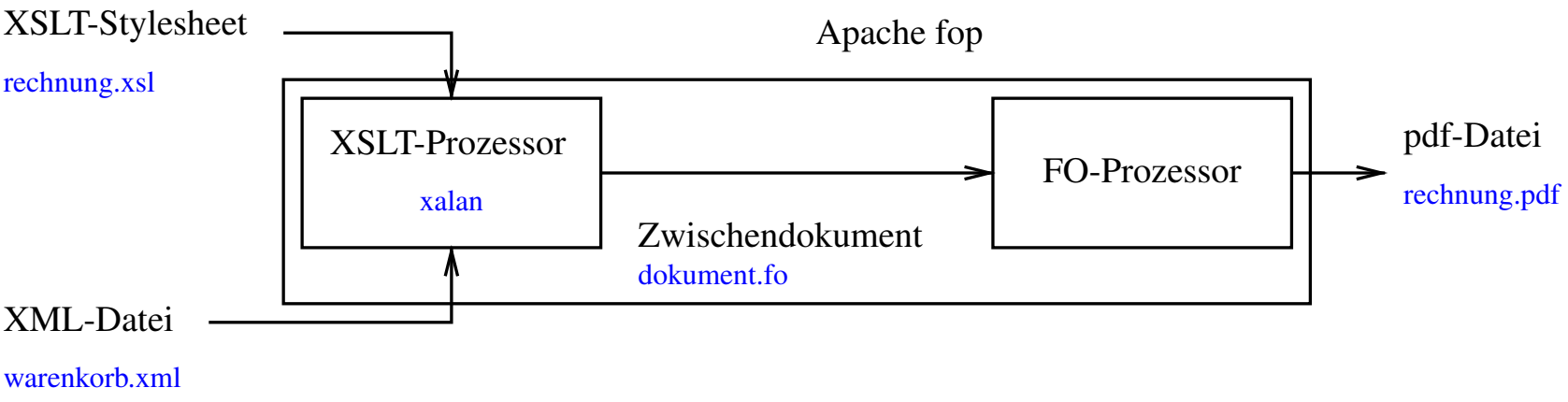
Der Transformationsvorgang am Beispiel von Apache fop



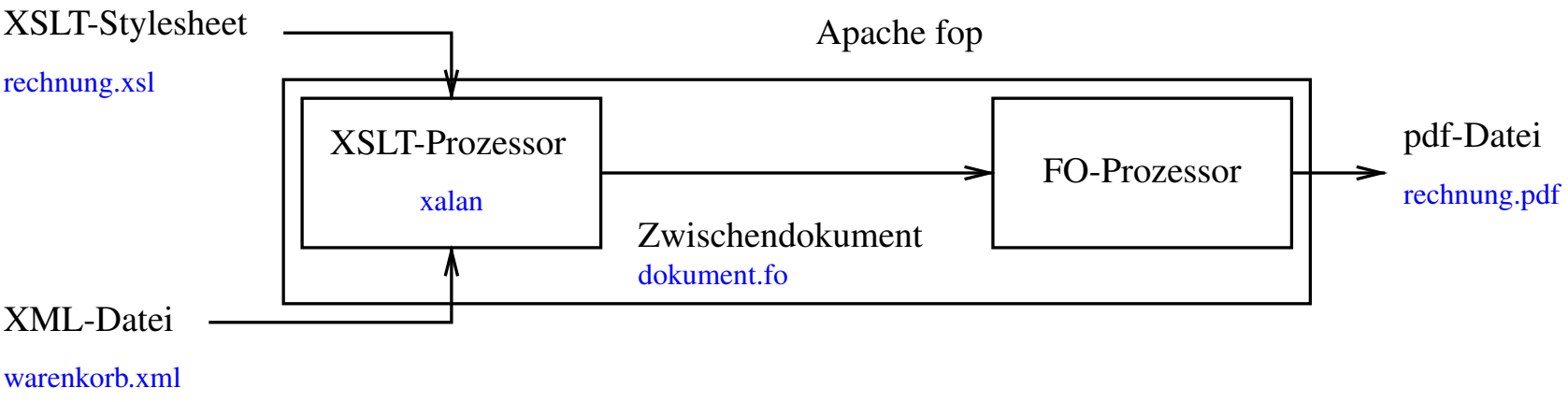
Der Transformationsvorgang am Beispiel von Apache fop



Der Transformationsvorgang am Beispiel von Apache fop

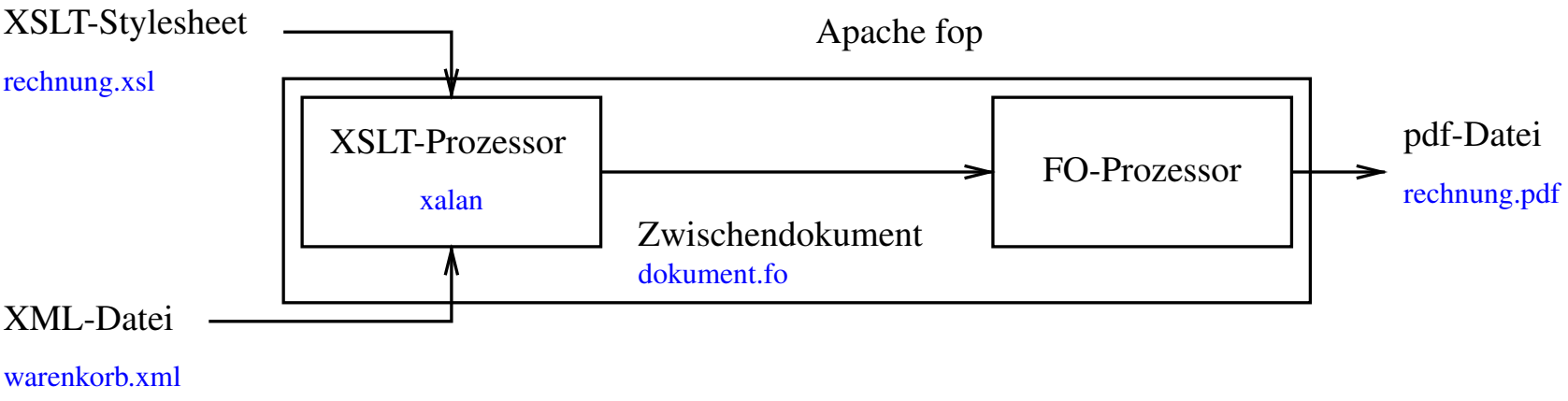


Der Transformationsvorgang am Beispiel von Apache fop



Aufruf:

Der Transformationsvorgang am Beispiel von Apache fop



Aufruf:

```
fop -xml warenkorb.xml -xsl rechnung.xsl -pdf rechnung.pdf
```

Ein DIN-Brief mit Rechnung

libro it

alfred neumann, waelderstrasse 7, 79341 kenzingen

Zeile1
Zeile2
Zeile3
Zeile4
Zeile5
Zeile6

sachberarbeiter
telefon
emaille-adr
datum

bezugszeichenzeile

Sehr geehrte Damen und Herren,

Pos.	Art.-Nr.	Autor	Titel	Aufl.	Verlag	Menge	Einzel-preis	Preis in EUR
1	978-3-540-73338-6	P. Gantenooo, W. Alex	Debian GNU/ Linux	3	Springer	50	59.95	2997.50
2	978-0-470-19274-0	Michael Kay	XSLT 2.0 and XPATH 2.0	4	Wiley Publishing	1	49.90	49.90
3	3-8266-7209-7	F.J. Herpers, T.J. Sebesteyen	XSL, Das Einsteigerseminar	1	verlag moderne industrie	1	9.95	9.95
4	978-3-540-24524-7	Benedikt Stockebrand	IPv6 in Practice	1	Springer	1	9.95	9.95
Gesamtsumme:								3067.30

bankverbindungen:
sparkasse freiburg - konto 987654321 - blz 68050101
postbank karlsruhe - konto 123456789 - blz 66010075

