DAIMLERCHRYSLER

Tutorial: XML in der Praxis

Mario Jeckle
DaimlerChrysler Forschungszentrum Ulm
mario.jeckle@daimlerchrysler.com
mario@jeckle.de
www.jeckle.de

XML in der Praxis

I Einführendes

- XML-Historie
- HTML, SGML und XML
- Anatomie eines XML-Dokuments
- Grammatiken und Vokabulare
- Namespaces

II *Under the hood...*

Entwurf von XML-Sprachen

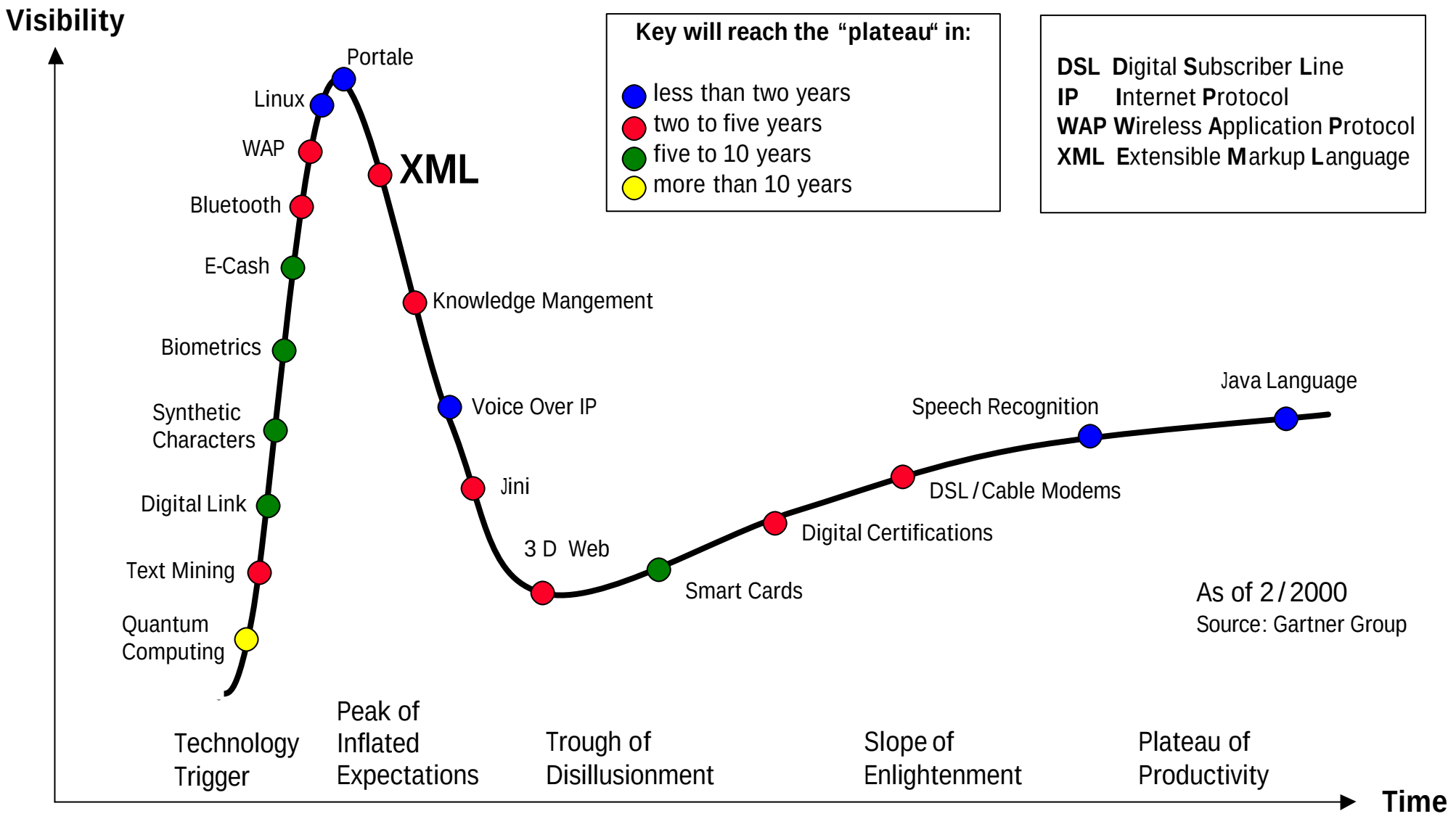
- XML Schema
- XMI generation rules

III *putting it to into practice...*

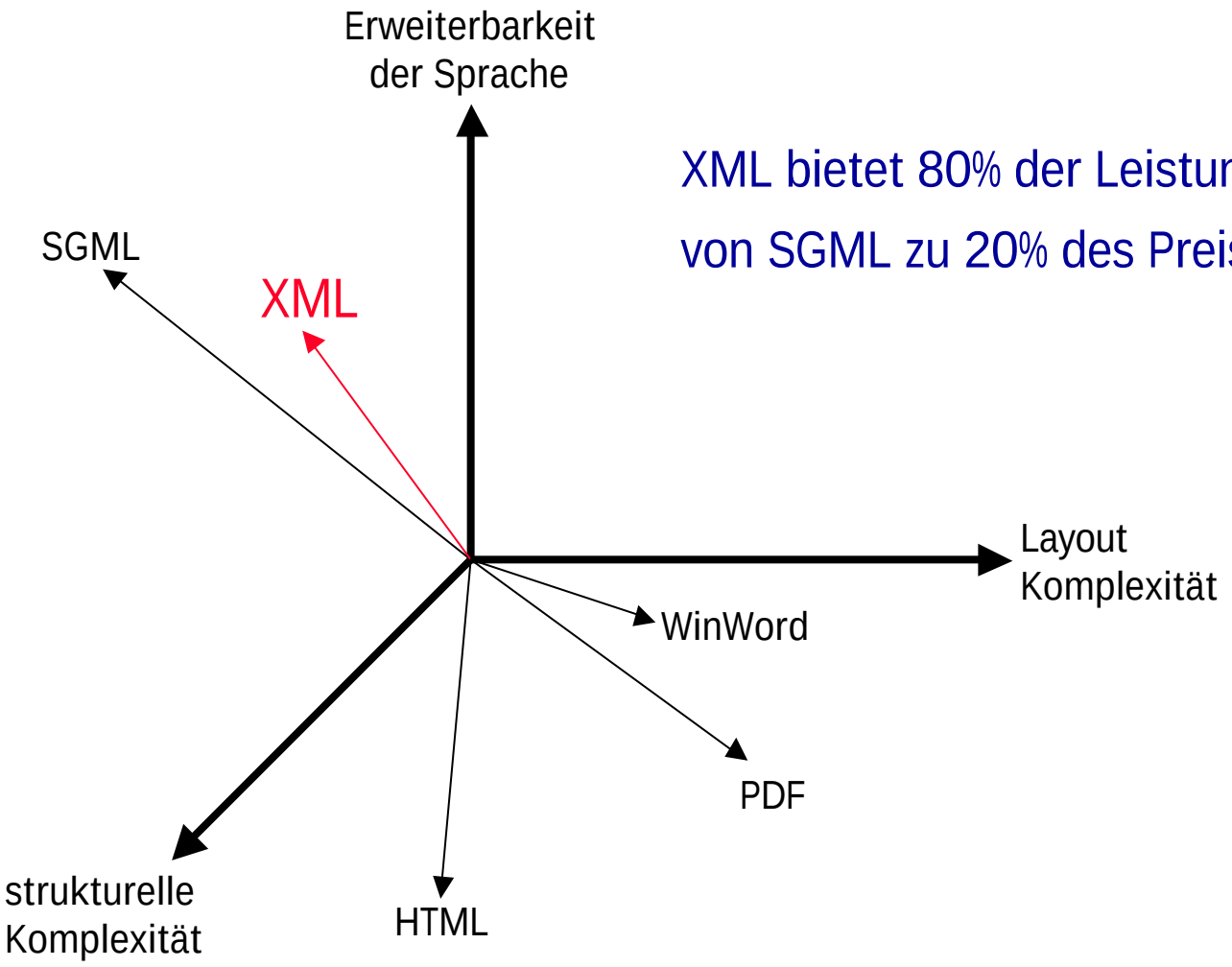
- Metadatenaustausch mit dem XML Metadata Interchange Format (XMI)

IV Fallstudie

XML im Hype Life Cycle



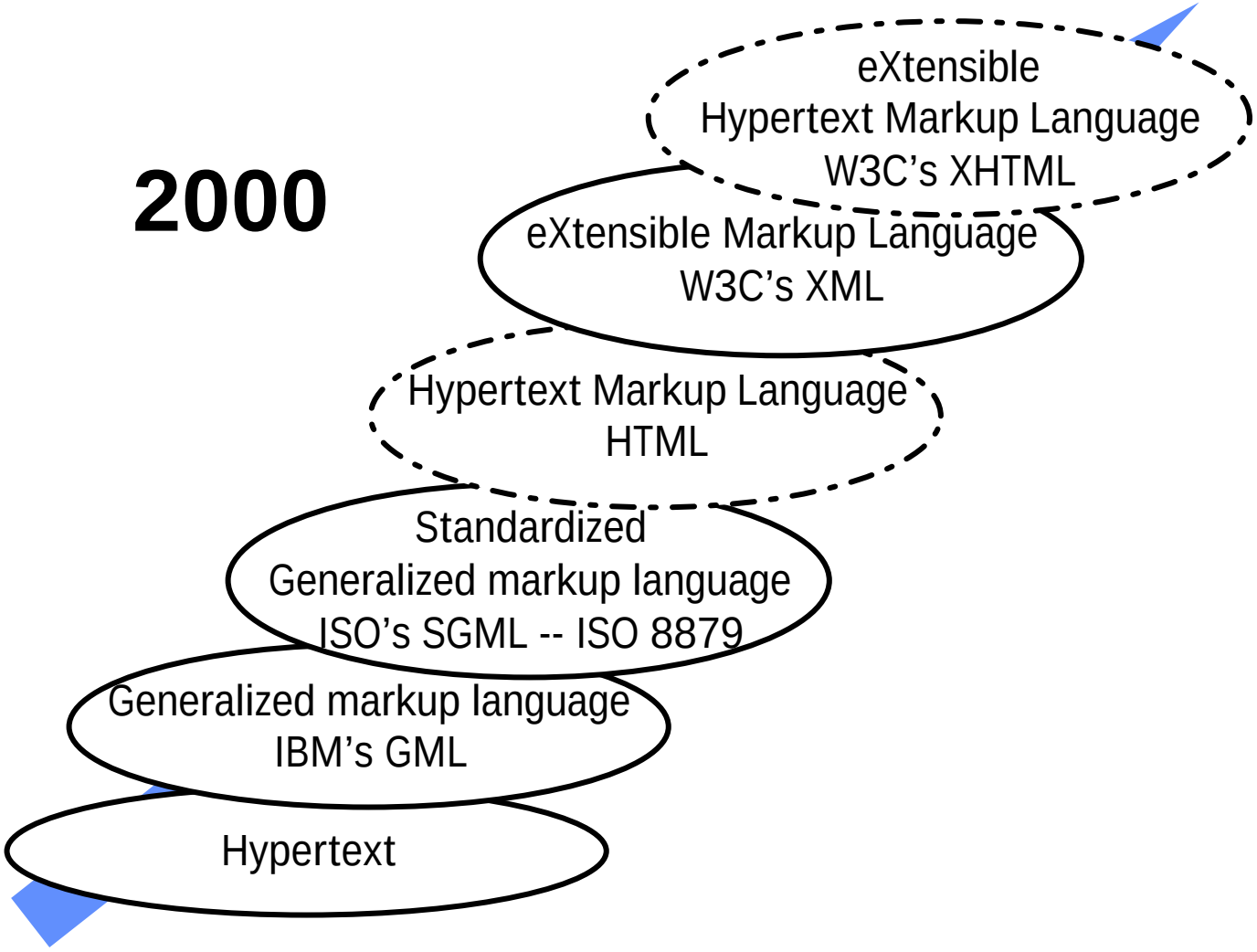
XML poor man‘s SGML?



XML bietet 80% der Leistungsfähigkeit

von SGML zu 20% des Preises

XML-Historie



HTML vs. XML


```
<html>
<head><title>jeckle.de</title></head>
<body bgcolor="#003366" topmargin="0" leftmargin="0"  marginwidth="0" marginheight="0">
<table width="100%" border="0" cellspacing="0" cellpadding="0" height="65%" bgcolor="#f0f0f0">
  <tr>
    <td valign="bottom"> </td>
    <td colspan="2" valign="bottom" align="center">
      <table width="800" border="0" cellspacing="0" cellpadding="0" align="center" height="315">
        <tr bgcolor="#f0f0f0">
          <td height="25"> </td>        <td height="25"> </td>
          <td height="25"> </td>        <td height="25"> </td>
          <td height="25"> </td>        <td height="25"> </td>
          <td height="25"> </td>        <td height="25"> </td>
...
```

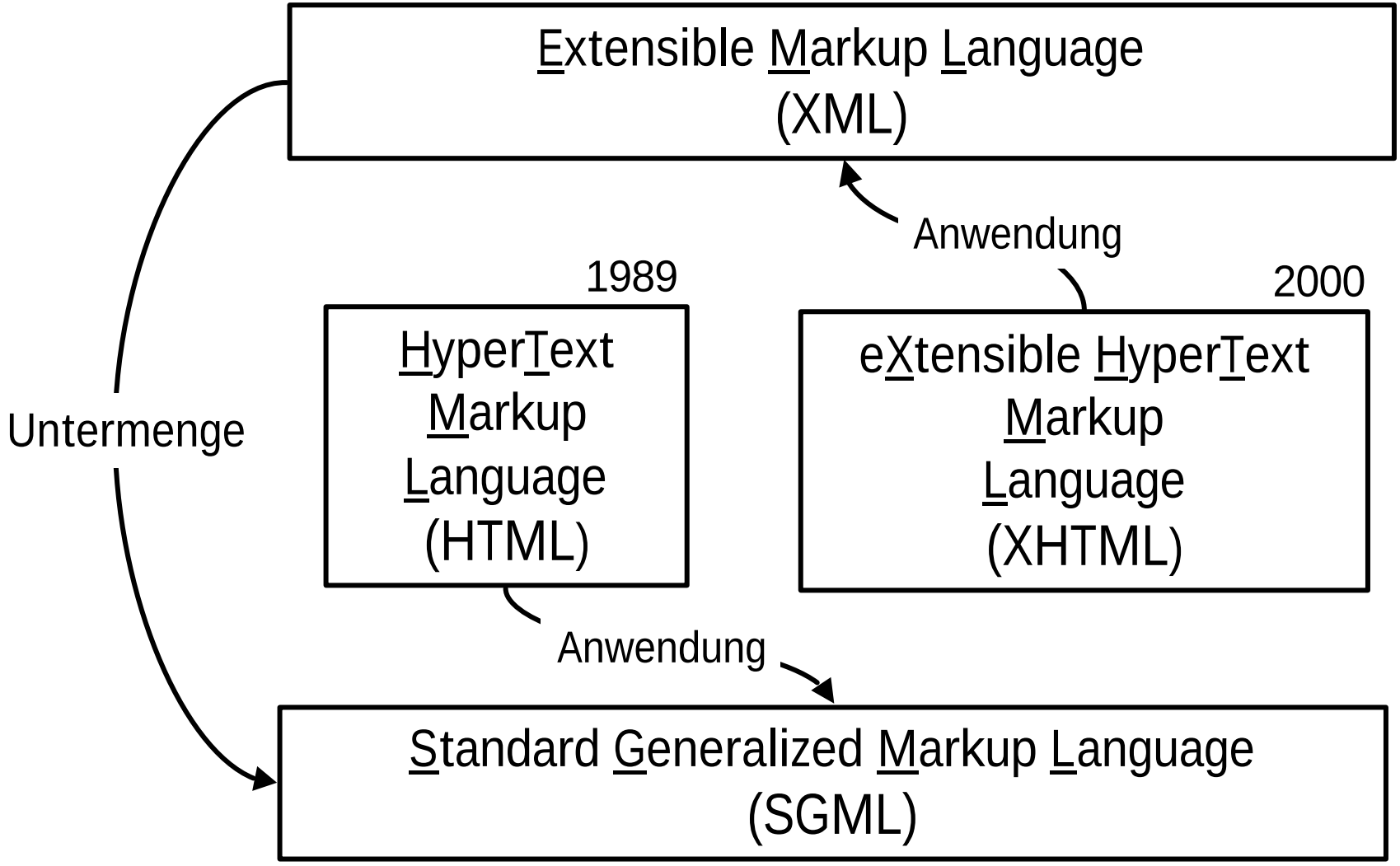
Tim Berners-Lee

Beginn Fettdruck (*bold*)

Ende Fettdruck

- HTML legt hauptsächlich das Präsentationsverhalten fest
- XML definiert die syntaktische Struktur der Information

HTML, SGML und XML



Anatomie eines XML-Dokuments

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

Start Tag

```
<vortrag datum="2000-10-09">
```

Tag Name

Element

Attribut Name

Attribut Wert

Attribut

```
</vortrag>
```

End Tag

Anatomie eines XML-Dokuments

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

Start Tag

```
<vortrag datum="2000-10-09">
    <autor>
        <vorname>Mario</vorname>
        <nachname>Jeckle</nachname>
    </autor>
</vortrag>
```

Element

End Tag

Grammatiken und Vokabulare

XML Dokument

```
<?xml version = "1.0"?>
<!DOCTYPE vortrag SYSTEM "vortrag.dtd">
<vortrag datum="2000-10-09">
        <abstract>
        XML in der Praxis ....
        </abstract>
</vortrag>
```

XML Document Type Definition (DTD)

```
<!ELEMENT vortrag (abstract)>
<!ATTLIST vortrag
        datum CDATA #REQUIRED>
<!ELEMENT abstract ANY>
```

Grammatiken und Vokabulare

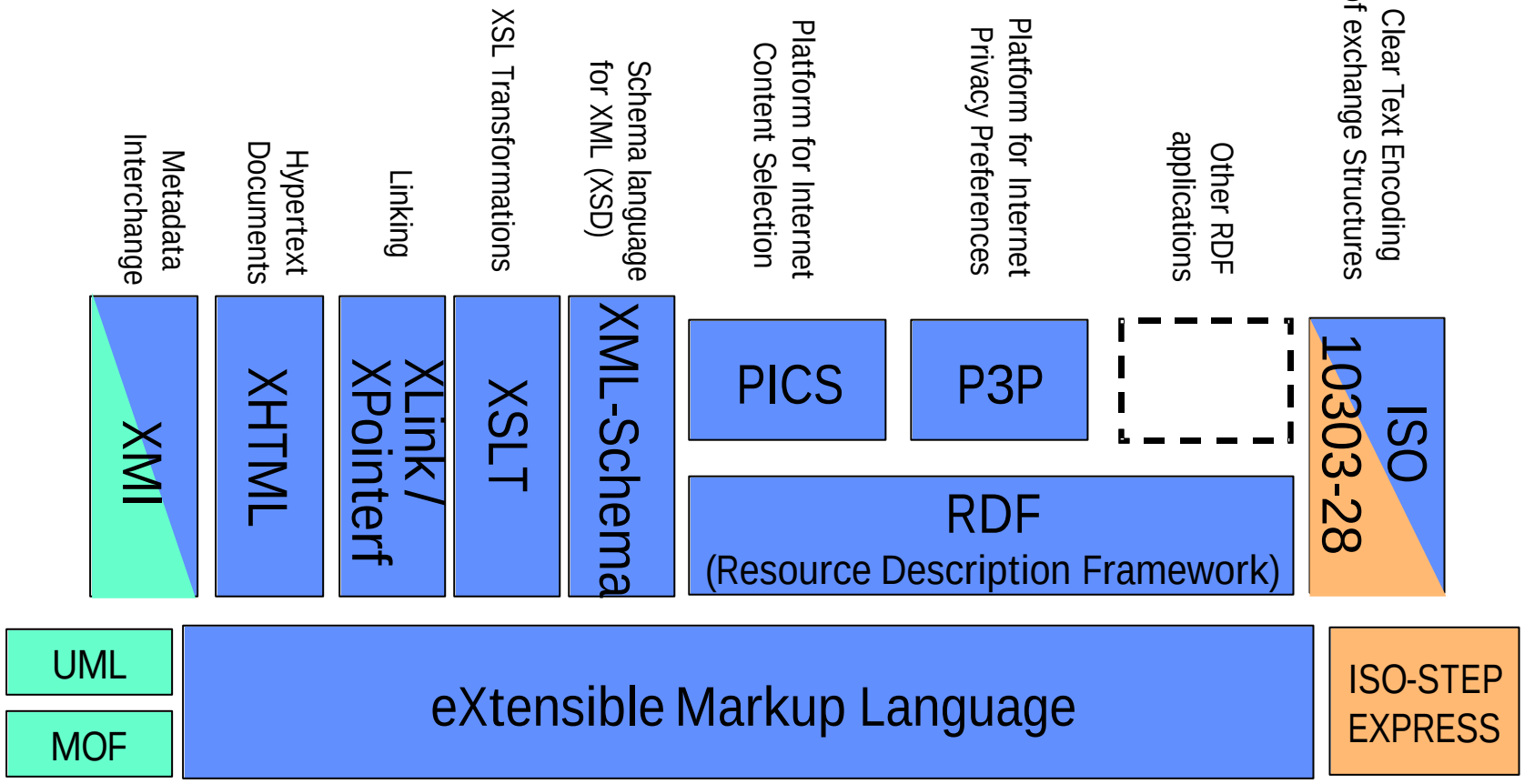
Die (vergleichsweise) einfache XML-Sprachdefinition hat zur inflationären Vermehrung der XML-Sprachen geführt.

Teilweise existieren sogar direkt miteinander konkurrierende Vorschläge für dieselbe Anwendungsdomäne.

->Semantik beachten

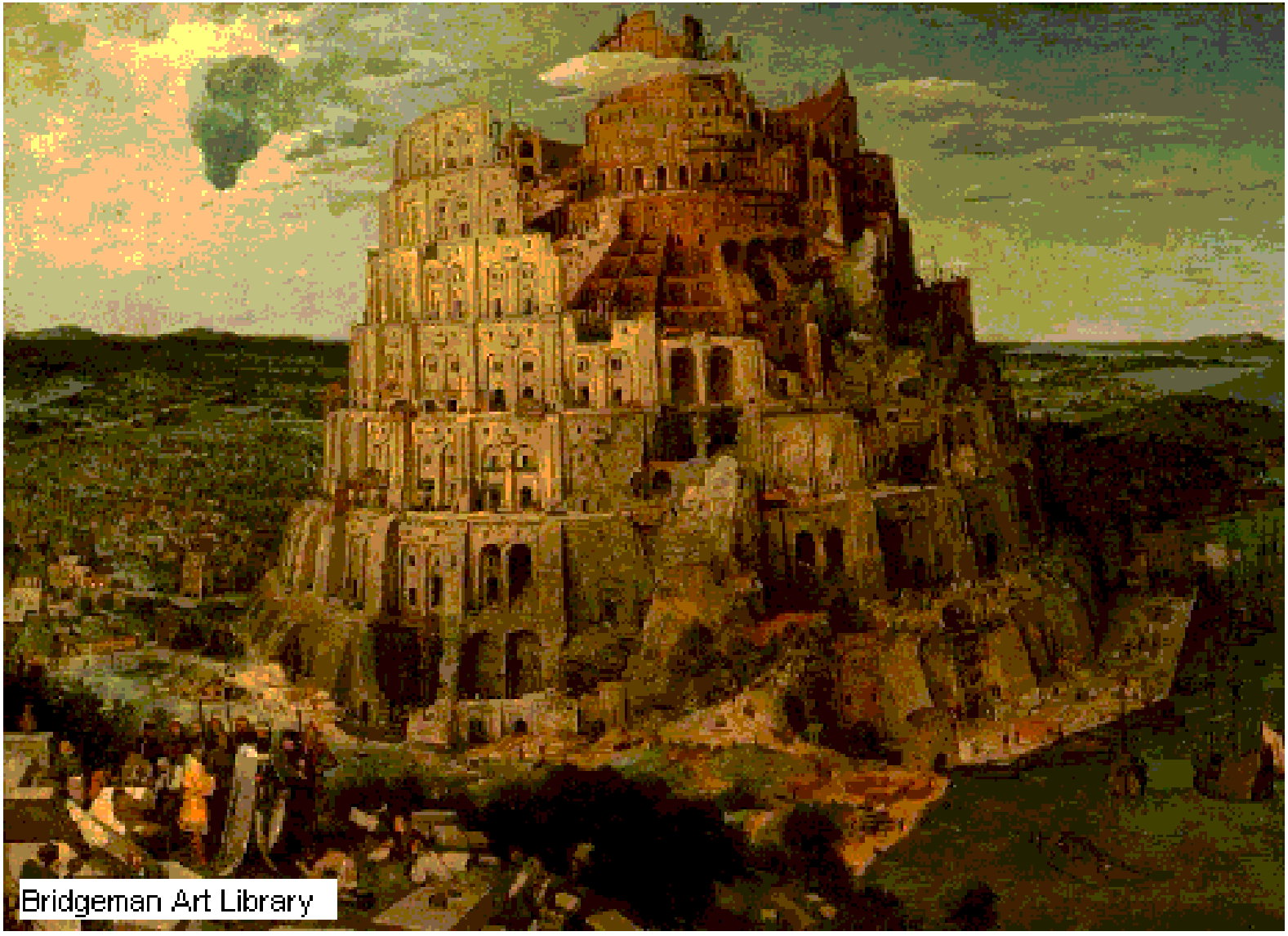
-> an Standards orientieren

XML-Sprachfamilie



Die XML ist eine Sprache zur Sprachdefinition (eine *Metasprache*).
Mit ihr definierte Sprachen bilden die XML-Sprachfamilie.

Grammatiken und Vokabulare



Grammatiken und Vokabulare – Interoperabilitätsaspekte

XML Namespaces

```
<rechnung>
  <kunde>
    <name>
        <vorname>Erika</vorname>
        <nachname>Mustermann</nachname>
    </name>
...
  </kunde>
...
</rechnung>
```

```
<produkt>
  <name>
    <marke>XY</marke>
    <id>BlitzBlank</id>
  </name>
  ...
</produkt>
```

Namespaces ermöglichen das Zusammenwirken verschiedener XML-Sprachen in einem Dokument. Sie beseitigen etwaige Namens-konflikte auf Tag-Ebene.

Grammatiken und Vokabulare – Interoperabilitätsaspekte
XML Namespaces

```
<xyz:rechnung>
  <xyz:kunde>
    <xyz:name>
        <xyz:vorname>Erika</xyz:vorname>
        <xyz:nachname>Mustermann</xyz:nachname>
    </xyz:name>
...
  </xyz:kunde>
...
</xyz:rechnung>
```

```
<abc:produkt >
  <abc:name>
    <abc:marke>XY
    </abc:marke>
    <abc:id>BlitzBlank
    </ abc:id>
  </ abc:name>
  ...
</abc:produkt>
```

Grundidee der Namespaces:
Ergänzung der Tagnamen und eindeutigen Schlüsselanteil

Grammatiken und Vokabulare – Interoperabilitätsaspekte
XML Namespaces

```
<rechnung xmlns="http://xyz.com">
  <kunde>
    <name>
        <vorname>Erika</vorname>
        <nachname>Mustermann</nachname>
    </name>
...
  </kunde>
...
</rechnung>
```

```
<produkt xmlns="http://abc.com">
  <name>
    <marke>XY</marke>
    <id>BlitzBlank</id>
  </name>
  ...
</produkt>
```

"On the web == has an URI" (Tim Berners-Lee)

Nutzung des *Uniform Resource Location* (URI) Mechanismus zur eindeutigen Identifikation der Elementtyp-Herkunft.

Grammatiken und Vokabulare – Interoperabilitätsaspekte

XML Namespaces – Namespace trouble

```
<MakingTrouble xmlns="abc.com">
        <noProblem/>
        <TSNH xmlns="#fragment1"/>
</MakingTrouble>

<TroubleMaker xmlns="xyz.org">
        <noProbelm/>
        <TSNH xmlns="#fragment2"/>
</TroubleMaker>
```

Namespace: abc.com

Grammatiken und Vokabulare – Interoperabilitätsaspekte

XML Namespaces – Namespace trouble

```
<MakingTrouble xmlns="abc.com">
        <noProblem/>
        <TSNH xmlns="#fragment1"/>
</MakingTrouble>

<TroubleMaker xmlns="xyz.org">
        <noProbelm/>
        <TSNH xmlns="#fragment2"/>
</TroubleMaker>
```

Namespace: abc.com#fragment1

Grammatiken und Vokabulare – Interoperabilitätsaspekte

XML Namespaces – Namespace trouble

```
<MakingTrouble xmlns="abc.com">
        <noProblem/>
        <TSNH xmlns="#fragment1"/>
</MakingTrouble>

<TroubleMaker xmlns="xyz.org">
        <noProbelm/>
        <TSNH xmlns="#fragment2"/>
</TroubleMaker>
```

Namespace: xyz.org

Grammatiken und Vokabulare – Interoperabilitätsaspekte

XML Namespaces – Namespace trouble

```
<MakingTrouble xmlns="abc.com">
        <noProblem/>
        <TSNH xmlns="#fragment1"/>
</MakingTrouble>

<TroubleMaker xmlns="xyz.org">
        <noProbelm/>
        <TSNH xmlns="#fragment2"/>
</TroubleMaker>
```

Namespace: xyz.org#fragment2

Grammatiken und Vokabulare – Interoperabilitätsaspekte

XML Namespaces – Namespace trouble

```
<MakingTrouble xmlns="abc.com">
        <noProblem/>
        <TSNH xmlns="#fragment1"/>
</MakingTrouble>

<TroubleMaker xmlns="xyz.org">
        <noProbelm/>
        <TSNH xmlns="#fragment2"/>
</TroubleMaker>
```

```
<MakingTrouble xmlns="abc.com">
        <noProblem/>
        <TSNH xmlns="#fragment2"/>
</MakingTrouble>

<TroubleMaker xmlns="xyz.org">
        <noProblem/>
        <TSNH xmlns="#fragment1"/>
</TroubleMaker>
```

Grammatiken und Vokabulare – Interoperabilitätsaspekte

XML Namespaces – Namespace trouble

```
<MakingTrouble xmlns="abc.com">
        <noProblem/>
        <TSNH xmlns="#fragment1"/>
</MakingTrouble>

<TroubleMaker xmlns="xyz.org">
        <noProbelm/>
        <TSNH xmlns="#fragment2"/>
</TroubleMaker>
```

```
<MakingTrouble xmlns="abc.com">
        <noProblem/>
        <TSNH xmlns="#fragment2"/>
</MakingTrouble>

<TroubleMaker xmlns="xyz.org">
        <noProblem/>
        <TSNH xmlns="#fragment1"/>
</TroubleMaker>
```

Namespace: abc.com ✓

Grammatiken und Vokabulare – Interoperabilitätsaspekte

XML Namespaces – Namespace trouble

```
<MakingTrouble xmlns="abc.com">
        <noProblem/>
        <TSNH xmlns="#fragment1"/>
</MakingTrouble>

<TroubleMaker xmlns="xyz.org">
        <noProbelm/>
        <TSNH xmlns="#fragment2"/>
</TroubleMaker>
```

```
<MakingTrouble xmlns="abc.com">
        <noProblem/>
        <TSNH xmlns="#fragment2"/>
</MakingTrouble>

<TroubleMaker xmlns="xyz.org">
        <noProblem/>
        <TSNH xmlns="#fragment1"/>
</TroubleMaker>
```

Namespace: xyz.org

Grammatiken und Vokabulare – Interoperabilitätsaspekte

XML Namespaces – Namespace trouble

```
<MakingTrouble xmlns="abc.com">
        <noProblem/>
        <TSNH xmlns="#fragment1"/>
</MakingTrouble>

<TroubleMaker xmlns="xyz.org">
        <noProbelm/>
        <TSNH xmlns="#fragment2"/>
</TroubleMaker>
```

```
<MakingTrouble xmlns="abc.com">
        <noProblem/>
        <TSNH xmlns="#fragment2"/>
</MakingTrouble>

<TroubleMaker xmlns="xyz.org">
        <noProblem/>
        <TSNH xmlns="#fragment1"/>
</TroubleMaker>
```

Namespace: abc.com#fragment2

Grammatiken und Vokabulare – Interoperabilitätsaspekte

XML Namespaces – Namespace trouble

```
<MakingTrouble xmlns="abc.com">
        <noProblem/>
        <TSNH xmlns="#fragment1"/>
</MakingTrouble>

<TroubleMaker xmlns="xyz.org">
        <noProbelm/>
        <TSNH xmlns="#fragment2"/>
</TroubleMaker>
```

```
<MakingTrouble xmlns="abc.com">
        <noProblem/>
        <TSNH xmlns="#fragment2"/>
</MakingTrouble>

<TroubleMaker xmlns="xyz.org">
        <noProblem/>
        <TSNH xmlns="#fragment1"/>
</TroubleMaker>
```

Namespace: xyz.org#fragment1

Entwurf von XML Sprachen – XML Schema

Zum Begriff *Schema*

[lat.-griech. *Haltung, Stellung; Gestalt, Figur, Form*]
(allgemein): Muster, Entwurf, Aufriss; Konzept.
(Philosophie und Logik):

1) Wissenschafts- und verfahrensmethodologisch werden Schemata bewusst und planmäßig in Form einer zielgerichteten Ordnung von Begriffen und Denkabläufen, als hypothetisches Konstrukt zur Erklärung von Zusammenhängen (z.B. *soziales Schema*) sowie Verfahrensplan (v.a. bei wiederkehrenden Situationen) angewendet.
2) Erkenntnistheoretisch hat nach I.Kant jeder reine Begriff (Kategorie) sein Schema, d.h. einen inneren Vorgang, zu durchlaufen, mit dem er sich durch eine stellvertretende Vorstellung anschaulich realisiert. Die Verstandesbegriffe haben je ein transzendentales Schema, wodurch ihnen erste eine Beziehung zum Sinnlichen gegeben wird. Der *Schematismus* ist somit das Verfahren des Verstandes, einem Begriff objektive Realität durch die ihm entsprechende Anschauung zu verschaffen.