Inhalt

Java Architecture for XML-Binding

- Von XML zu Java: Generierung von Java-Code aus einem XSD-Schema
- Und zurück: Generierung eines XSD-Schemas aus Java-Code (machen wir nicht)
- Serialisierung eines Java-Objekts in ein XML-Dokument: Marshalling
- Erzeugung eines Java-Objekts aus XML-Daten: Unmarshalling

Java Architecture for XML-Binding

- Von XML zu Java: Generierung von Java-Code aus einem XSD-Schema
- Und zurück: Generierung eines XSD-Schemas aus Java-Code (machen wir nicht)
- Serialisierung eines Java-Objekts in ein XML-Dokument: Marshalling
- Erzeugung eines Java-Objekts aus XML-Daten: Unmarshalling

Java Architecture for XML-Binding

- Von XML zu Java: Generierung von Java-Code aus einem XSD-Schema
- Und zurück: Generierung eines XSD-Schemas aus Java-Code (machen wir nicht)
- Serialisierung eines Java-Objekts in ein XML-Dokument: Marshalling
- Erzeugung eines Java-Objekts aus XML-Daten: Unmarshalling

Java Architecture for XML-Binding

- Von XML zu Java: Generierung von Java-Code aus einem XSD-Schema
- Und zurück: Generierung eines XSD-Schemas aus Java-Code (machen wir nicht)
- Serialisierung eines Java-Objekts in ein XML-Dokument: Marshalling
- Erzeugung eines Java-Objekts aus XML-Daten: Unmarshalling

Java Architecture for XML-Binding

- Von XML zu Java: Generierung von Java-Code aus einem XSD-Schema
- Und zurück: Generierung eines XSD-Schemas aus Java-Code (machen wir nicht)
- Serialisierung eines Java-Objekts in ein XML-Dokument: Marshalling
- Erzeugung eines Java-Objekts aus XML-Daten: Unmarshalling

Java Architecture for XML-Binding

- Von XML zu Java: Generierung von Java-Code aus einem XSD-Schema
- Und zurück: Generierung eines XSD-Schemas aus Java-Code (machen wir nicht)
- Serialisierung eines Java-Objekts in ein XML-Dokument: Marshalling
- Erzeugung eines Java-Objekts aus XML-Daten: Unmarshalling

Java Architecture for XML-Binding

- Von XML zu Java: Generierung von Java-Code aus einem XSD-Schema
- Und zurück: Generierung eines XSD-Schemas aus Java-Code (machen wir nicht)
- Serialisierung eines Java-Objekts in ein XML-Dokument: Marshalling
- Erzeugung eines Java-Objekts aus XML-Daten: Unmarshalling

Java Architecture for XML-Binding

- Von XML zu Java: Generierung von Java-Code aus einem XSD-Schema
- Und zurück: Generierung eines XSD-Schemas aus Java-Code (machen wir nicht)
- Serialisierung eines Java-Objekts in ein XML-Dokument: Marshalling
- Erzeugung eines Java-Objekts aus XML-Daten: Unmarshalling

Java Architecture for XML-Binding

- Von XML zu Java: Generierung von Java-Code aus einem XSD-Schema
- Und zurück: Generierung eines XSD-Schemas aus Java-Code (machen wir nicht)
- Serialisierung eines Java-Objekts in ein XML-Dokument: Marshalling
- Erzeugung eines Java-Objekts aus XML-Daten: Unmarshalling

Java Architecture for XML-Binding

- Von XML zu Java: Generierung von Java-Code aus einem XSD-Schema
- Und zurück: Generierung eines XSD-Schemas aus Java-Code (machen wir nicht)
- Serialisierung eines Java-Objekts in ein XML-Dokument: Marshalling
- Erzeugung eines Java-Objekts aus XML-Daten: Unmarshalling

Generierung von JAVA-Code

XSD-Datei (gekürzt):

```
1  <xs:schema ... >
2  <xs:element name="kontakte">
3   <xs:complexType><xs:sequence>
4    <xs:element name="Person" maxOccurs="unbounded">
5     <xs:complexType><xs:sequence>
6       <xs:element name="name" type="xs:string" />
7       <xs:element name="adresse" type="xs:string"
            minOccurs="0" />
8      </xs:sequence>
9      <xs:attribute name="id" type="xs:long" use="
           required" />
10    </xs:complexType> ...
```

Generierung von JAVA-Code

XSD-Datei (gekürzt):

```
1  <xs:schema ... >
2  <xs:element name="kontakte">
3   <xs:complexType><xs:sequence>
4    <xs:element name="Person" maxOccurs="unbounded">
5     <xs:complexType><xs:sequence>
6       <xs:element name="name" type="xs:string" />
7       <xs:element name="adresse" type="xs:string"
            minOccurs="0" />
8      </xs:sequence>
9      <xs:attribute name="id" type="xs:long" use="
           required" />
10    </xs:complexType> ...
```

Generierung von JAVA-Code

XSD-Datei (gekürzt):

```
1  <xs:schema ... >
2  <xs:element name="kontakte">
3   <xs:complexType><xs:sequence>
4    <xs:element name="Person" maxOccurs="unbounded">
5     <xs:complexType><xs:sequence>
6       <xs:element name="name" type="xs:string" />
7       <xs:element name="adresse" type="xs:string"
            minOccurs="0" />
8      </xs:sequence>
9      <xs:attribute name="id" type="xs:long" use="
           required" />
10    </xs:complexType> ...
```

Generierung von JAVA-Code

XSD-Datei (gekürzt):

```
1  <xs:schema ... >
2  <xs:element name="kontakte">
3    <xs:complexType><xs:sequence>
4      <xs:element name="Person" maxOccurs="unbounded">
5        <xs:complexType><xs:sequence>
6          <xs:element name="name" type="xs:string" />
7          <xs:element name="adresse" type="xs:string"
               minOccurs="0" />
8        </xs:sequence>
9        <xs:attribute name="id" type="xs:long" use="
             required" />
10     </xs:complexType> ...
```

Generierung von JAVA-Code

XSD-Datei (gekürzt):

```
1  <xs:schema ... >
2  <xs:element name="kontakte">
3   <xs:complexType><xs:sequence>
4    <xs:element name="Person" maxOccurs="unbounded">
5     <xs:complexType><xs:sequence>
6      <xs:element name="name" type="xs:string" />
7      <xs:element name="adresse" type="xs:string"
           minOccurs="0" />
8     </xs:sequence>
9     <xs:attribute name="id" type="xs:long" use="
          required" />
10   </xs:complexType> ...
```

Generierung von JAVA-Code

XSD-Datei (gekürzt):

```
1  <xs:schema ... >
2  <xs:element name="kontakte">
3   <xs:complexType><xs:sequence>
4    <xs:element name="Person" maxOccurs="unbounded">
5     <xs:complexType><xs:sequence>
6      <xs:element name="name" type="xs:string" />
7      <xs:element name="adresse" type="xs:string"
           minOccurs="0" />
8     </xs:sequence>
9     <xs:attribute name="id" type="xs:long" use="
         required" />
10   </xs:complexType> ...
```

Generierung von JAVA-Code

XSD-Datei (gekürzt):

```
1  <xs:schema ... >
2  <xs:element name="kontakte">
3   <xs:complexType><xs:sequence>
4    <xs:element name="Person" maxOccurs="unbounded">
5     <xs:complexType><xs:sequence>
6       <xs:element name="name" type="xs:string" />
7       <xs:element name="adresse" type="xs:string"
            minOccurs="0" />
8      </xs:sequence>
9      <xs:attribute name="id" type="xs:long" use="
           required" />
10    </xs:complexType> ...
```