[aus: Brockhaus -- Die Enzyklopädie, Mannheim, 1999, Bd. 19, S. 274]

Entwurf von XML Sprachen – XML Schema

Zum Begriff *Schema* in der Informatik

(allgemein) **Ein *Schema* beschreibt eine Datenstruktur**.
Schemata entstehend i.A. als Ergebnis eines (Informations- / Daten-) Modellierungsprozesses.
Typischerweise spricht man im Datenbank-Umfeld vom Schema, im Sinne der Struktur- und Inhaltsbeschreibung, der in der Datenbank abgelegten Daten.
In diesem Kontext werden auch verschiedene *Schematypen* (wie *konzeptuelles, logisches, internes* und *externes S*.) unterschieden.

Entwurf von XML Sprachen – XML Schema

Zum Begriff *Schema* im Kontext *XML*

Das Schema einer XML-Datei wird normgemäß durch eine *Document Type Definition* (DTD), gebildet.

Oftmals werden die Begriffe *Schema* und *Grammatik* im Umfeld XML synonym gebraucht, dies trifft jedoch nicht zu.

Während die *Grammatik* -- als Syntaxbeschreibung -- eins SGML-, und damit auch XML-, Dokuments absolut benötigt wird, um die Dokumentstruktur zu verstehen, definiert ein *Schema* Regeln und Einschränkungen an die logische Dokumentstruktur.

Als Konsequenz der fixierten Dokumentsyntax von XML (in SGML kann diese variieren) wird durch eine XML-DTD lediglich der semantische Anteil der Dokumentbeschreibung realisiert.

Es sind jedoch beliebige Beschreibungssprachen konkreter XML-Dokument-Ausprägungen denkbar.

XML-Schema ist eine XML-Sprache zur Beschreibung beliebiger XML-Dokumentinhalte.

Entwurf von XML Sprachen – XML Schema

Well-formedness

XML Dokument

```
<?xml version="1.0" encoding="UTF8" ?>
<!DOCTYPE ProjektAbwicklung SYSTEM "projekt.dtd">

<ProjektAbwicklung>
<person persID='P100' gehaltsGrp="2">
  <vorname>Hans</vorname>
  <vorname>Georg</vorname>
  <name>Meier</name>
  <projektLeiter prjRef='F300'/>
  <projektMitarb prjRef='F310'/>
</person>
<projekt prjID='F300' start='1.1.2000' budget='1000'/>
<projekt prjID='F310' budget='10000'/>
<projekt prjID='F320' budget='25000'/>
</ProjektAbwicklung>
```

Dokument...

- Enthält mindestens ein Element
- Baumartige Organisation (Wurzelelement existiert)
- Korrekte Elementschachtelung
- Übereinstimmende öffnende und schließende Tag-Namen
- Attributnamen sind innerhalb Elementen eindeutig
- Attributwerte sind in doppelten Anführungszeichen
- Attributwerte verweisen nicht auf externe Entities
- Keine Markupsymbole in Attributwerten
- Entity-Deklaration vor Referenzierung
- Keine Entity-Referenz verweist auf unparsed Entity

Entwurf von XML Sprachen – XML Schema

Validness

XML Dokument

```
<?xml version="1.0" encoding="UTF8" ?>
<!DOCTYPE ProjektAbwicklung SYSTEM "projekt.dtd">

<ProjektAbwicklung>
<person persID='P100' gehaltsGrp="2">
  <vorname>Hans</vorname>
  <vorname>Georg</vorname>
  <name>Meier</name>
  <projektLeiter prjRef='F300'/>
  <projektMitarb prjRef='F310'/>
</person>
<projekt prjID='F300' start='1.1.2000' budget='1000'/>
<projekt prjID='F310' budget='10000'/>
<projekt prjID='F320' [illegible]t='25000'/>
</ProjektAbwicklu[illegible]
```

Validates

XML Docume[illegible]ype Definition

```
<?xml version="1.0" e[illegible]F-8" ?>

<!ELEMENT ProjektAb[illegible] (person+, projekt*)>
<!ELEMENT person (vorname+, name, projektLeiter?,
                  projektMitarb*)>
<!ATTLIST person   persID   ID  #REQUIRED
      gehaltsGrp   (1|1a|2) "1">
<!ELEMENT vorname     (#PCDATA)>
<!ELEMENT name        (#PCDATA)>
<!ELEMENT projektLeiter  EMPTY>
<!ATTLIST projektLeiter
       prjRef      IDREF  #REQUIRED>

<!ELEMENT projektMitarb EMPTY>
<!ATTLIST projektMitarb
      prjRef       IDREF #REQUIRED>
```

Ein XML-Dokument ist gültig *(valid)*, wenn

- eine DTD existiert
  (und referenziert oder eingebunden ist)
- die durch die DTD formulierten Struktur- und Inhaltsregeln eingehalten sind

Entwurf von XML Sprachen – XML Schema

Validness

Allein die Existenz und Referenzierung oder Inkludierung einer DTD rechtfertigt es nicht, von einem gültigen XML-Dokument zu sprechen.

Die Klasse der nicht validierenden Parser läßt per definitionem jegliche DTD beim Prüfungsvorgang außer Acht.

Auch kann ein Parser dieses Typs beliebige Teile der DTD (beispielsweise *extenal subsets*) ignorieren.

Entwurf von XML Sprachen – XML Schema

Validness

Die Begriffe *well-formed* und *valid* stammen von Tim Bray.
Trotz des Einwands durch Sperberg Mc-Queen, beide Begriffe wären bereits in anderen Kontexten vorbelegt, überlebte der Vorschlag -- nicht zuletzt in Ermangelung eines Besseren -- bis heute.

Der Begriff *well-formed* steht in keinem Bezug zur gleichen Bezeichnung in der formalen mathematischen Logik.

In SGML sind für *well-formed* bzw. *valid* die Begriffe *tag-valid* bzw. *type-valid* gebräuchlich.

Ein Dokument, für das nicht der Anspruch erhoben wird es sei *valid* hat:

- keine DTD (weder *internal subset* noch DOCTYPE-Referenz)
- keine vorliegende DTD (z.B. zugänglich über Netz, aber nicht lokal repliziert)
- eine vorliegende DTD, die jedoch nicht zur Validierung herangezogen wird

Entwurf von XML Sprachen – XML Schema

Der DTD-Mechanismus in XML v1.0

- Dokument ist streng hierarchisch gegliedert
- *ELEMENTs* als innere Knoten
- *ATTLISTs* zur Attributierung der Knoten
- Keine Datentypen (abgesehen von CHAR-Data)
- Rudimentärer Referenzierungsmechanismus (ID, IDREF(S))
- Selektionstyp (enum)
- Vorgabewerte
- DTD ist nicht XML

=> Notwendige Konstrukte zum Ausdruck mächtigerer Semantik müssen aufwendig und proprietär realisiert werden

Entwurf von XML Sprachen – XML Schema

Motivation Schema-Mechanismen für XML

Gemeinsames Vokabular

- ermöglicht Kommunikation und Interaktion auf Basis einheitlich definierter Begriffe.
  Daher ist neben der strukturellen Definition eine Semantikfestlegung unabdingbar

Formale Beschreibung

- Grundvoraussetzung maschineller Verarbeitung
- Idealerweise (vergleichsweise) einfache Verarbeitung
  (schlanke, eindeutige Definitionen, möglichst kontextfreie oder reguläre Sprache)

Austauschbasis

- Explizite Strukturdefinition ist Grundvoraussetzung des Informationsaustauschs.

Entwurf von XML Sprachen – XML Schema

Anforderungen an einen XML-Schemamechanismus

Structural schemas

- Mächtigkeit analog des bestehenden DTD-Mechanismus um Dokumentstruktur (Reihenfolge, Auftrittsvielfachheit von Elementen und Attribute) zu beschreiben
  Insbesondere sollen folgende Erweiterungen verwirklicht werden:
  - Namespace Integration
  - Definition von Einschränkungen für Elementinhalte
  - Integration Strukturschema und primitive Datentypen
  - Vererbung: DTD unterstützt nur *kind-of*-Beziehungen
  - Erweiterter Referenzierungsmechanismus (URI)

Entwurf von XML Sprachen – XML Schema

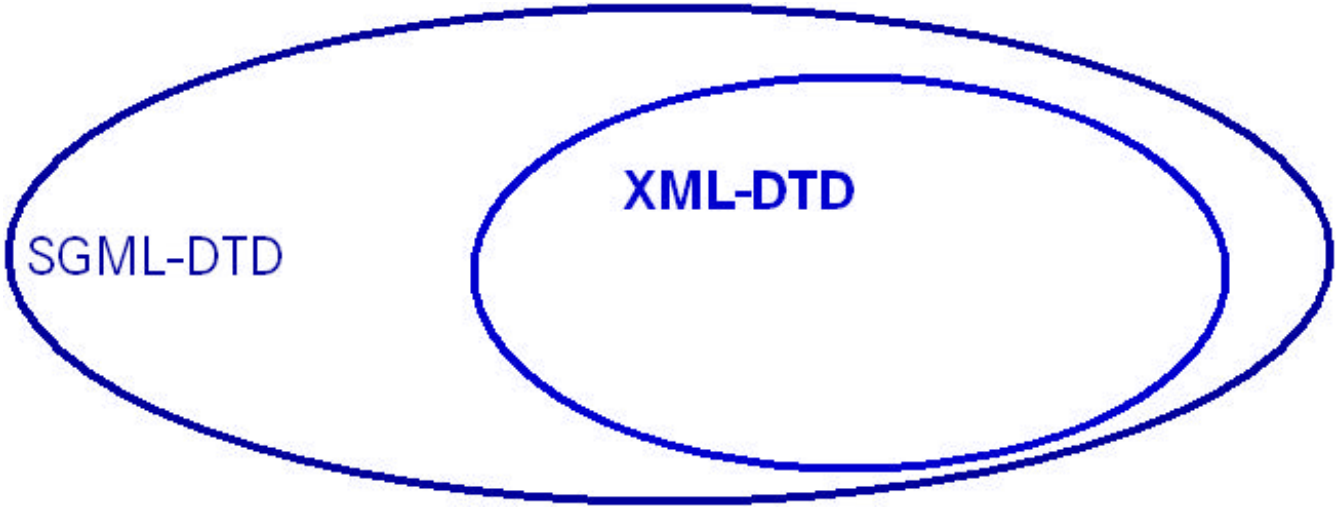
Anforderungen an einen XML-Schemamechanismus

Primitive Datentypen

- „klassische“ atomare Datentypen, ergänzt um SQL-artige, wie *integer*, *date.*
- Programmiersprachen-übliche (typischerweise Java-artige) *build-in types*
- uninterpretierte Binärstrukturen
- (durch Anwender) erweiterbares Typsystem
- lexikalische Definitionen
- Einschränkungen an Typen

Entwurf von XML Sprachen – XML Schema

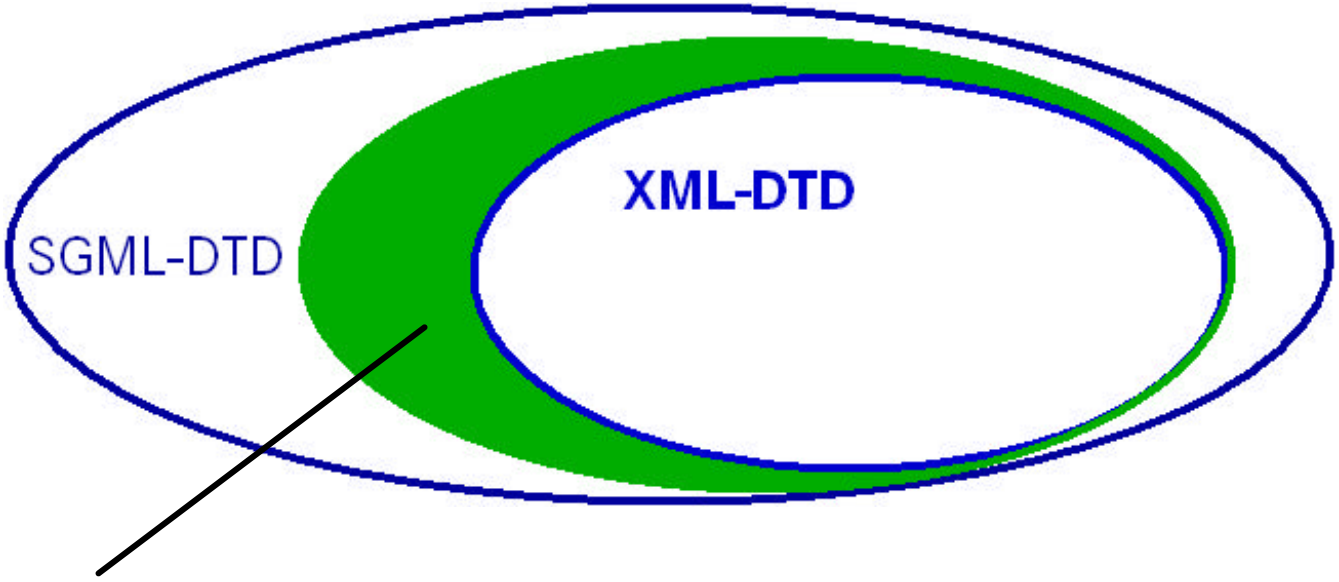
Entwicklungsoptionen ...



Ausdrucksmächtigkeit der XML-DTD bildet eine (echte) Untermenge des SGML-DTD Mechanismus.

Entwurf von XML Sprachen – XML Schema

Entwicklungsoptionen ...



Erweiterung des XML-DTD Mechanismus um weitere Elemente der SGML-DTD.

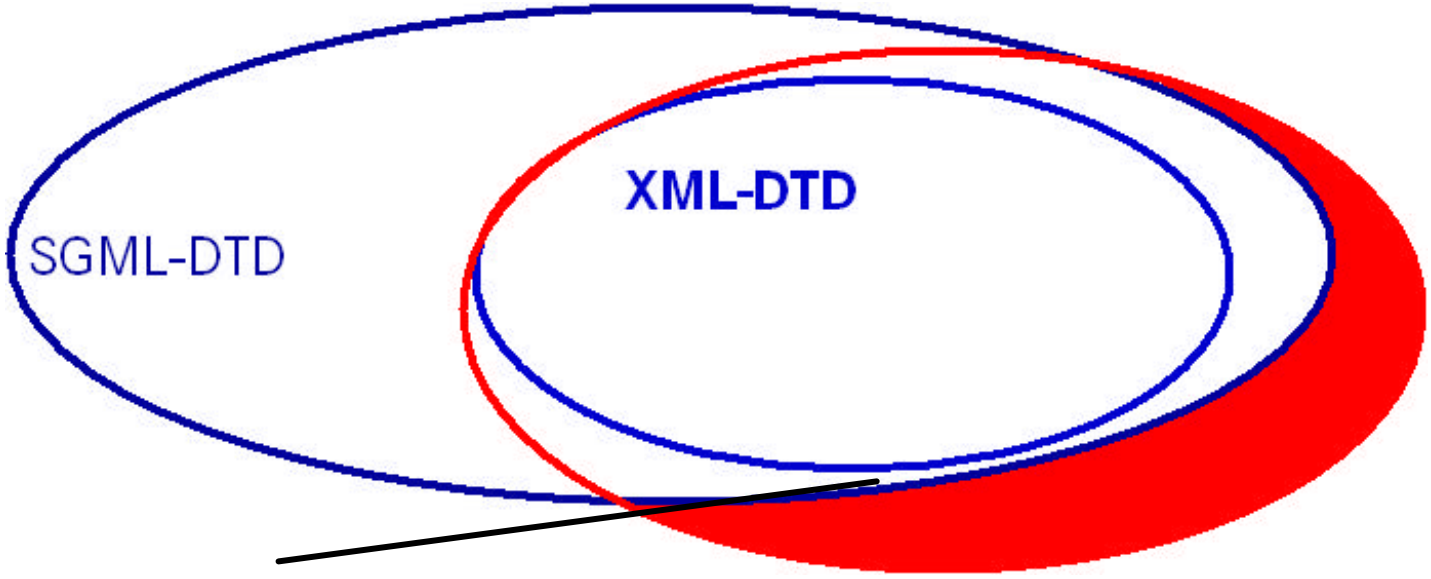
(+) Ausdrucksmächtigkeit nähert sich (wieder) der von SGML an

(-) ...die Komplexität auch

(-) Ausdrucksmächtigkeit kann die von SGML niemals übertreffen

Entwurf von XML Sprachen – XML Schema

Entwicklungsoptionen ...



Erweiterung des XML-DTD Mechanismus um Elemente, die *nicht* mit SGML-Mitteln ausdrückbar sind.

(+) Freiheitsgrad hinsichtlich beliebiger Erweiterungen

(-) XML-Grundforderung nach Untermengenbeziehung zu SGML entfällt

(-) immernoch zwei verschiedene Sprachen für Inhalt und Schema

Entwurf von XML Sprachen – XML Schema

Schemadialekte

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

- Data Types for DTD (DT4DTD)

Wissensbeschreibung

- Document Content Description for XML (DCD)

  (*RDF basierte Weiterentwicklung von XML-Data*)

Inspiriert durch XML-API-Entwicklung

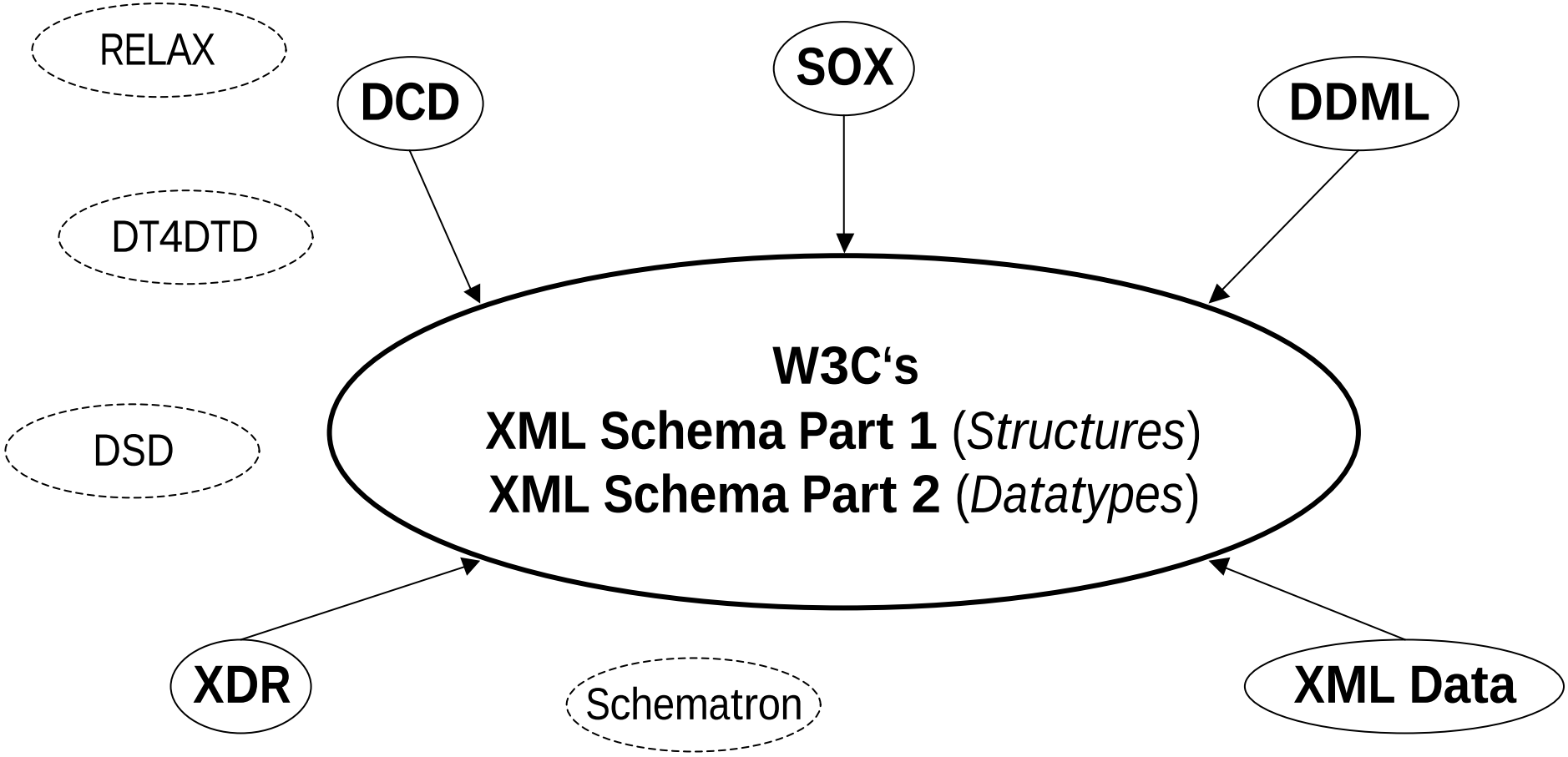
- Schema for Object oriented XML (SOX)

XML-Sprachen zur Inhaltsbeschreibung

- Document Definition Markup Language/XSchema (DDML)
- Schematron (XSL-basierte Auswertung der Dokumentstruktur)
- XML-Data/XML-Data Reduced (XDR) *(erster Ansatz noch vor Verabschiedung XML 1.0)*
- Document Structure Description (DSD)
- **W3C's XML-Schema**

Entwurf von XML Sprachen – XML Schema

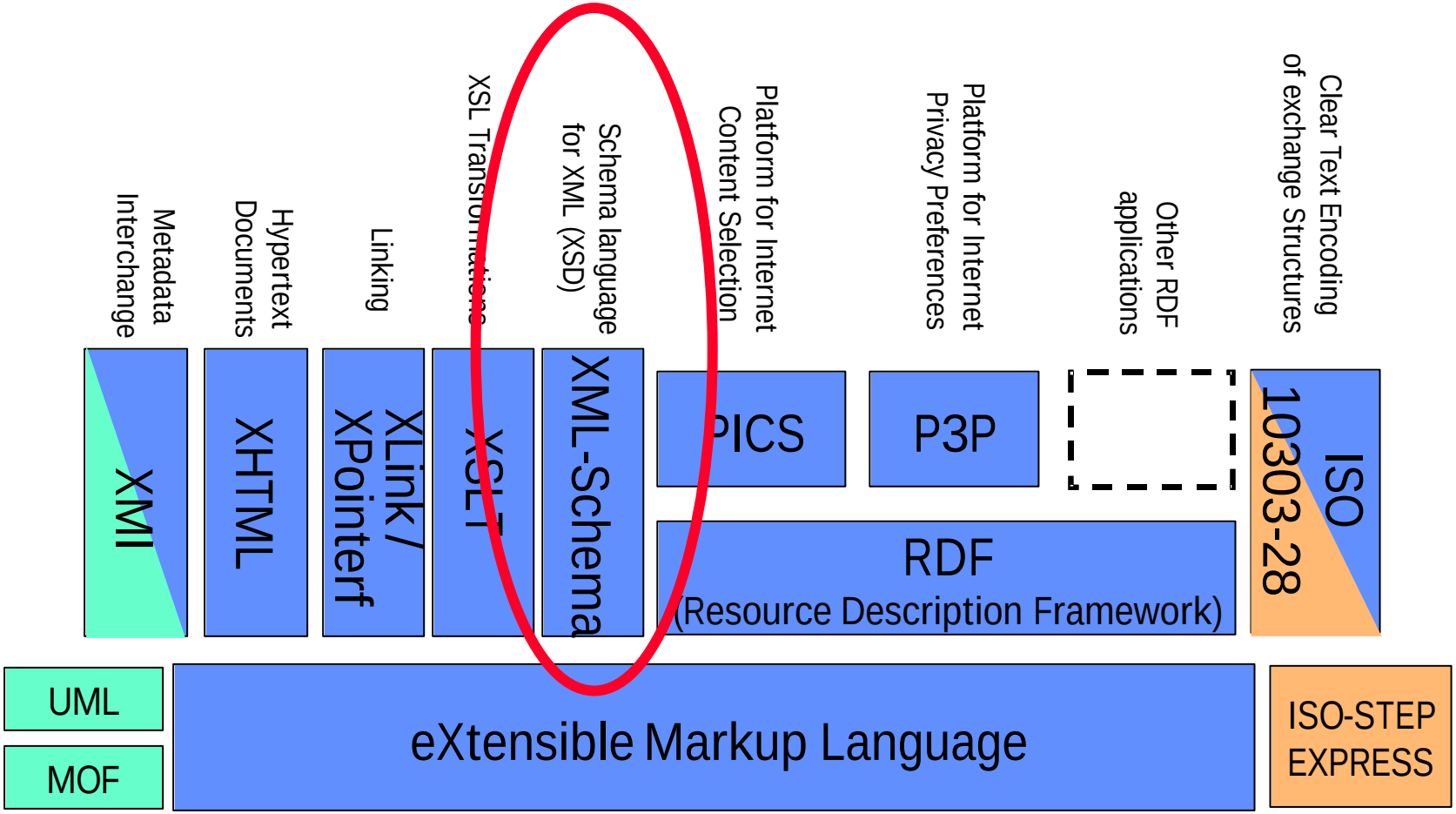
Konfluente Schemadialekte



W3C's XML-Schema konsolidiert und integriert die wesentlichen existierenden Ansätze und entwickelt sie fort.

Entwurf von XML Sprachen – XML Schema

W3C's XML Schema eingeordnet in die Welt der XML-Sprachen



Entwurf von XML Sprachen – XML Schema

W3C's XML Schema -- Eigenschaften

W3C‘s XML beschreiben XML-Dokumentklassen

unter Verwendung der XML-Dokumentkonstrukte um

- Bedeutung
- Aussage
- Beziehung

der Dokumentbestandteile einzuschränken und zu dokumentieren.

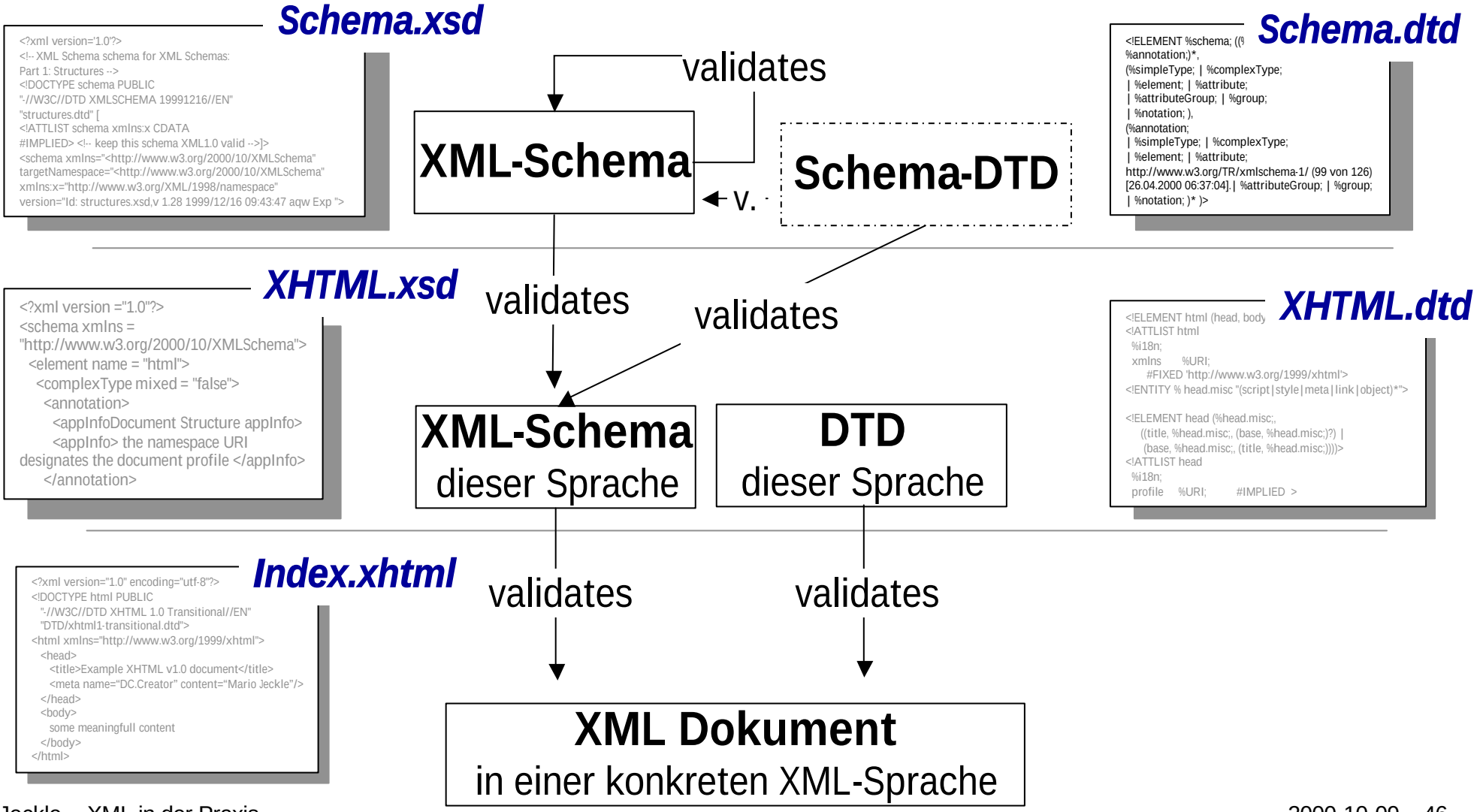
Bestandteile eines XML-Dokuments:

- Datentypen
- Elemente und ihr Inhalt
- Attribute und ihre Werte
- Entities und ihr Inhalt
- Notations

XML Schema sind selbstbeschreibend

Entwurf von XML Sprachen – XML Schema

Technologie Metamodellierung



Entwurf von XML Sprachen – XML Schema

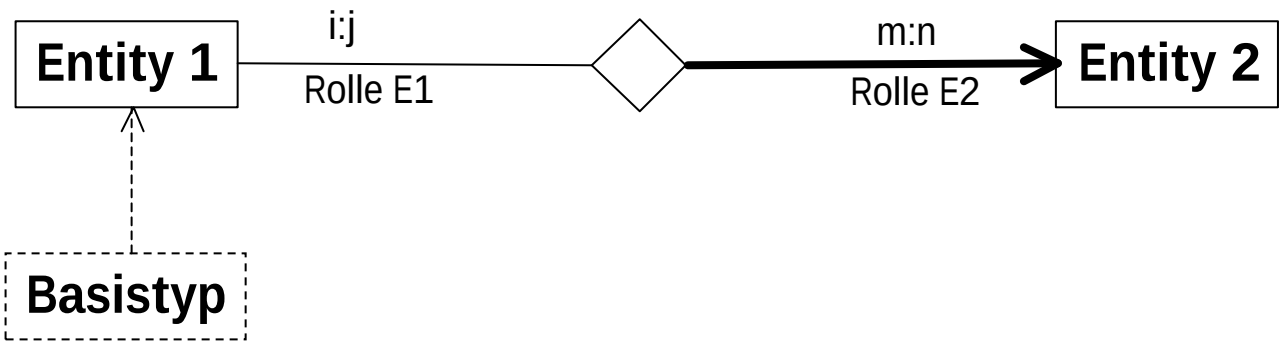
- XML-Schemata sind *keine Datenmodelle!*

  ...jedoch eignen sich Formalismen aus der Datenmodellierung zur anschaulichen intuitiven Beschreibung der Konstrukte.
- XML-Schemata beschreiben die Syntax einer konkreten XML-Sprache, nicht deren Semantik!
- XML-Schemata ermöglichen (syntaktische) Interoperabilität
- Generelle Regel: *Wiederverwendung exsistierender Schemata vor Entwicklung eigener.*