Generierung von JAVA-Code

XSD-Datei (gekürzt):

```
1  <xs:schema ... >
2  <xs:element name="kontakte">
3   <xs:complexType><xs:sequence>
4    <xs:element name="Person" maxOccurs="unbounded">
5     <xs:complexType><xs:sequence>
6       <xs:element name="name" type="xs:string" />
7       <xs:element name="adresse" type="xs:string"
            minOccurs="0" />
8      </xs:sequence>
9      <xs:attribute name="id" type="xs:long" use="
           required" />
10    </xs:complexType> ...
```

Generierung von JAVA-Code

XSD-Datei (gekürzt):

```
1  <xs:schema ... >
2  <xs:element name="kontakte">
3   <xs:complexType><xs:sequence>
4    <xs:element name="Person" maxOccurs="unbounded">
5     <xs:complexType><xs:sequence>
6       <xs:element name="name" type="xs:string" />
7       <xs:element name="adresse" type="xs:string"
            minOccurs="0" />
8      </xs:sequence>
9      <xs:attribute name="id" type="xs:long" use="
           required" />
10    </xs:complexType> ...
```

Generierung von JAVA-Code

XSD-Datei (gekürzt):

```
1  <xs:schema ... >
2  <xs:element name="kontakte">
3   <xs:complexType><xs:sequence>
4    <xs:element name="Person" maxOccurs="unbounded">
5     <xs:complexType><xs:sequence>
6       <xs:element name="name" type="xs:string" />
7       <xs:element name="adresse" type="xs:string"
            minOccurs="0" />
8     </xs:sequence>
9     <xs:attribute name="id" type="xs:long" use="
          required" />
10   </xs:complexType> ...
```

Generierung von JAVA-Code

XSD-Datei (gekürzt):

```
1  <xs:schema ... >
2  <xs:element name="kontakte">
3   <xs:complexType><xs:sequence>
4    <xs:element name="Person" maxOccurs="unbounded">
5     <xs:complexType><xs:sequence>
6       <xs:element name="name" type="xs:string" />
7       <xs:element name="adresse" type="xs:string"
            minOccurs="0" />
8      </xs:sequence>
9      <xs:attribute name="id" type="xs:long" use="
           required" />
10    </xs:complexType> ...
```

Generierung von JAVA-Code

XSD-Datei (gekürzt):

```
1  <xs:schema ... >
2  <xs:element name="kontakte">
3   <xs:complexType><xs:sequence>
4    <xs:element name="Person" maxOccurs="unbounded">
5     <xs:complexType><xs:sequence>
6       <xs:element name="name" type="xs:string" />
7       <xs:element name="adresse" type="xs:string"
            minOccurs="0" />
8      </xs:sequence>
9      <xs:attribute name="id" type="xs:long" use="
           required" />
10    </xs:complexType> ...
```

Generierung von JAVA-Code

XSD-Datei (gekürzt):

```
1  <xs:schema ... >
2  <xs:element name="kontakte">
3   <xs:complexType><xs:sequence>
4    <xs:element name="Person" maxOccurs="unbounded">
5     <xs:complexType><xs:sequence>
6       <xs:element name="name" type="xs:string" />
7       <xs:element name="adresse" type="xs:string"
            minOccurs="0" />
8      </xs:sequence>
9      <xs:attribute name="id" type="xs:long" use="
           required" />
10    </xs:complexType> ...
```

Generierung von JAVA-Code

Beispiel einer nach diesem XSD-Schema *validen* XML-Datei:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<kontakte>
    <Person id="19631223">
        <name>dienert, michael</name>
        <adresse>waelderstrasse 7, 79341 kenzingen</adresse>
    </Person>
    <Person id="19700101">
        <name>neuman, alfred e.</name>
        <adresse>locaRoca 666, 1234 dancelingen</adresse>
    </Person>
    <Person id="20000210">
        <name>bosak, jon</name>
        <adresse>
           4150 Network Cir, Santa Clara, CA USA
        </adresse>
    </Person>
</kontakte>
```

Generierung von JAVA-Code

Beispiel einer nach diesem XSD-Schema *validen* XML-Datei:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<kontakte>
    <Person id="19631223">
        <name>dienert, michael</name>
        <adresse>waelderstrasse 7, 79341 kenzingen</adresse>
    </Person>
    <Person id="19700101">
        <name>neuman, alfred e.</name>
        <adresse>locaRoca 666, 1234 dancelingen</adresse>
    </Person>
    <Person id="20000210">
        <name>bosak, jon</name>
        <adresse>
           4150 Network Cir, Santa Clara, CA USA
        </adresse>
    </Person>
</kontakte>
```

Generierung von JAVA-Code

Beispiel einer nach diesem XSD-Schema *validen* XML-Datei:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<kontakte>
    <Person id="19631223">
        <name>dienert, michael</name>
        <adresse>waelderstrasse 7, 79341 kenzingen</adresse>
    </Person>
    <Person id="19700101">
        <name>neuman, alfred e.</name>
        <adresse>locaRoca 666, 1234 dancelingen</adresse>
    </Person>
    <Person id="20000210">
        <name>bosak, jon</name>
        <adresse>
           4150 Network Cir, Santa Clara, CA USA
        </adresse>
    </Person>
</kontakte>
```

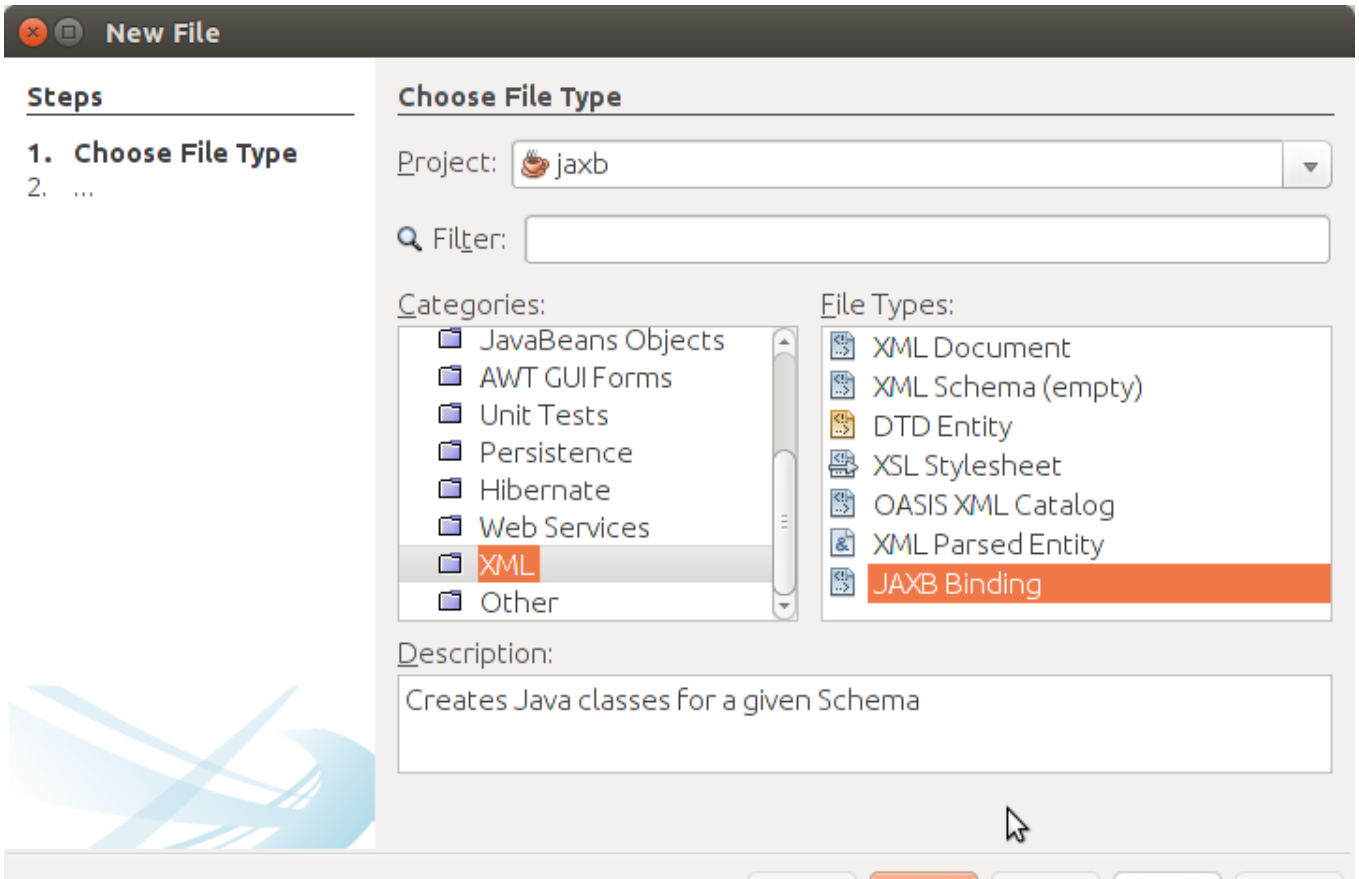
Generierung von JAVA-Code

Java-Code generieren mit dem Kommando `xjc`:

```
xjc kontakte.xsd
```

Direktes Erzeugen der Quellen innerhalb eines NB-Projekts:

File-Menue → New File → Categorie: XML → Type: JAXB Binding



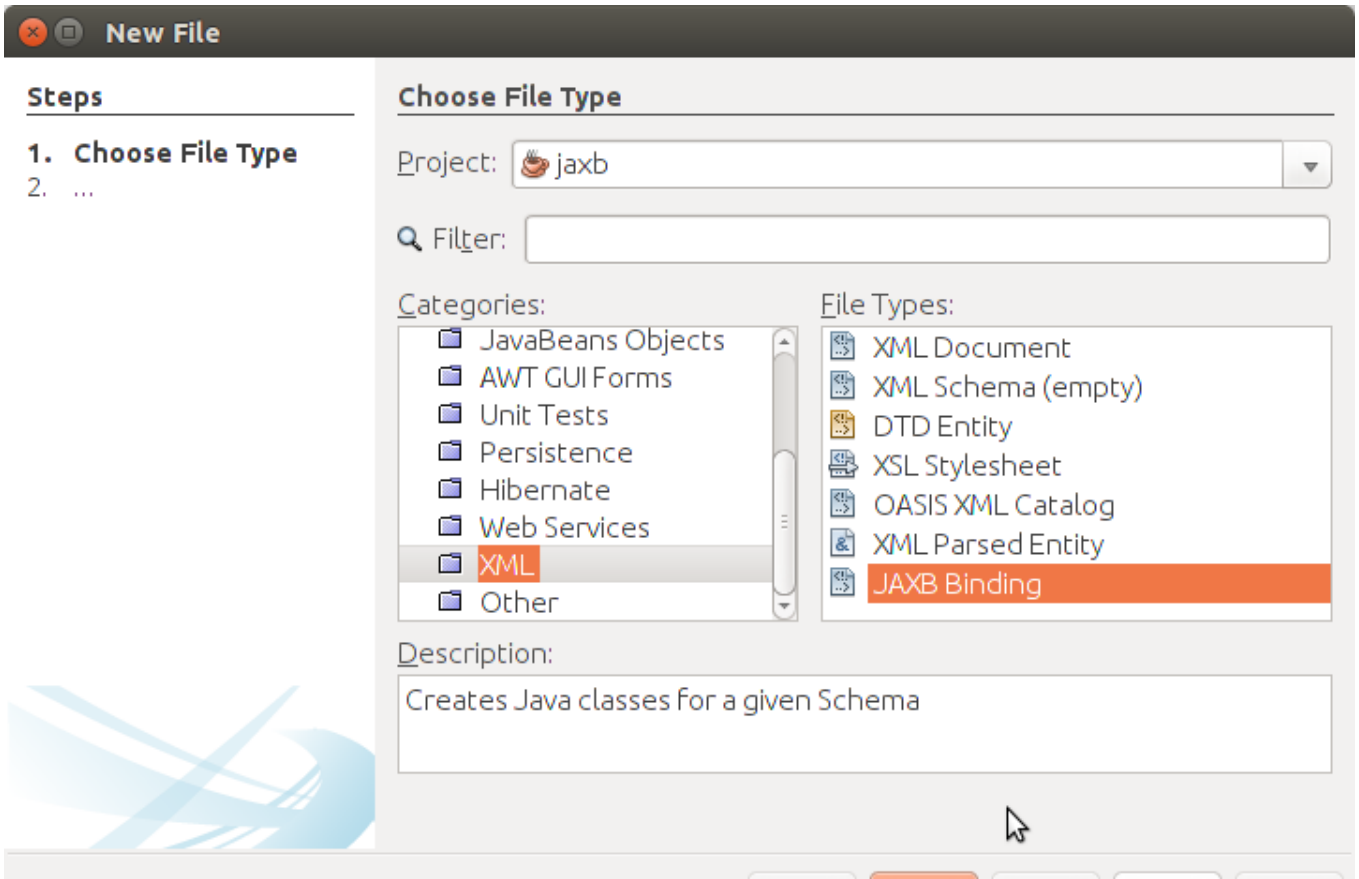
Generierung von JAVA-Code

Java-Code generieren mit dem Kommando `xjc`:

```
xjc kontakte.xsd
```

Direktes Erzeugen der Quellen innerhalb eines NB-Projekts:

File-Menue → New File → Categorie: XML → Type: JAXB Binding



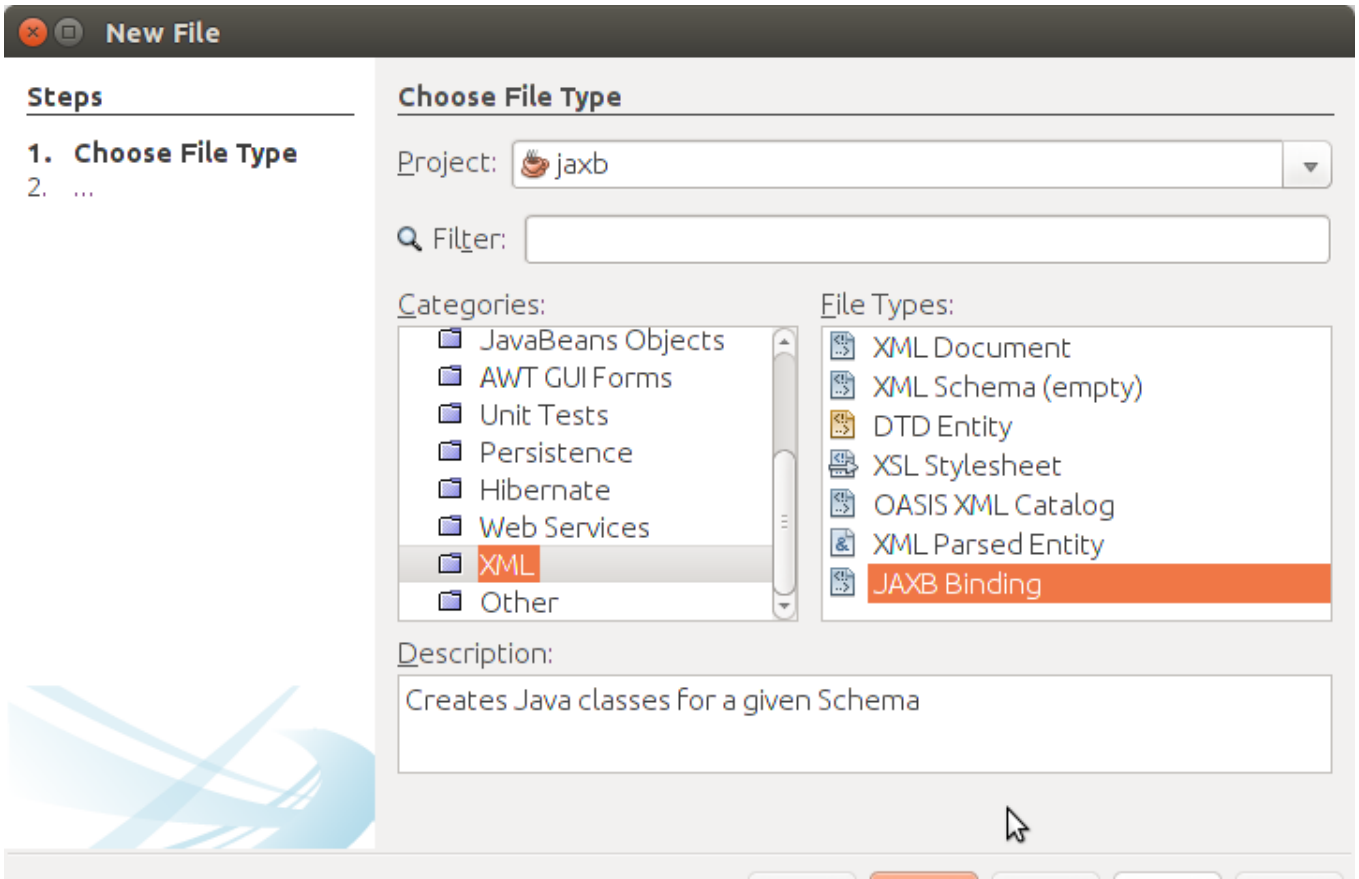
Generierung von JAVA-Code

Java-Code generieren mit dem Kommando `xjc`:

```
xjc kontakte.xsd
```

Direktes Erzeugen der Quellen innerhalb eines NB-Projekts:

File-Menue → New File → Categorie: XML → Type: JAXB Binding



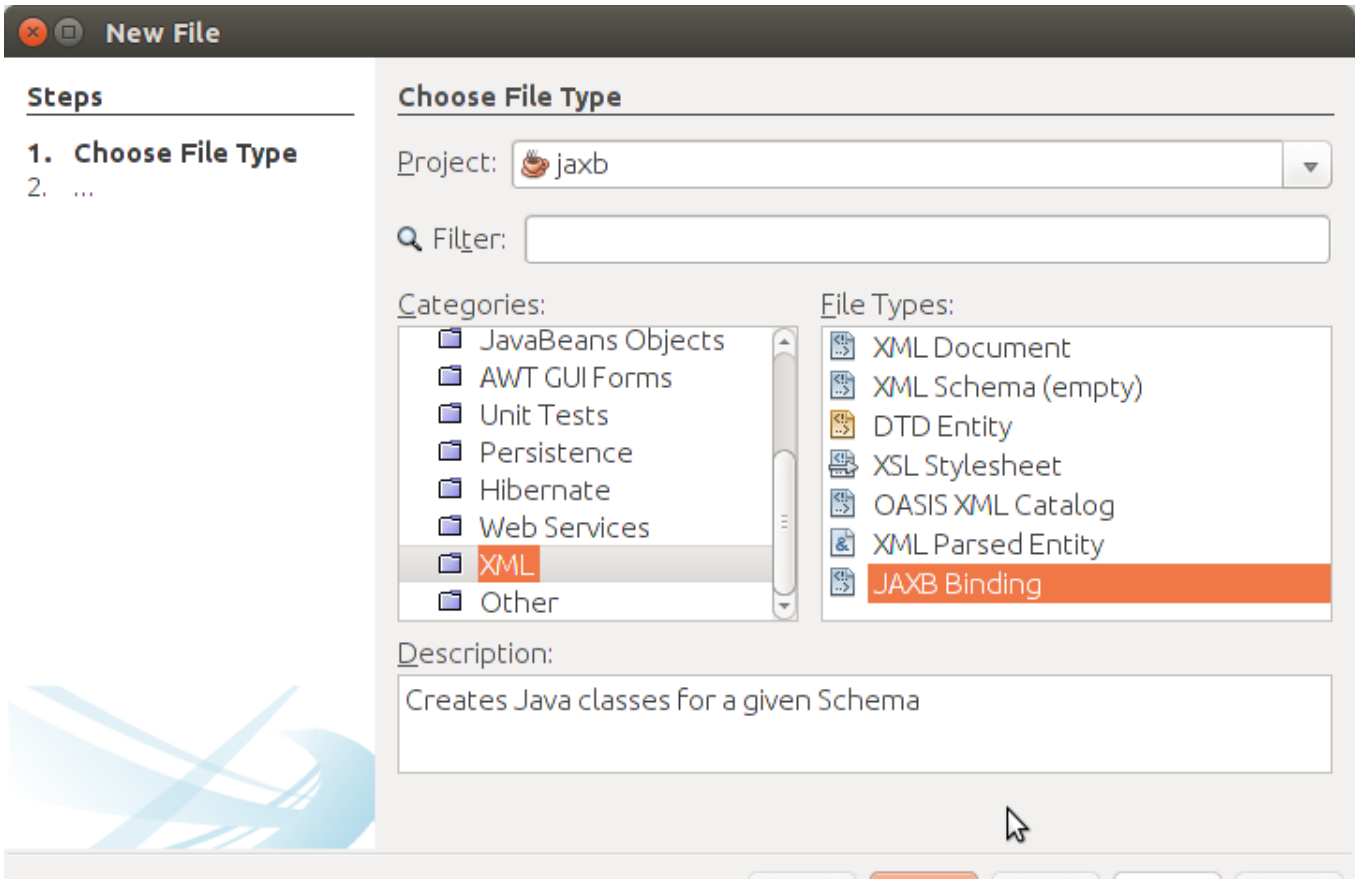
Generierung von JAVA-Code

Java-Code generieren mit dem Kommando `xjc`:

```
xjc kontakte.xsd
```

Direktes Erzeugen der Quellen innerhalb eines NB-Projekts:

File-Menue → New File → Categorie: XML → Type: JAXB Binding



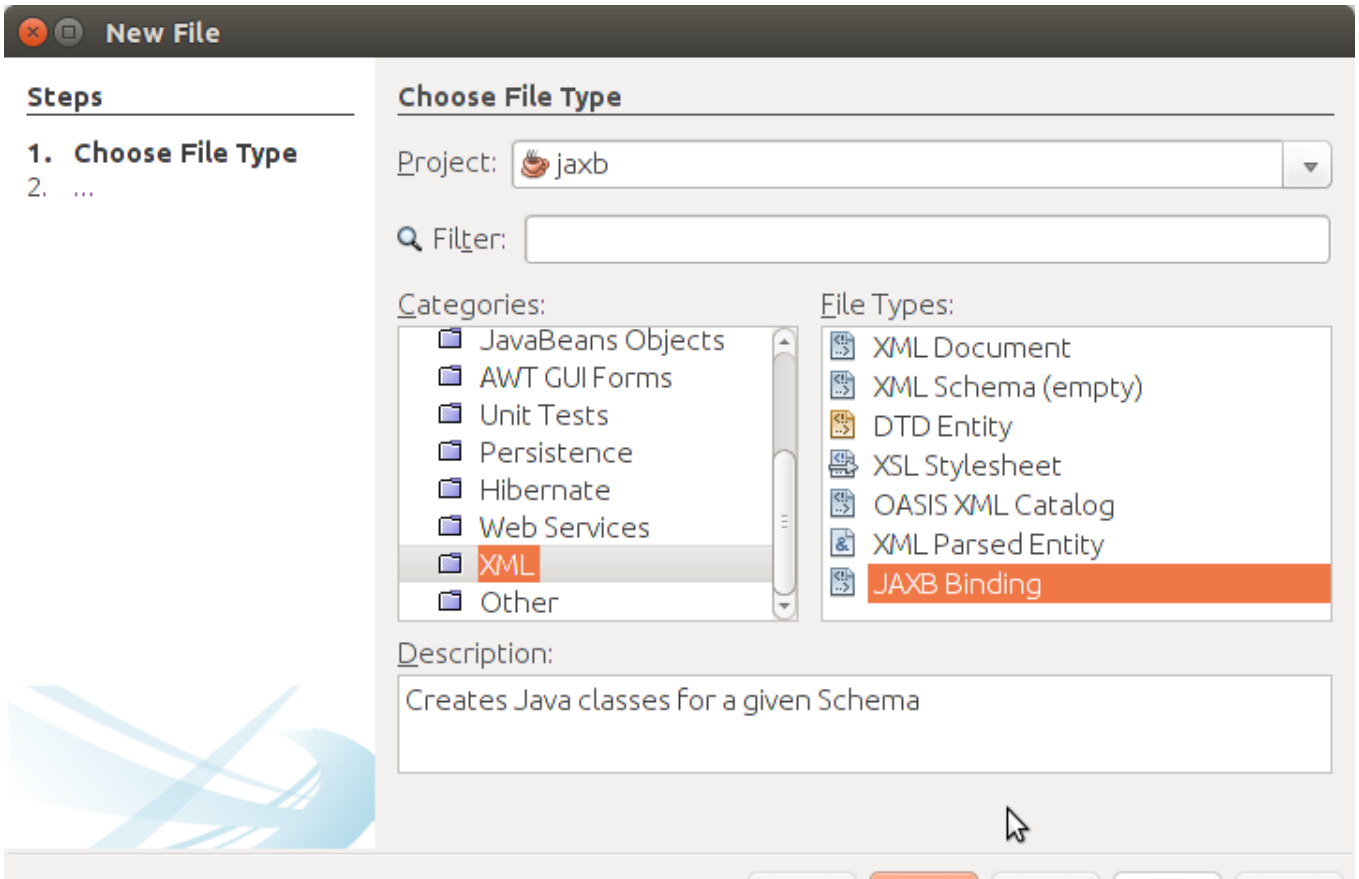
Generierung von JAVA-Code

Java-Code generieren mit dem Kommando `xjc`:

```
xjc kontakte.xsd
```

Direktes Erzeugen der Quellen innerhalb eines NB-Projekts:

File-Menue → New File → Categorie: XML → Type: JAXB Binding



Generierung von JAVA-Code

- Die XSD-Datei beschreibt eine xml-Datei, die beliebig viele (`maxOccurs="unbounded"`) <Person>-Elemente enthalten darf.
- Jedes <Person>-Element muss ein Atrribut **id** haben.
- Jedes <Person>-Element enthält **genau ein** (weder minOccurs noch maxOccurs vorhanden) Element <name>.
- Jedes <Person>-Element enthält **optional** (`minOccurs=0`, maxOccurs nicht vorhanden, d.h `maxOccurs="1"`) ein Element <adresse>.

Generierung von JAVA-Code

- Die XSD-Datei beschreibt eine xml-Datei, die beliebig viele (`maxOccurs="unbounded"`) <Person>-Elemente enthalten darf.
- Jedes <Person>-Element muss ein Atrribut **id** haben.
- Jedes <Person>-Element enthält **genau ein** (weder minOccurs noch maxOccurs vorhanden) Element <name>.
- Jedes <Person>-Element enthält **optional** (`minOccurs=0`, maxOccurs nicht vorhanden, d.h `maxOccurs="1"`) ein Element <adresse>.

Generierung von JAVA-Code

- Die XSD-Datei beschreibt eine xml-Datei, die beliebig viele (`maxOccurs="unbounded"`) <Person>-Elemente enthalten darf.
- Jedes <Person>-Element muss ein Atrribut **id** haben.
- Jedes <Person>-Element enthält **genau ein** (weder minOccurs noch maxOccurs vorhanden) Element <name>.
- Jedes <Person>-Element enthält **optional** (`minOccurs=0`, maxOccurs nicht vorhanden, d.h `maxOccurs="1"`) ein Element <adresse>.

Generierung von JAVA-Code

- Die XSD-Datei beschreibt eine xml-Datei, die beliebig viele (`maxOccurs="unbounded"`) <Person>-Elemente enthalten darf.
- Jedes <Person>-Element muss ein Atrribut **id** haben.
- Jedes <Person>-Element enthält **genau ein** (weder minOccurs noch maxOccurs vorhanden) Element <name>.
- Jedes <Person>-Element enthält **optional** (`minOccurs=0`, maxOccurs nicht vorhanden, d.h `maxOccurs="1"`) ein Element <adresse>.

Generierung von JAVA-Code

- Die XSD-Datei beschreibt eine xml-Datei, die beliebig viele (`maxOccurs="unbounded"`) <Person>-Elemente enthalten darf.
- Jedes <Person>-Element muss ein Atrribut **id** haben.
- Jedes <Person>-Element enthält **genau ein** (weder minOccurs noch maxOccurs vorhanden) Element <name>.
- Jedes <Person>-Element enthält **optional** (`minOccurs=0`, maxOccurs nicht vorhanden, d.h `maxOccurs="1"`) ein Element <adresse>.

Generierung von JAVA-Code

- JAXB erzeugt eine Klasse, mit dem Namen des Wurzelelements: `Kontakte.java`
- Die vielen <Personen>-Elemente bildet JAXB auf eine `ArrayList` ab, die Objekte vom Typ `Kontakte.Person` enthalten darf. d.h. `Person` ist innere Klasse von `Kontakte`.
- Die innere Klasse **Person** wiederum ist nichts anderes wie eine **JavaBean** mit den Eigenschaften:
  - `String name;`
  - `String adresse;`
  - `long id;`

Generierung von JAVA-Code

- JAXB erzeugt eine Klasse, mit dem Namen des Wurzelelements: `Kontakte.java`
- Die vielen <Personen>-Elemente bildet JAXB auf eine `ArrayList` ab, die Objekte vom Typ `Kontakte.Person` enthalten darf. d.h. `Person` ist innere Klasse von `Kontakte`.
- Die innere Klasse **Person** wiederum ist nichts anderes wie eine **JavaBean** mit den Eigenschaften:
  - `String name;`
  - `String adresse;`
  - `long id;`

Generierung von JAVA-Code

- JAXB erzeugt eine Klasse, mit dem Namen des Wurzelelements: `Kontakte.java`
- Die vielen <Personen>-Elemente bildet JAXB auf eine `ArrayList` ab, die Objekte vom Typ `Kontakte.Person` enthalten darf. d.h. `Person` ist innere Klasse von `Kontakte`.
- Die innere Klasse **Person** wiederum ist nichts anderes wie eine **JavaBean** mit den Eigenschaften:
  - `String name;`
  - `String adresse;`
  - `long id;`

Generierung von JAVA-Code

- JAXB erzeugt eine Klasse, mit dem Namen des Wurzelelements: `Kontakte.java`
- Die vielen <Personen>-Elemente bildet JAXB auf eine `ArrayList` ab, die Objekte vom Typ `Kontakte.Person` enthalten darf. d.h. `Person` ist innere Klasse von `Kontakte`.
- Die innere Klasse **Person** wiederum ist nichts anderes wie eine **JavaBean** mit den Eigenschaften:
  - `String name;`
  - `String adresse;`
  - `long id;`

Generierung von JAVA-Code

- JAXB erzeugt eine Klasse, mit dem Namen des Wurzelelements: `Kontakte.java`
- Die vielen <Personen>-Elemente bildet JAXB auf eine `ArrayList` ab, die Objekte vom Typ `Kontakte.Person` enthalten darf. d.h. `Person` ist innere Klasse von `Kontakte`.
- Die innere Klasse **Person** wiederum ist nichts anderes wie eine **JavaBean** mit den Eigenschaften:
  - `String name;`
  - `String adresse;`
  - `long id;`

Generierung von JAVA-Code

- JAXB erzeugt eine Klasse, mit dem Namen des Wurzelelements: `Kontakte.java`
- Die vielen <Personen>-Elemente bildet JAXB auf eine `ArrayList` ab, die Objekte vom Typ `Kontakte.Person` enthalten darf. d.h. `Person` ist innere Klasse von `Kontakte`.
- Die innere Klasse **Person** wiederum ist nichts anderes wie eine **JavaBean** mit den Eigenschaften:
  - `String name;`
  - `String adresse;`
  - `long id;`

Generierung von JAVA-Code

- JAXB erzeugt eine Klasse, mit dem Namen des Wurzelelements: `Kontakte.java`
- Die vielen <Personen>-Elemente bildet JAXB auf eine `ArrayList` ab, die Objekte vom Typ `Kontakte.Person` enthalten darf. d.h. `Person` ist innere Klasse von `Kontakte`.
- Die innere Klasse **Person** wiederum ist nichts anderes wie eine **JavaBean** mit den Eigenschaften:
  - `String name;`
  - `String adresse;`
  - `long id;`

Generierung von JAVA-Code

- JAXB erzeugt eine Klasse, mit dem Namen des Wurzelelements: `Kontakte.java`
- Die vielen <Personen>-Elemente bildet JAXB auf eine `ArrayList` ab, die Objekte vom Typ `Kontakte.Person` enthalten darf. d.h. `Person` ist innere Klasse von `Kontakte`.
- Die innere Klasse **Person** wiederum ist nichts anderes wie eine **JavaBean** mit den Eigenschaften:
  - `String name;`
  - `String adresse;`
  - `long id;`

Generierung von JAVA-Code

- JAXB erzeugt eine Klasse, mit dem Namen des Wurzelelements: `Kontakte.java`
- Die vielen <Personen>-Elemente bildet JAXB auf eine `ArrayList` ab, die Objekte vom Typ `Kontakte.Person` enthalten darf. d.h. `Person` ist innere Klasse von `Kontakte`.
- Die innere Klasse **Person** wiederum ist nichts anderes wie eine **JavaBean** mit den Eigenschaften:
  - `String name;`
  - `String adresse;`
  - `long id;`

Marshalling: Java-Ojekt nach XML

- Marshalling: ähnlich Serialisierung von Objekten; Orignalbedeutung: Einweisen eines Flugzeugs auf dem Flugfeld zur Startbahn durch den Marshall.
- Code-Schnipsel:

```
1  // create JAXB context and instantiate marshaller
2  JAXBContext context = JAXBContext.newInstance(Kontakte.
       class);
3  Marshaller m = context.createMarshaller();
4  m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.
       TRUE);
5
6  // Write to System.out
7  m.marshal(kontakte, System.out);
8
9  // Write to File
10 m.marshal(kontakte, new File("/tmp/kontakte.xml"));
```

Marshalling: Java-Ojekt nach XML

- Marshalling: ähnlich Serialisierung von Objekten;
  Orignalbedeutung: Einweisen eines Flugzeugs auf dem Flugfeld zur Startbahn durch den Marshall.
- Code-Schnipsel:

```
1  // create JAXB context and instantiate marshaller
2  JAXBContext context = JAXBContext.newInstance(Kontakte.
       class);
3  Marshaller m = context.createMarshaller();
4  m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.
       TRUE);
5
6  // Write to System.out
7  m.marshal(kontakte, System.out);
8
9  // Write to File
10 m.marshal(kontakte, new File("/tmp/kontakte.xml"));
```

Marshalling: Java-Ojekt nach XML

- Marshalling: ähnlich Serialisierung von Objekten; Orignalbedeutung: Einweisen eines Flugzeugs auf dem Flugfeld zur Startbahn durch den Marshall.
- Code-Schnipsel:

```
1  // create JAXB context and instantiate marshaller
2  JAXBContext context = JAXBContext.newInstance(Kontakte.
       class);
3  Marshaller m = context.createMarshaller();
4  m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.
       TRUE);
5
6  // Write to System.out
7  m.marshal(kontakte, System.out);
8
9  // Write to File
10 m.marshal(kontakte, new File("/tmp/kontakte.xml"));
```

Marshalling: Java-Ojekt nach XML

- Marshalling: ähnlich Serialisierung von Objekten; Orignalbedeutung: Einweisen eines Flugzeugs auf dem Flugfeld zur Startbahn durch den Marshall.
- Code-Schnipsel:

```
1  // create JAXB context and instantiate marshaller
2  JAXBContext context = JAXBContext.newInstance(Kontakte.
       class);
3  Marshaller m = context.createMarshaller();
4  m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.
       TRUE);
5
6  // Write to System.out
7  m.marshal(kontakte, System.out);
8
9  // Write to File
10 m.marshal(kontakte, new File("/tmp/kontakte.xml"));
```

Marshalling: Java-Ojekt nach XML

- Marshalling: ähnlich Serialisierung von Objekten; Orignalbedeutung: Einweisen eines Flugzeugs auf dem Flugfeld zur Startbahn durch den Marshall.
- Code-Schnipsel:

```
1  // create JAXB context and instantiate marshaller
2  JAXBContext context = JAXBContext.newInstance(Kontakte.
       class);
3  Marshaller m = context.createMarshaller();
4  m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.
       TRUE);
5
6  // Write to System.out
7  m.marshal(kontakte, System.out);
8
9  // Write to File
10 m.marshal(kontakte, new File("/tmp/kontakte.xml"));
```

Marshalling: Java-Ojekt nach XML

- Marshalling: ähnlich Serialisierung von Objekten; Orignalbedeutung: Einweisen eines Flugzeugs auf dem Flugfeld zur Startbahn durch den Marshall.
- Code-Schnipsel:

```
1  // create JAXB context and instantiate marshaller
2  JAXBContext context = JAXBContext.newInstance(Kontakte.
       class);
3  Marshaller m = context.createMarshaller();
4  m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.
       TRUE);
5
6  // Write to System.out
7  m.marshal(kontakte, System.out);
8
9  // Write to File
10 m.marshal(kontakte, new File("/tmp/kontakte.xml"));
```

Marshalling: Java-Ojekt nach XML

- Marshalling: ähnlich Serialisierung von Objekten; Orignalbedeutung: Einweisen eines Flugzeugs auf dem Flugfeld zur Startbahn durch den Marshall.
- Code-Schnipsel:

```
1  // create JAXB context and instantiate marshaller
2  JAXBContext context = JAXBContext.newInstance(Kontakte.
       class);
3  Marshaller m = context.createMarshaller();
4  m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.
       TRUE);
5
6  // Write to System.out
7  m.marshal(kontakte, System.out);
8
9  // Write to File
10 m.marshal(kontakte, new File("/tmp/kontakte.xml"));
```

Marshalling: Java-Ojekt nach XML

- Marshalling: ähnlich Serialisierung von Objekten; Orignalbedeutung: Einweisen eines Flugzeugs auf dem Flugfeld zur Startbahn durch den Marshall.
- Code-Schnipsel:

```
1  // create JAXB context and instantiate marshaller
2  JAXBContext context = JAXBContext.newInstance(Kontakte.
       class);
3  Marshaller m = context.createMarshaller();
4  m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.
       TRUE);
5
6  // Write to System.out
7  m.marshal(kontakte, System.out);
8
9  // Write to File
10 m.marshal(kontakte, new File("/tmp/kontakte.xml"));
```

Unmarshalling: XML-Dokument nach Java-Objekt

- Unmarshalling: Erzeugung von Objekten der mit `xjc` generierten Klassen. Hier:`Kontakte` und `Kontakte.Person`
- Code Schnipsel:

```
1 private List<Kontakte.Person> personListe;
2
3  JAXBContext context = JAXBContext.newInstance(Kontakte.
       class);
4  Unmarshaller u = context.createUnmarshaller();
5
6  Kontakte kontakteNeu = (Kontakte) u.unmarshal(
7     new File("/home/micha/andereKontakte.xml"));
8
9   personListe = kontakteNeu.getPerson();
```

Unmarshalling: XML-Dokument nach Java-Objekt

- Unmarshalling: Erzeugung von Objekten der mit `xjc` generierten Klassen. Hier:`Kontakte` und `Kontakte.Person`
- Code Schnipsel:

```
1 private List<Kontakte.Person> personListe;
2
3  JAXBContext context = JAXBContext.newInstance(Kontakte.
       class);
4  Unmarshaller u = context.createUnmarshaller();
5
6  Kontakte kontakteNeu = (Kontakte) u.unmarshal(
7     new File("/home/micha/andereKontakte.xml"));
8
9   personListe = kontakteNeu.getPerson();
```

Unmarshalling: XML-Dokument nach Java-Objekt

- Unmarshalling: Erzeugung von Objekten der mit `xjc` generierten Klassen. Hier:`Kontakte` und `Kontakte.Person`
- Code Schnipsel:

```
1 private List<Kontakte.Person> personListe;
2
3  JAXBContext context = JAXBContext.newInstance(Kontakte.
       class);
4  Unmarshaller u = context.createUnmarshaller();
5
6  Kontakte kontakteNeu = (Kontakte) u.unmarshal(
7     new File("/home/micha/andereKontakte.xml"));
8
9   personListe = kontakteNeu.getPerson();
```

Unmarshalling: XML-Dokument nach Java-Objekt

- Unmarshalling: Erzeugung von Objekten der mit `xjc` generierten Klassen. Hier:`Kontakte` und `Kontakte.Person`
- Code Schnipsel:

```
1 private List<Kontakte.Person> personListe;
2
3  JAXBContext context = JAXBContext.newInstance(Kontakte.
       class);
4  Unmarshaller u = context.createUnmarshaller();
5
6  Kontakte kontakteNeu = (Kontakte) u.unmarshal(
7     new File("/home/micha/andereKontakte.xml"));
8
9   personListe = kontakteNeu.getPerson();
```

Unmarshalling: XML-Dokument nach Java-Objekt

- Unmarshalling: Erzeugung von Objekten der mit `xjc` generierten Klassen. Hier:`Kontakte` und `Kontakte.Person`
- Code Schnipsel:

```
1 private List<Kontakte.Person> personListe;
2
3  JAXBContext context = JAXBContext.newInstance(Kontakte.
       class);
4  Unmarshaller u = context.createUnmarshaller();
5
6  Kontakte kontakteNeu = (Kontakte) u.unmarshal(
7     new File("/home/micha/andereKontakte.xml"));
8
9   personListe = kontakteNeu.getPerson();
```

Unmarshalling: XML-Dokument nach Java-Objekt

- Unmarshalling: Erzeugung von Objekten der mit `xjc` generierten Klassen. Hier:`Kontakte` und `Kontakte.Person`
- Code Schnipsel:

```
1 private List<Kontakte.Person> personListe;
2
3  JAXBContext context = JAXBContext.newInstance(Kontakte.
       class);
4  Unmarshaller u = context.createUnmarshaller();
5
6  Kontakte kontakteNeu = (Kontakte) u.unmarshal(
7     new File("/home/micha/andereKontakte.xml"));
8
9   personListe = kontakteNeu.getPerson();
```

Unmarshalling: XML-Dokument nach Java-Objekt

- Unmarshalling: Erzeugung von Objekten der mit `xjc` generierten Klassen. Hier:`Kontakte` und `Kontakte.Person`
- Code Schnipsel:

```
1 private List<Kontakte.Person> personListe;
2
3  JAXBContext context = JAXBContext.newInstance(Kontakte.
       class);
4  Unmarshaller u = context.createUnmarshaller();
5
6  Kontakte kontakteNeu = (Kontakte) u.unmarshal(
7     new File("/home/micha/andereKontakte.xml"));
8
9   personListe = kontakteNeu.getPerson();
```

Unmarshalling: XML-Dokument nach Java-Objekt

- Unmarshalling: Erzeugung von Objekten der mit `xjc` generierten Klassen. Hier:`Kontakte` und `Kontakte.Person`
- Code Schnipsel:

```
1 private List<Kontakte.Person> personListe;
2
3  JAXBContext context = JAXBContext.newInstance(Kontakte.
       class);
4  Unmarshaller u = context.createUnmarshaller();
5
6  Kontakte kontakteNeu = (Kontakte) u.unmarshal(
7     new File("/home/micha/andereKontakte.xml"));
8
9   personListe = kontakteNeu.getPerson();
```

Unmarshalling: XML-Dokument nach Java-Objekt

- Unmarshalling: Erzeugung von Objekten der mit `xjc` generierten Klassen. Hier:`Kontakte` und `Kontakte.Person`
- Code Schnipsel:

```
1 private List<Kontakte.Person> personListe;
2
3  JAXBContext context = JAXBContext.newInstance(Kontakte.
       class);
4  Unmarshaller u = context.createUnmarshaller();
5
6  Kontakte kontakteNeu = (Kontakte) u.unmarshal(
7     new File("/home/micha/andereKontakte.xml"));
8
9   personListe = kontakteNeu.getPerson();
```

Unmarshalling: XML-Dokument nach Java-Objekt

- Unmarshalling: Erzeugung von Objekten der mit `xjc` generierten Klassen. Hier:`Kontakte` und `Kontakte.Person`
- Code Schnipsel:

```
1 private List<Kontakte.Person> personListe;
2
3  JAXBContext context = JAXBContext.newInstance(Kontakte.
       class);
4  Unmarshaller u = context.createUnmarshaller();
5
6  Kontakte kontakteNeu = (Kontakte) u.unmarshal(
7     new File("/home/micha/andereKontakte.xml"));
8
9   personListe = kontakteNeu.getPerson();
```

Unmarshalling: XML-Dokument nach Java-Objekt

- Unmarshalling: Erzeugung von Objekten der mit `xjc` generierten Klassen. Hier:`Kontakte` und `Kontakte.Person`
- Code Schnipsel:

```
1 private List<Kontakte.Person> personListe;
2
3  JAXBContext context = JAXBContext.newInstance(Kontakte.
       class);
4  Unmarshaller u = context.createUnmarshaller();
5
6  Kontakte kontakteNeu = (Kontakte) u.unmarshal(
7     new File("/home/micha/andereKontakte.xml"));
8
9   personListe = kontakteNeu.getPerson();
```

Danksagung

Vielen Dank für’s Zuhören und Mitmachen!