  => Recherchierender Zugriff auf verfügbare Schemarepositories

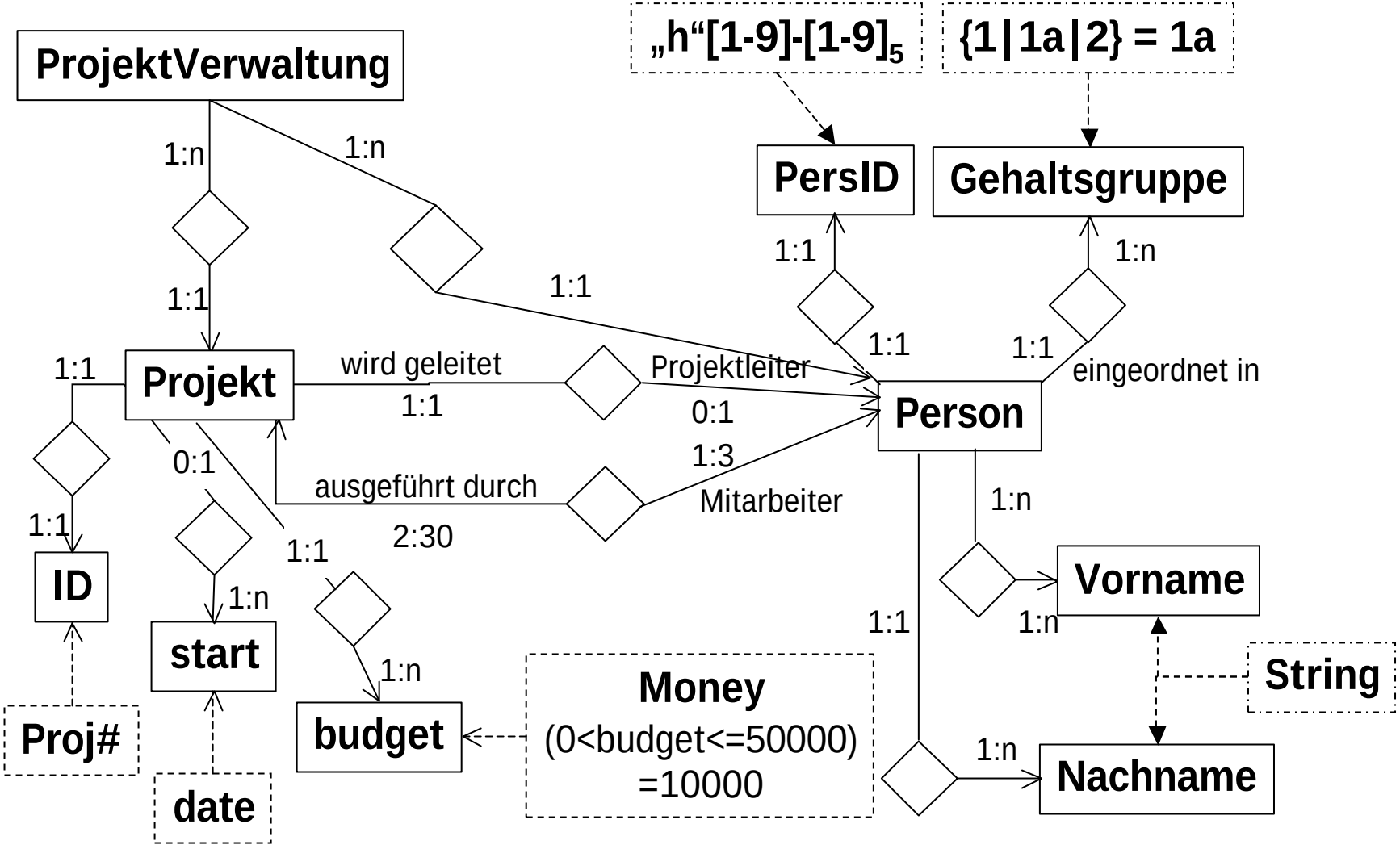
Entwurf von XML Sprachen – XML Schema

Graphische Beschreibungssprache des Beispiels



Entwurf von XML Sprachen – XML Schema

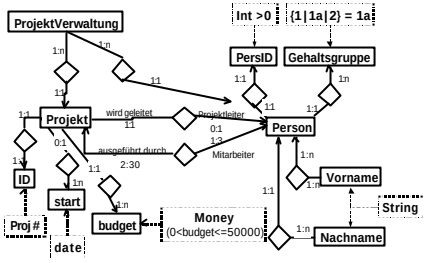
Beispiel

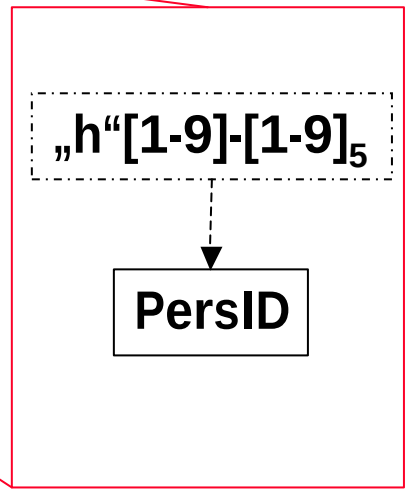


Entwurf von XML Sprachen – XML Schema

Beispiel ausgedrückt als DTD

```
<!ELEMENT ProjektVerwaltung  (Person+ , Projekt+ )>

<!ELEMENT Person (Vorname+, Nachname+)>

<!ATTLIST Person
  PersID      ID   #REQUIRED
  Gehaltsgruppe    (1|1a|2) '1a'
  mitarbeitInProjekt   IDREFS   #REQUIRED>

<!ELEMENT Vorname (#PCDATA)>
<!ELEMENT Nachname (#PCDATA)>

<!ELEMENT Projekt EMPTY>

<!ATTLIST Projekt
  ID       ID    #REQUIRED
  date      CDATA   #IMPLIED
  budget      CDATA   #REQUIRED
  Projektleiter   IDREF   #REQUIRED
  Mitarbeiter   IDREFS   #REQUIRED>
```



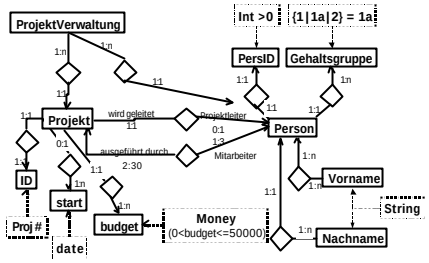


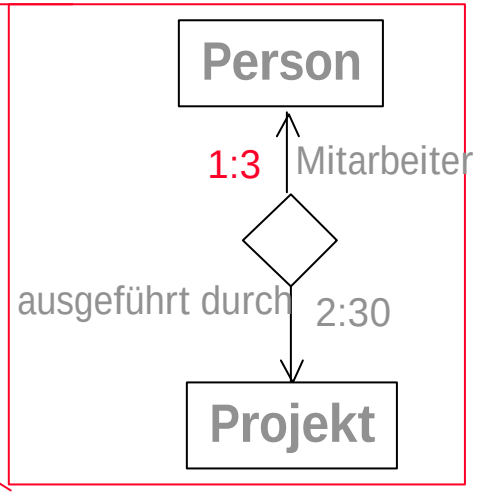
(lexikalische Festlegung)
Mit DTD nicht darstellbar!

Entwurf von XML Sprachen – XML Schema

Beispiel ausgedrückt als DTD

```
<!ELEMENT ProjektVerwaltung  (Person+ , Projekt+ )>

<!ELEMENT Person (Vorname+, Nachname+)>

<!ATTLIST Person
  PersID      ID   #REQUIRED
  Gehaltsgruppe    (1|1a|2) '1a'
  mitarbeitInProjekt   IDREFS   #REQUIRED>

<!ELEMENT Vorname (#PCDATA)>
<!ELEMENT Nachname (#PCDATA)>

<!ELEMENT Projekt EMPTY>

<!ATTLIST Projekt
  ID       ID    #REQUIRED
  date      CDATA   #IMPLIED
  budget      CDATA   #REQUIRED
  Projektleiter   IDREF   #REQUIRED
  Mitarbeiter   IDREFS   #REQUIRED>
```



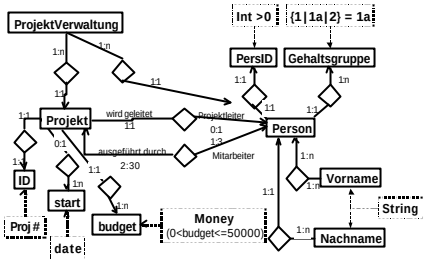


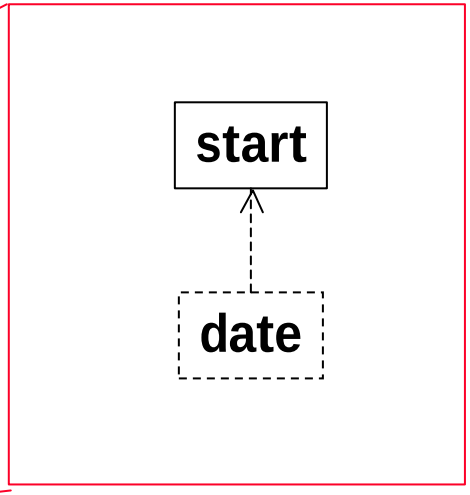
(Kardinalität)
Mit DTD nicht darstellbar!

Entwurf von XML Sprachen – XML Schema

Beispiel ausgedrückt als DTD

```
<!ELEMENT ProjektVerwaltung  (Person+ , Projekt+ )>

<!ELEMENT Person (Vorname+, Nachname+)>

<!ATTLIST Person
  PersID      ID   #REQUIRED
  Gehaltsgruppe    (1|1a|2) '1a'
  mitarbeitInProjekt   IDREFS   #REQUIRED>

<!ELEMENT Vorname (#PCDATA)>
<!ELEMENT Nachname (#PCDATA)>

<!ELEMENT Projekt EMPTY>

<!ATTLIST Projekt
  ID       ID    #REQUIRED
  start      CDATA   #IMPLIED
  budget      CDATA   #REQUIRED
  Projektleiter   IDREF   #REQUIRED
  Mitarbeiter   IDREFS   #REQUIRED>
```



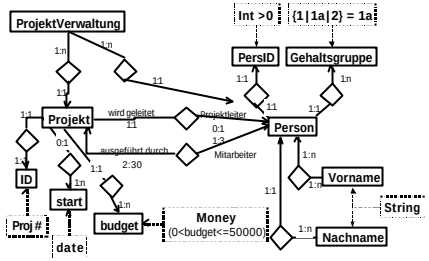


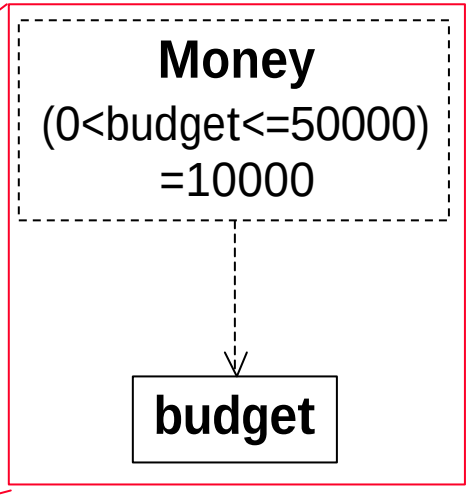
(Datentyp)
Mit DTD nicht darstellbar!

Entwurf von XML Sprachen – XML Schema

Beispiel ausgedrückt als DTD

```
<!ELEMENT ProjektVerwaltung  (Person+ , Projekt+ )>

<!ELEMENT Person (Vorname+, Nachname+)>

<!ATTLIST Person
  PersID      ID   #REQUIRED
  Gehaltsgruppe    (1|1a|2) '1a'
  mitarbeitInProjekt   IDREFS   #REQUIRED>

<!ELEMENT Vorname (#PCDATA)>
<!ELEMENT Nachname (#PCDATA)>

<!ELEMENT Projekt EMPTY>

<!ATTLIST Projekt
  ID       ID    #REQUIRED
  start      CDATA   #IMPLIED
  budget      CDATA   #REQUIRED
  Projektleiter   IDREF   #REQUIRED
  Mitarbeiter   IDREFS   #REQUIRED>
```





(Domänenrestriktion und **Vorgabewert)**
Mit DTD nicht darstellbar!

Entwurf von XML Sprachen – XML Schema

Anatomie eines XSD-Dokuments

```
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
  <xsd:annotation>
    <xsd:documentation>
    Uninterpreted Commend
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType ... >
    <xsd:element ...>
    <xsd:attribute ...>
  </xsd:complexType>
  <xsd:simpleType ...>
    <xsd:pattern ...>
  </xsd:simpleType>
</xsd:schema>
```

Schema-Start mit Namespacedeklaration

Entwurf von XML Sprachen – XML Schema

Anatomie eines XSD-Dokuments

```
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
  <xsd:annotation>
    <xsd:documentation>
    Uninterpreted Commend
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType ... >
    <xsd:element ...>
    <xsd:attribute ...>
  </xsd:complexType>
  <xsd:simpleType ...>
    <xsd:pattern ...>
  </xsd:simpleType>
</xsd:schema>
```

Dokumentierender Kommentar

Entwurf von XML Sprachen – XML Schema

Anatomie eines XSD-Dokuments

```
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
  <xsd:annotation>
     <xsd:documentation>
     Uninterpreted Commend
     </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType ... >
     <xsd:element ...>
     <xsd:attribute ...>
  </xsd:complexType>
  <xsd:simpleType ...>
     <xsd:pattern ...>
  </xsd:simpleType>
</xsd:schema>
```

Anwenderdefinierte (komplexe) Typen

Entwurf von XML Sprachen – XML Schema

Anatomie eines XSD-Dokuments

```
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
  <xsd:annotation>
     <xsd:documentation>
     Uninterpreted Commend
     </xsd:documentation>
   </xsd:annotation>
   <xsd:complexType ... >
     <xsd:element ...>
     <xsd:attribute ...>
   </xsd:complexType>
   <xsd:simpleType ...>
     <xsd:pattern ...>
   </xsd:simpleType>
</xsd:schema>
```

Elemente und Attribute

Entwurf von XML Sprachen – XML Schema

Anatomie eines XSD-Dokuments

```
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
  <xsd:annotation>
     <xsd:documentation>
     Uninterpreted Commend
     </xsd:documentation>
   </xsd:annotation>
   <xsd:complexType ... >
     <xsd:element ...>
     <xsd:attribute ...>
   </xsd:complexType>
   <xsd:simpleType ...>
     <xsd:pattern ...>
   </xsd:simpleType>
</xsd:schema>
```

Anwenderdefinierte (einfache) Datentypen

Entwurf von XML Sprachen – XML Schema

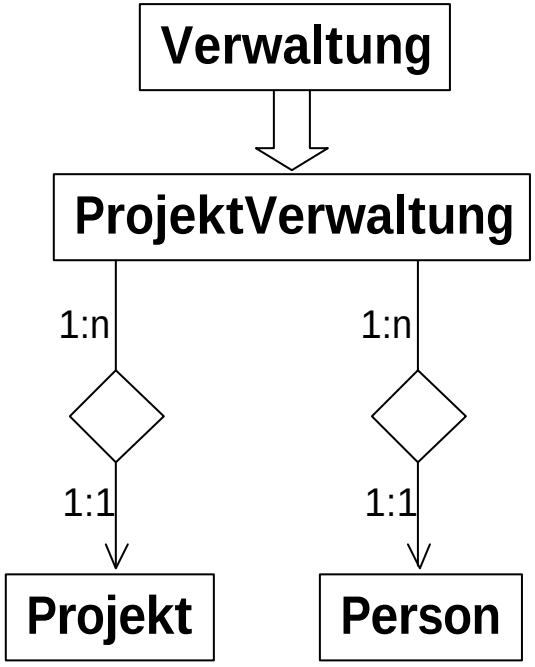
Typdefinition – *complex type*

```
<complexType name="complexType" abstract="true">
  <complexContent>
    <extension base="annotated">
      <group ref="complexTypeModel"/>
      <attribute name="name" type="NCName">
        <annotation>
          <documentation>Will be restricted to required or forbidden</documentation>
        </annotation>
      </attribute>
      <attribute name="mixed" type="boolean" use="default" value="false">
        <annotation>
          <documentation>Not allowed if simpleContent child is chosen.
          May be overriden by setting on complexContent child.</documentation>
        </annotation>
      </attribute>
      <attribute name="abstract" type="boolean" use="default" value="false"/>
      <attribute name="final" type="derivationSet"/>
      <attribute name="block" type="derivationSet" use="default" value=""/>
    </extension>
  </complexContent>
</complexType>
```

Entwurf von XML Sprachen – XML Schema

Typdefinition – *complex type*

- Abstrakte Typdefinition möglich; Typ darf nicht in XML-Dokumentinstanzen auftreten (*abstract*)
- Restriktion verschiedener Substitutionsmimiken (*block*)
- Vererbungs-Restriktion (*final*)
- Vererbung durch Typerweiterung oder -Einschränkung (*base-Attribut einer extension*)




```
<xsd:complexType name="ProjektVerwaltungType" mixed="no"
  abstract="false" block="restriction" final="restriction">
   <xsd:extension base="Verwaltung"/>
   <xsd:element name="Person" minOccurs="1" maxOccurs="unbound"/>
   <xsd:element name="Projekt" minOccurs="1" maxOccurs="unbound"/>
</xsd:complexType>
```

Entwurf von XML Sprachen – XML Schema

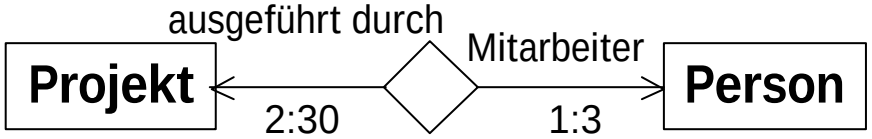
Typdefinition – *element*

```
<complexType name="element" abstract="true">
  <annotation>
    <documentation>The element element can be used either
    at the toplevel to define an element-type binding globally,
    or within a content model to either reference a globally-defined
    element or type or declare an element-type binding locally.
    The ref form is not allowed at the top level.</documentation>
  </annotation>
  <complexContent>
    <extension base="annotated">
      <sequence>
        <choice minOccurs="0">
            <element name="simpleType" type="localSimpleType"/>
            <element name="complexType" type="localComplexType"/>
        </choice>
        <element ref="identityConstraint"
       minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
      <attributeGroup ref="defRef"/>
      <attribute name="type" type="QName"/>
      <attribute name="substitutionGroup" type="QName"/>
      <attributeGroup ref="occurs"/>
      <attribute name="default" type="string"/>
      <attribute name="fixed" type="string"/>
      <attribute name="nullable" type="boolean" use="default"
      value="false"/>
      <attribute name="abstract" type="boolean" use="default"
      value="false"/>
      <attribute name="final" type="derivationSet"
      use="default" value=""/>
      <attribute name="block" type="blockSet"
      use="default" value=""/>
      <attribute name="form" type="formChoice"/>
    </extension>
  </complexContent>
</complexType>
```

Entwurf von XML Sprachen – XML Schema

Typdefinition – *element*

- (ad-hoc Polymorphie, *Overloading*) Gleich benannte Elemente mit verschiedenen Inhaltsmodellen sind zulässig.



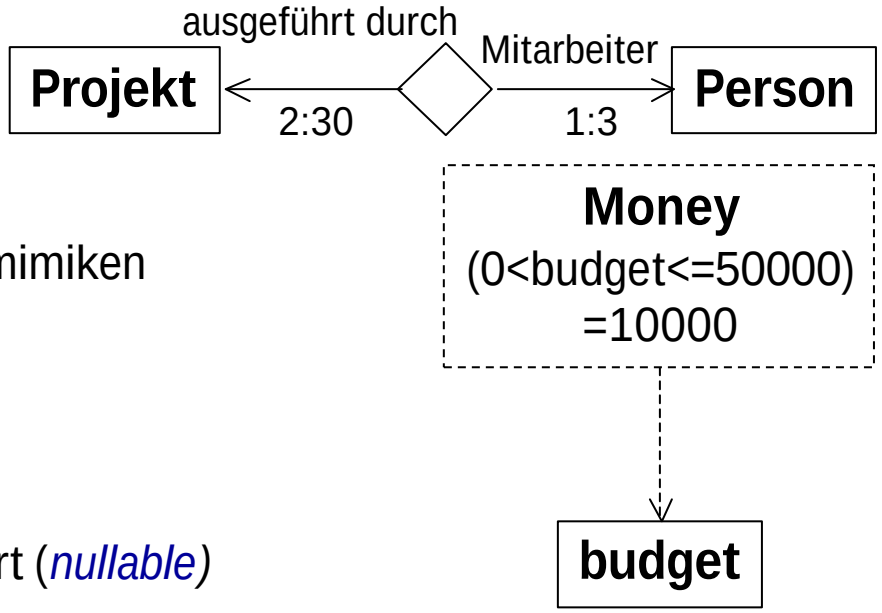
- Elementäquivalenz zu bestehenden Element (*substitutionGroup*)
- Explizite Modalität und Kardinalität (*minOccurs*, *maxOccurs*)

  (Vorgabebelegung 1)

```
<xsd:element name="Projekt" type="ProjektType"/>
<xsd:element name="Projekt" type="xsd:string"/>
<xsd:complexType name="Projekt">
  <xsd:element ref="Mitarbeiter" minOccurs="2" maxOccurs="30"/>
</xsd:complexType>
<xsd:element name="myComment" substitutionGroup="comment"/>
```

Entwurf von XML Sprachen – XML Schema

Typdefinition – *element*



- Abstrakte Elemente (*abstract*)
- Vorgabewerte (*default*)
- Restriktion verschiedener Substitutionsmimiken (*block*)
- Konstante Belegung (*fixed*)
- Vererbungs-Restriktion (*final*)
- Lukasiewicz- / Tri-State-Logik / NULL-Wert (*nullable*)
- Referenzierung auf bestehende Elemente (*ref*)

```
<xsd:element name="Mensch" abstract="true" block="restriction" />
<xsd:element name="budget" default="10000"/>
<xsd:element ref="Person"/>
```

Entwurf von XML Sprachen – XML Schema

Nullwerte

XML-Schema unterstützt kein explizites Symbol für fehlende Werte (*NULL).*
Um die Unterstützung fehlender Werte zuzulassen muß das entsprechende Element im Schema mit *nullable="true"* attributiert werden.

```
<xsd:element name="nEl" nullable="true"/>
```

Im XML-Dokument wird das in XML-Schema vordefinierte Attribut *null* auf *true* gesetzt.
Der explizite *NULL*-Wert ist Teil des XML-Schema-Namensraums für XML-Schema Instanzen (*instance documents)* (http://www.w3.org/2000/10/XMLSchema-instance). Das Namespacepräfix *xsi* ist zwingend.

```
<nEl xsi:null="true"/>
```

Entwurf von XML Sprachen – XML Schema

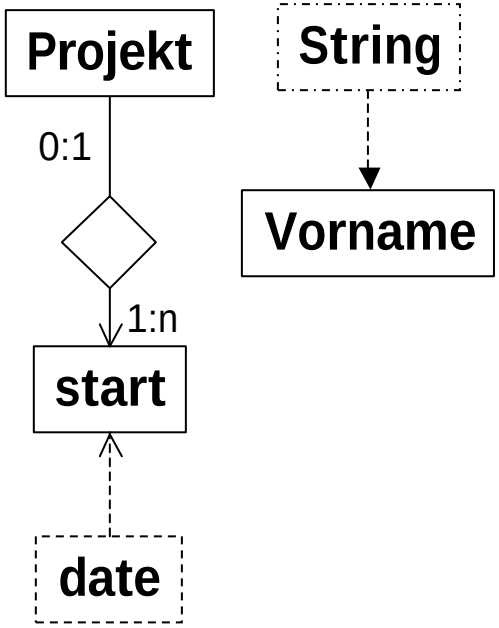
Typdefinition -- *attribute*

```
<complexType name="attribute">
  <complexContent>
    <extension base="annotated">
    <sequence>
      <element name="simpleType" minOccurs="0" type="localSimpleType"/>
    </sequence>
      <attributeGroup ref="defRef"/>
     <attribute name="type" type="QName"/>
     <attribute name="use" use="default" value="optional">
     <simpleType>
        <restriction base="NMTOKEN">
          <enumeration value="prohibited"/>
          <enumeration value="optional"/>
          <enumeration value="required"/>
          <enumeration value="default"/>
          <enumeration value="fixed"/>
         </restriction>
        </simpleType>
      </attribute>
      <attribute name="value" use="optional" type="string"/>
      <attribute name="form" type="formChoice"/>
    </extension>
  </complexContent>
</complexType>
```

Entwurf von XML Sprachen – XML Schema

Typdefinition -- *attribute*

- Typisierung gemäß *XML Schema Part 2* (*type*)
- Vorgabewerte (use=*default*)
- Konstante Belegung (use=*fixed*)
- Optionale Vergabe (use=*optional*)
- Zwingende Vergabe / *mandatory* (use=*required*)
- Untersagung der Verwendung (use=*prohibited*)
- Referenzierung auf bestehendes Attribut (*ref*)
- Mengenwertige Attribute (*minOccurs*, *maxOccurs*)



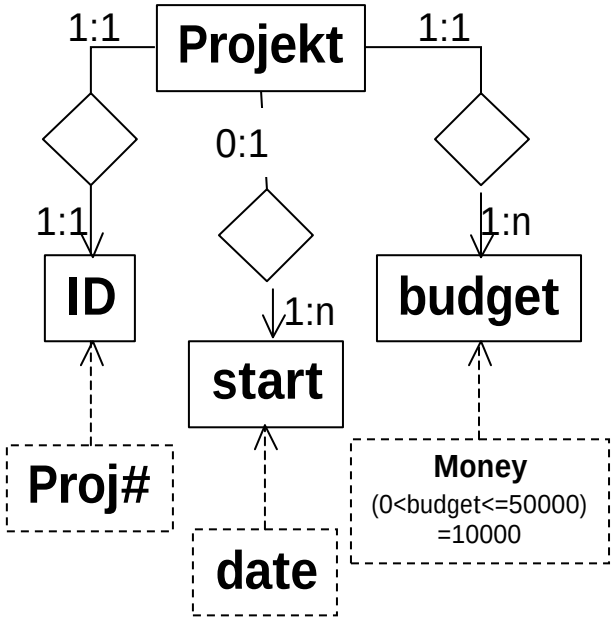

```
<xsd:attribute name="Vorname" type="dt:string" />
<xsd:attribute name="start" type="dt:date" use="optional" />
```

Entwurf von XML Sprachen – XML Schema

Typdefinition – *Attributgruppen*

- Zusammenfassung von Attributen zu einer benannten Gruppe
- Verbessert Les- und Wartbarkeit des Schemas durch Zentralisierung der Information (ähnlich den *parameter entities* in XML 1.0)

```
<xsd:complexType name="Projekt">
  ...
  <attributeGroup ref="Project'sAttributes"/>
</xsd:complexType>
<xsd:attributeGroup name="Project'sAttributes">
  <xsd:attribute name="ID" type="ProjectNo"/>
  <xsd:attribute name="start" type="xsd:date"/>
  <xsd:attribute ref="BudgetAtt"/>
</xsd:attributeGroup>
```



Entwurf von XML Sprachen – XML Schema

Typdefinition – A*nnotationen*

- Explizite Dokumentationsmöglichkeit
- Informale Beschreibung für (menschliche) Schemanutzer (*documentation*)
- Formale Beschreibung für schemaverarbeitende Applikationen (*appInfo*)
- Werden im Validierungsprozeß nicht berücksichtigt

```
<xsd:annotation>
   <xsd:appinfo source="http://www...."/>
    <xsd:documentation xml:lang="en-en">
       Schema created 2000-10-09 by ...
     </xsd:documentation
</xsd:annotation>
<xsd:annotation>
  <xsd:appinfo>
     <dcx:creator>...</dcx:creator><dcx:version>1.0</dcx.version>
</xsd:annotation>
```

Entwurf von XML Sprachen – XML Schema

Schema Part 2 -- Datentypen

- Zunehmend *datenorientierte* Betrachtung erfordert ein leistungsfähiges Typsystem zur Darstellung von "alltäglichen" Standardsituationen
- Erhöhung des Qualitätsgrades der entstehenden XML-Dokumente durch semantische Anreicherung
- Erweiterung generischer Parser um zusätzliche inhaltliche Prüfung
- Verminderung des *impedance mismatch* zwischen Anwendungssystem (Programmiersprache, Datenbanksystem, ...) und Streaming-Format
- Typsystem auf Basis bekannter Standards wie ISO 11404, SQL, Java
- Charakterisierung der Typen in
    - atomare vs. Listentypen
      (atomic vs. list types)
    - primitive vs. abgeleitete Typen
      (primitive vs. derived types)
    - Standardtypen vs. Andwenderdefinierte
      (built-in vs. user-derived datatypes)

Entwurf von XML Sprachen – XML Schema

Schema Part 2 -- Datentypen

Datentypcharakterisierung

Atomare Datentypen: Unteilbare Werte *(z.B. string)*

Listentypen: Endliche Menge atomarer Typausprägungen *(z.B. list of integer)*

Vereinigungstypen: Union verbindet atomare und listenartige Typen

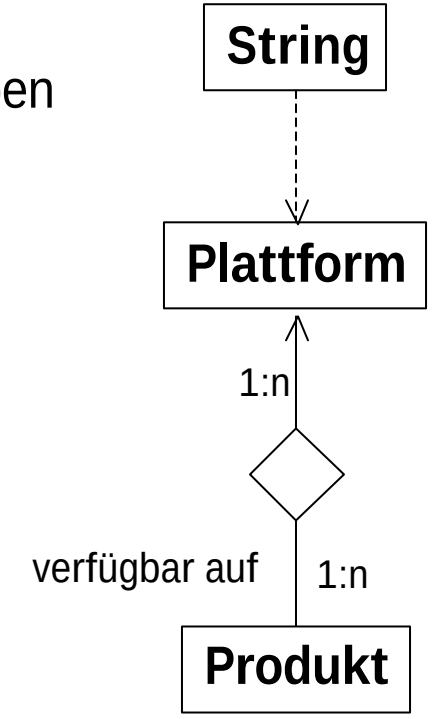
Definition:

```
<simpleType name="Plattformen">
  <list itemType="string"/>
</simpleType>
```

Verwendung (in XML-Dokumentinstanz):

```
<verfügbarePlattformen xsi:type="Plattformen">
  Windows   HP-UX   AIX   Linux
</verfügbarePlattformen>
```

XML *whitespaces* (0x9, 0x20, 0xA, 0xD) gelten als Element-separatoren (=> Vorsicht bei Strings mit Leerzeichen)



Entwurf von XML Sprachen – XML Schema

Schema Part 2 -- Datentypen

Datentypcharakterisierung

- **Standardtypen:** in der XML Schema 2-Spezifikation definiert
- **Anwenderdefinierte Typen:** abgeleitete Einschränkung bestehender Typen
- **build-in Datentypen:** unabhängig von anderen Typen definiert
- **abgeleitete Datentypen:** (abhängige) Definition unter Verwendung bereits existierender Datentypen

Realisierung:

- Bezug auf Basistypen durch das *base*-Attribut
- Basistypen können selbst wieder primitiv oder abgeleitet sein (-> Typhierarchie)

Entwurf von XML Sprachen – XML Schema

Schema Part 2 -- Datentypen

Ausschnitt aus *schema for schemas* (WD 2000-09-22)*:*

First the buildin primitive datatypes. These definitions are for information only, the real buildin definitions are magic. Note in particular there is no type named 'anySimpleType' .
The primitives should really be derived from no type at all, and anySimpleType should be derived as a union of all the primives.

```
<xsd:simpleType name="decimal">
  <restriction base="anySimpleType" />
</xsd:simpleType>
<xsd:simpleType name="integer">
  <restriction base="decimal" />
</xsd:simpleType>
<xsd:simpleType name="nonPositiveInteger">
  <restriction base="integer" />
</xsd:simpleType>
```

Entwurf von XML Sprachen – XML Schema

Schema Part 2 – Datentypen – *build in datatypes*

- ID
- IDREF
- IDREFS
- ENTITY
- ENTITIES
- NMTOKEN
- NMTOKENS
- Name
- QName
- NCName
- NOTATION

- integer
- nonPositiveInteger
- negativeInteger
- long
- int
- short
- byte
- nonNegativeInteger
- unsignedLong
- unsingnedInt
- unsignedShort
- unsignedByte
- positiveInteger
- boolean
- float
- double
- decimal

- string
- date
- time
- timeInstant
- timePeriod
- month
- year
- century
- recurringDate
- recurringDay
- timeDuration
- recurringDuration
- binary
- uriReference
- language

Entwurf von XML Sprachen – XML Schema

Schema Part 2 – Datentypen – *build in datatypes*

- integer — -1, 0, 7683, +555
- nonPositiveInteger — {..., -2, -1, 0}
- negativeInteger — {..., -3, -2, -1}
- long — -2^{63} <= long <= $2^{63}-1$
- int — -2^{31}<= int <= $2^{31}-1$
- short — -2^{15} <= short <= short $2^{15}-1$
- byte — -2^{7} <= byte <= $2^{7}-1$
- nonNegativeInteger — {0,1,2,...}
- positiveInteger — {1,2,3,...}
- unsignedLong — 0 <= unsignedLong <= $2^{64}-1$
- unsingnedInt — 0 <= unsignedInt <= $2^{32}-1$
- unsignedShort — 0 <= unsignedShort <= $2^{16}-1$
- unsignedByte — 0 <= unsignedByte <= $2^{8}-1$
- boolean — {true, 1, false, 0}
- float — 32-Bit Fließkommazahl gemäß IEEE 754-1985 — -1E4, 12.64E8, 12e-2, INF
- double — 64-Bit Fließkommazahl gemäß IEEE 754-1985
- decimal — -12, 8, 3.14151592, +1.0

Entwurf von XML Sprachen – XML Schema

Schema Part 2 – Datentypen – *build in datatypes*

• string	ISO 10646 und Unicode	"hello world"
• date	ISO 8601	2000-10-00
• time	ISO 8601	09:00:00+2:00
• timeInstant	ISO 8601	2000-10-09T09:00:00+2:00
• timePeriod	ISO 8601	P7M
• month	ISO 8601	P1M
• year	ISO 8601	P1Y
• century	ISO 8601	19
• recurringDate	ISO 8601	PT24H
• recurringDay	ISO 8601	P1M2D
• timeDuration	ISO 8601	P0Y0M0DT1H30M
• recurringDuration	*abstrakter Supertyp von* duration *und* period*; nicht direkt verwendbar!*	
• binary	*abstrakter Typ; nur verwandbar durch Ableitung und* encoding-*Spezifikation*	
• uriReference	IETF RFC2396	http://www.jeckle.de
• language	IETF RFC1766	de-de, en-uk, x-klingon

Entwurf von XML Sprachen – XML Schema

Schema Part 2 – Datentypen

Typdefinition anwendbar auf...

...Elemente

```
<xsd:element name="elementName" type="TypeName" />
```

- Elementtyp kann als *complexType* gemäß Schema 1 strukturiert sein, oder gemäß Schema 2 *build-in* bzw. selbstdefiniert sein

...Attribute

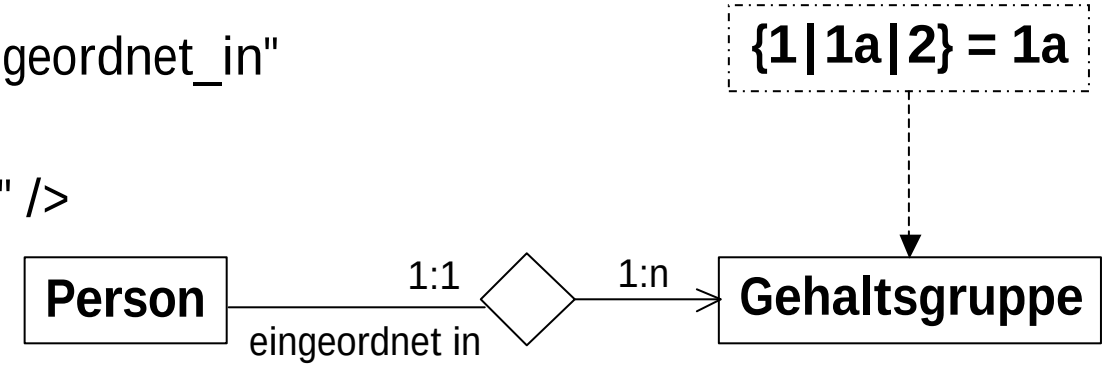
```
<xsd:attribute name="AttributeName" type="TypeName" />
```

- Attributtyp kann gemäß Schema 2 *build-in* bzw. selbstdefiniert sein
  => Wie in XML v1.0 üblich, keine (explizite) Strukturierung durch Markup innerhalb Attributen zugelassen

Entwurf von XML Sprachen – XML Schema

Schema Part 2 – Datentypen – Typdefinition *enumeration*

```
...
  <xsd:attribute name="eingeordnet_in"
    type="Gehaltsgruppe"
    value="1a" use="default" />
...
```

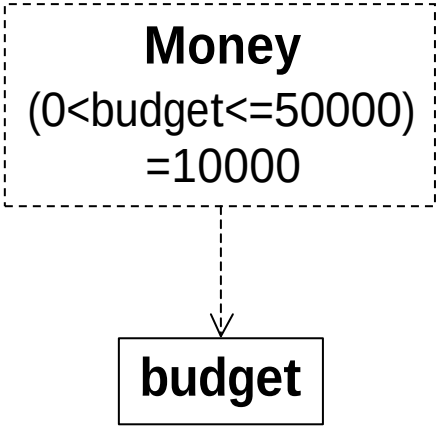



```
<xsd:simpleType name="Gehaltsgruppe" base="xsd:string">
  <xsd:annotation>
      <xsd:documentation>XY AG Gehaltsgruppen per 1999-12-31</xsd:documentation>
  </xsd:annotation>
  <xsd:enumeration value="1" />
  <xsd:enumeration value="1a" />
  <xsd:enumeration value="2" />
</xsd:simpleType>
```

Entwurf von XML Sprachen – XML Schema

Schema Part 2 – Datentypen – Typdefinition durch Ableitung

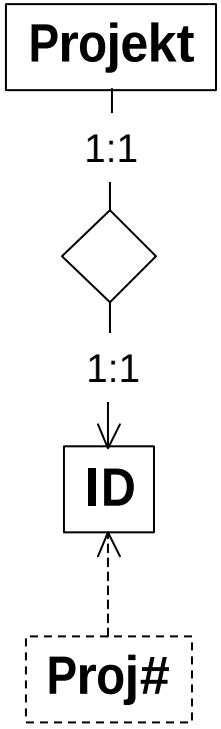
- Erweitung des bestehenden Typsystems durch anwenderdefinierte Typen.




```
<xsd:attribute name="budgetAtt" use="default" value="10000">
  <xsd:simpleType base="dt:float">
    <xsd:minExclusive value="0"/>
    <xsd:maxInclusive value="50000"/>
    <xsd:scale value="2"/>
  </xsd:simpleType>
</xsd:attribute>
```

Entwurf von XML Sprachen – XML Schema

Schema Part 2 – Datentypen – lexikalische Typdefinition



- Spezifikation des Wertebereichs eines Datentyps durch reguläre Ausdrücke

```
<xsd:simpleType name="ProjektNo">
  <pattern value="P-\(19|20)\Nd{2,}-\Lu+\Nd{3,5}"/>
</xsd:simpleType>
```

Business Regel:
Projektnummern beginnen stets mit **P**, darauf folgt das Jahr, abgetrennt durch einen Bindestrich eine mindestens aus einem Großbuchstaben (*Lu*) bestehende Identifikation, an die sich mindestens drei, aber höchstens fünf numerische Ziffern anschließen.

Entwurf von XML Sprachen – XML Schema

Schema validness

Aufbauend auf der generellen Konformität (*valdiness)* zu einer existierenden DTD wird die *schema validness* definiert:

- XML-Instanz referenziert "ihr" Schema (namespace-Deklaration im Root-Element)
- *XML-Schema-instance* Namespace (falls verwendet)
- Gültigkeitseinschränkungen an Schema Elemente
  (z.B. minOccurs <= maxOccurs)
- Konform zu *Schema for Schemas*

Entwurf von XML Sprachen – XML Schema

Schema validness

Index.xhtml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC
  "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Example XHTML v1.0 document</title>
    <meta name="DC.Creator" content="Mario Jeckle"/>
  </head>
  <body>
    some meaningfull content
  </body>
</html>
```

XHTML.xsd

```
<?xml version ="1.0"?>
<schema xmlns =
"http://www.w3.org/2000/10/XMLSchema">
  <element name = "html">
    <complexType mixed = "false">
      <annotation>
        <appInfoDocument Structure appInfo>
        <appInfo> the namespace URI
designates the document profile </appInfo>
      </annotation>
```

Schema.xsd

```
<?xml version='1.0'?>
<!-- XML Schema schema for XML Schemas:
Part 1: Structures -->
<!DOCTYPE schema PUBLIC
"-//W3C//DTD XMLSCHEMA 19991216//EN"
"structures.dtd" [
<!ATTLIST schema xmlns:x CDATA
#IMPLIED> <!-- keep this schema XML1.0 valid -->]>
<schema xmlns="<http://www.w3.org/2000/10/XMLSchema"
targetNamespace="<http://www.w3.org/2000/10/XMLSchema"
xmlns:x="http://www.w3.org/XML/1998/namespace"
version="Id: structures.xsd,v 1.28 1999/12/16 09:43:47 aqw Exp ">
```

XHTML Dokument → **XHTML Schema** → **Schema Schema**

XML Dokument konform zu Schemabeschreibung?

XML Dokument (Schema) konform zu (Schema) Schemabeschreibung?

Entwurf von XML Sprachen – XML Schema

Schema Zusammenfassung

- *XML Schema* bilden hinsichtlich Mächtigkeit eine Obermenge der XML DTD-Ausdrucksfähigkeit
- *XML Schema* stellen einen fundierten und zukunftsfähigen Ansatz zur Beschreibung von XML-Sprachen in Bezug auf Struktur und Datentypen dar
- Mittelfristig werden sie parallel zum XML v1.0 DTD-Mechanismus existieren und ergänzend eingesetzt werden
- Langfristig ist mit der breiten Marktakzeptanz, und als Konsequenz der Ablösung des bestehenden DTD-Mechanismus, der Schema-Sprachen zu rechnen
- *XML Schema* ist eine Sprache zur Beschreibung hierarchischer XML-Daten- und -Dokumentstrukturen, jedoch zur Datenmodellierung nur bedingt geeignet; *XML Schema* ist keine (*general purpose)* Datenmodellierungssprache
- Erste Toolimplementierungen liegen vor
- Mit der Verabschiedung (W3C Recommendation) ist im laufenden Jahr zu rechnen

Entwurf von XML Sprachen – XML Schema

Einordnung in die XML-Sprachfamilie

- *XML Schema* ist als XML-Sprache eine Anwendung der XML
- Jedes Schema ist ein XML-Dokument
- *Schema for Schema* ist selbst ein *schema-valid* XML-Dokument
- Nicht-hierarchische Strukturen sinnvollst mit XLL (XPointer, XLink, XPath) ausdrückbar
- Schemata für Modellierungssprachen und Datenmodelle (z.B. Austauschformat für UML-Diagramme) sinnvollst mit OMG's *XML Metadata Interchange* (XMI) ausdrückbar

Entwurf von XML Sprachen – XML Schema

Empfehlungen zum praktischen Einsatz

- Schema-Recherche vor Eigenentwicklung!
- Information die in verschiedenen Rollen auftritt sollte (generell) als *complexType* definiert werden, um die Wiederverwendbarkeit zu erhöhen
- Wenn möglich...
  - spezialisierte Datentypen nutzen
  - eigene Datentypen ableiten
  - reguläre Ausdrücke
  - Aufzählungstypen
- Bidirektional navigierbare Beziehungen mit Kardinalitätsanteil größer Eins in beiden Richtungen (sog. *n:m-Beziehungen*) in separate Elemente aufbrechen

Entwurf von XML Sprachen mit XMI

Einführung

XML Metadata Interchange (XMI)

- Internationaler Industriestandard, verabschiedet durch die Object Management Group (OMG)
- Anwendungs-Schwerpunkte
  - Modell- und Metadatenaustausch

    (-> siehe folgendes Kapitel)
  - Erzeugung von XML DTDs und –Schemata
- Modellaustauschkomponente:
  Umgesetzt und unterstützt von marktführenden CASE-Tools
- DTD-Erzeugungskomponente:
  Spezialisierte Werkzeuge verfügbar (IBMs XMI Toolkit)

Entwurf von XML Sprachen mit XMI

XML-Strukturen ...

- Wiederspiegelung komplexer (Business) Strukturen
- Modellierungskultur
- Entstehung von *Dokumentfamilien* im Unternehmen
- Standardkompatibilität (semantische Interoperabilität)
- Hyperlinking über Dokumenttypgrenzen hinweg
- Applikationsseitige System- und Prozeßintegration (*impeadance mismatch*)
- Langzeitspeicherung
- Adaptierbarkeit, Erweiterbarkeit
- Kein existierender (weit anerkannter) Entwurfs-Formalismus

Entwurf von XML Sprachen mit XMI

Gegenwärtige Situation in der Praxis

- Verwendung von Schemaeditoren und weiteren Werkzeugen

  (-> nicht integriert, keine Standardnotation, kein Entwurfsprozeß)

- Zu hohes Abstraktionsniveau der Schemasprachen

  (-> mit unter Reverse-Engineering notwendig)

- Fehlende Dokumentationsintegration

  (-> intransparente Semantik => Integrations- und Kopplungsproblematik)

- Statische Dokumentschemata

  (-> geringer Wiederverwendungsgrad durch fehlende Modularisierung)

- Schema allein (ohne zusätzliche Information) nicht verwendbar

Entwurf von XML Sprachen mit XMI

XMI-Ansatz: Generierung von XML-Strukturen aus UML-Modellen

- Existierendes Klassendiagramm wird unverändert zur Erzeugung von XML-DTDs (zukünftig auch Schemata) herangezogen.
  - nahtlose Prozeßintegration
  - Flexible, zeitnahe Adaptierbarkeit
  - Nachvollziehbare XML-Strukturen gleichbleibender Qualität
  - Fördert entstehen von Sprachfamilien
- XML-DTD / Schema entspricht strukturell dem Klassendiagramm, nicht jedoch semantisch.

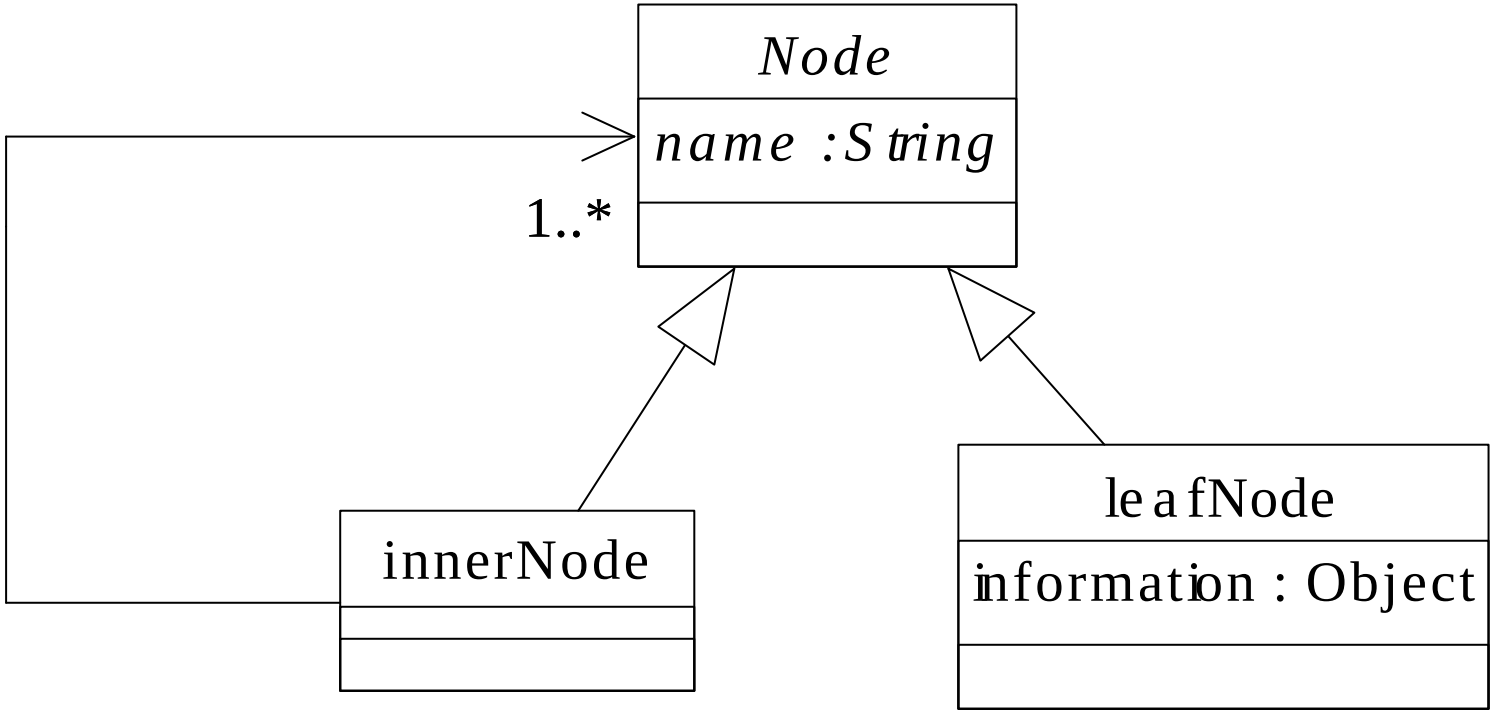
Entwurf von XML Sprachen mit XMI

Abbildung der OO-Konzepte in XML-Strukturen

Klasse	->	XML-Elemente, auch Ebenen
objektwertige Attribute	->	XML-Elemente, verschiedener Ebene
Kapselung	->	XML-Elemente, verschiedener Ebene
Referenzen	->	XML-Attribute (IDREF)
Attribute	->	XML-Attribute (CDATA)

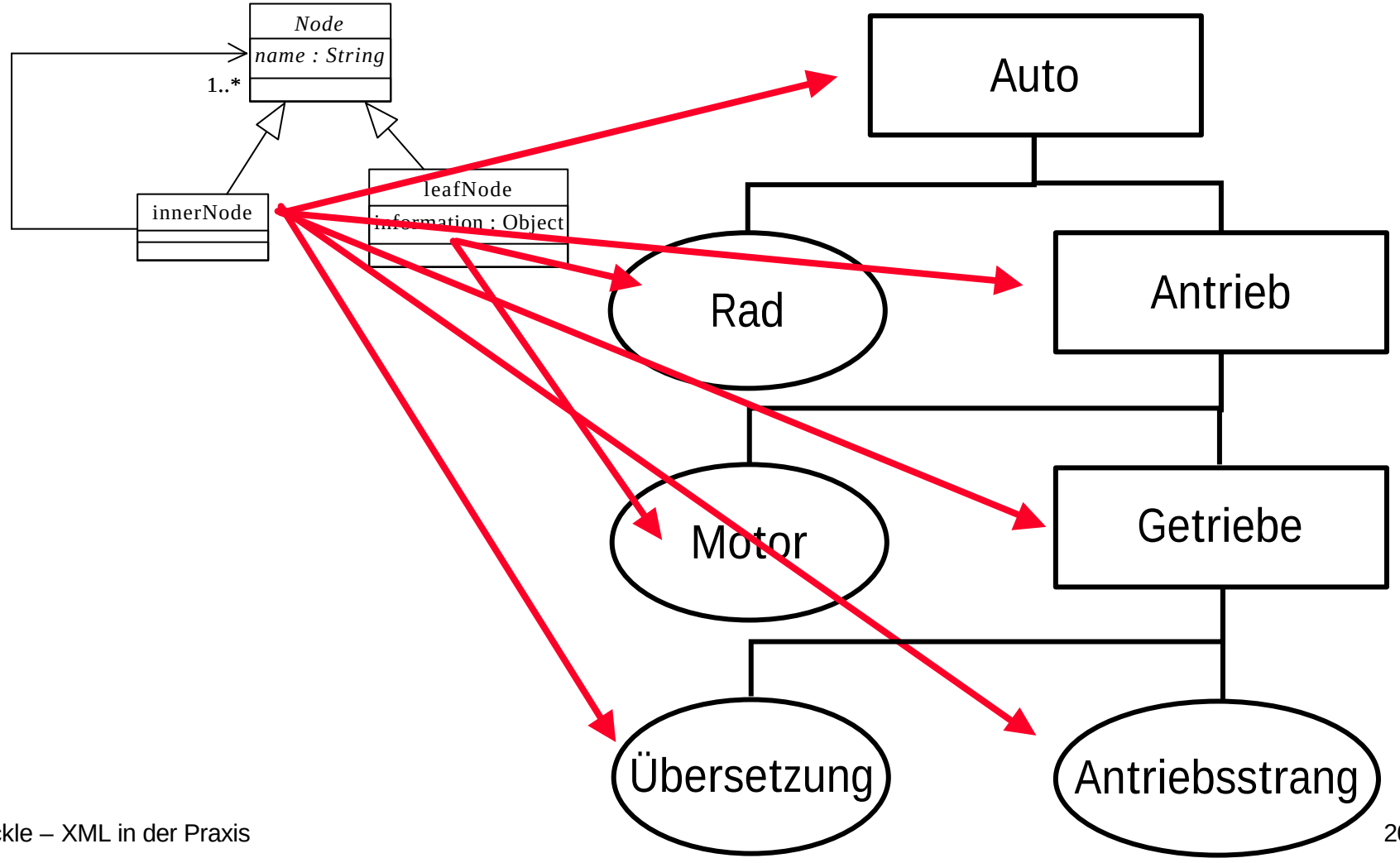
Entwurf von XML Sprachen mit XMI

Klassendiagramm (Metamodell) der zu erstellenden XML-Sprache

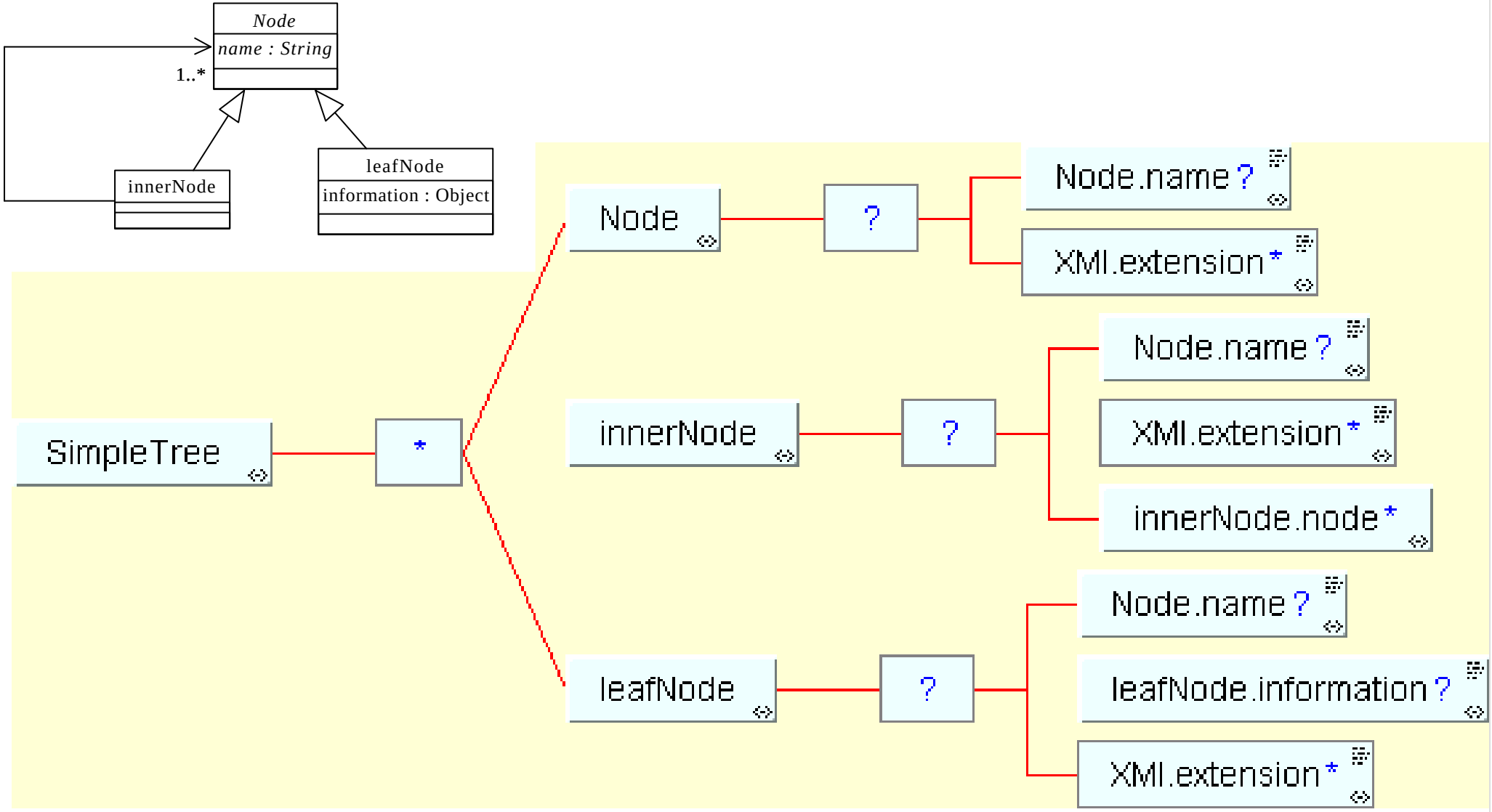


Entwurf von XML Sprachen mit XMI

Beispielausprägung (Objekte) der zu erstellenden XML-Sprache



Entwurf von XML Sprachen mit XMI



Modell und Metadatenaustausch mit XMI

Terminologie

Metadaten:

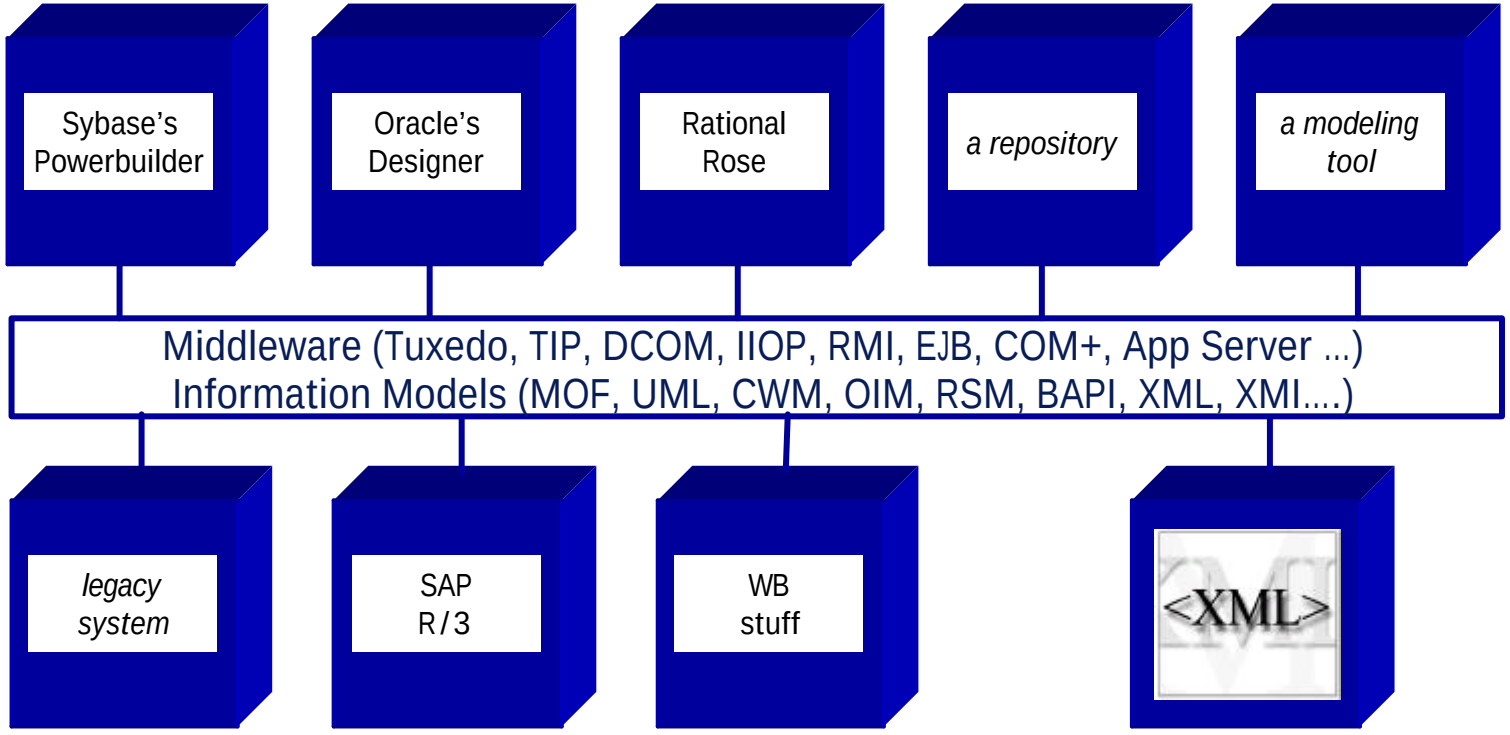
[griech. meta *inmitten, zwischen, nach, hinter*]

In der Informatik und verwandten Wissensgebieten als Wortbestandteile von *Metamodell*, *Metaklasse, Metaprogrammierung, Metasprache, Metaknowledge* verwandt.

Allgemein im Sinne einer zusätzlichen höheren Abstraktionsebene über der betrachteten Objektebene.

Modell und Metadatenaustausch mit XMI

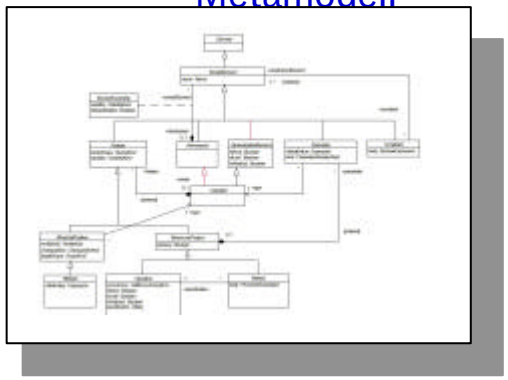
Muddleware dilemma



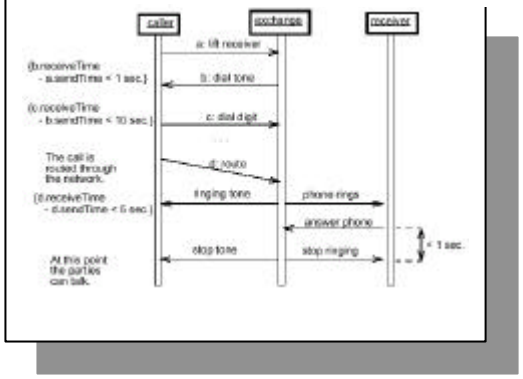
Modell und Metadatenaustausch mit XMI

Metamodell-Struktur

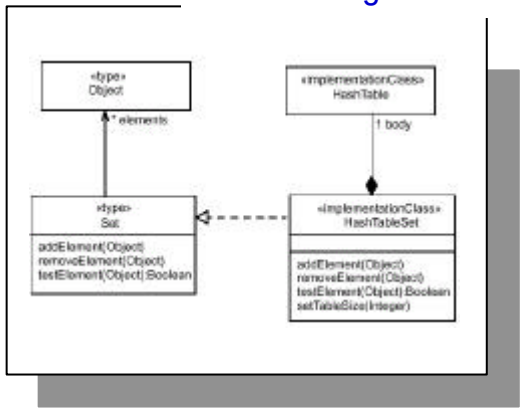
Metamodell



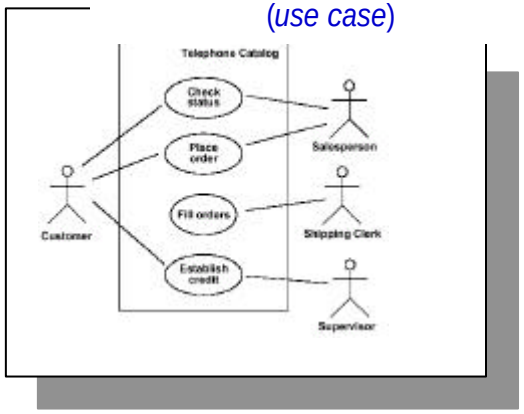
Dynamische Systemsicht
Sequenzdiagramm
(*sequence diagram*)



Statische Struktursicht
Klassendiagramm

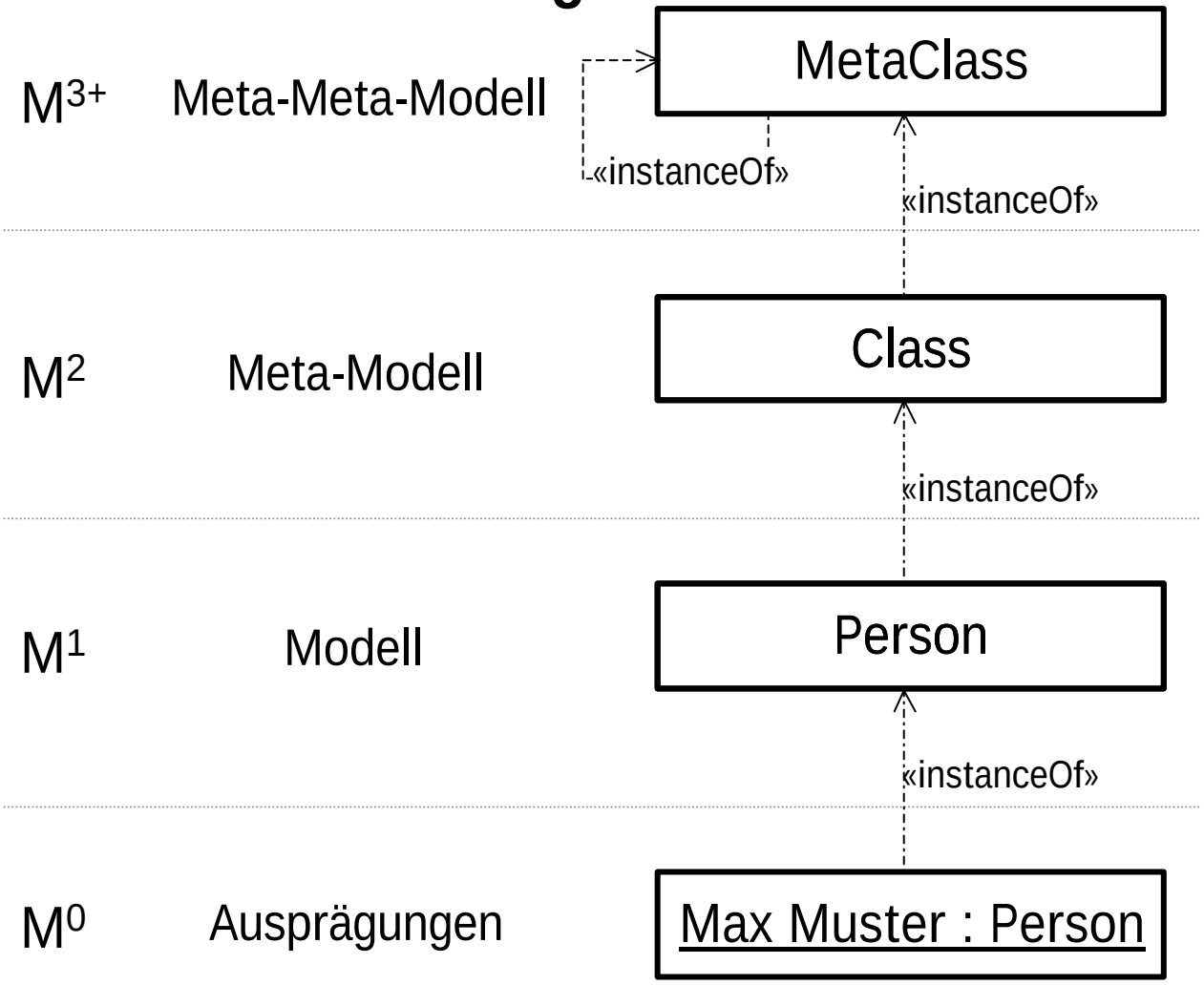


Externe Anwendungssicht
Anwendungsfall
(*use case*)



Modell und Metadatenaustausch mit XMI

Metamodellierung



Modell und Metadatenaustausch mit XMI

OMG-Metamodellhierarchie

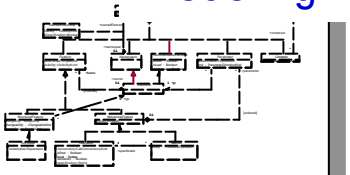
M^{3+} Meta-Meta-Modell

Meta Object Facility (MOF)

M^2 Meta-Modell

Unified Modeling Language (UML)

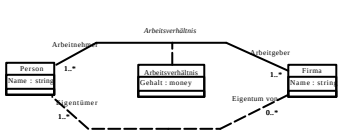
Common Warehouse Modeling Facility (CWM)



...

M^1 Modell

myModel (UML-Model)



...

myModel (CWM-Model)

Instance sets

M^0 Ausprägungen

Modell und Metadatenaustausch mit XMI

XMI[UML]

M^{3+} Meta Object Facilities
Meta Meta Model

M^2 UML & other Meta Models

UML & others as XML DTDs

M^1 Model

UML & other Models as XML Documents

M^0 Instances

XMI[UML]-DTD

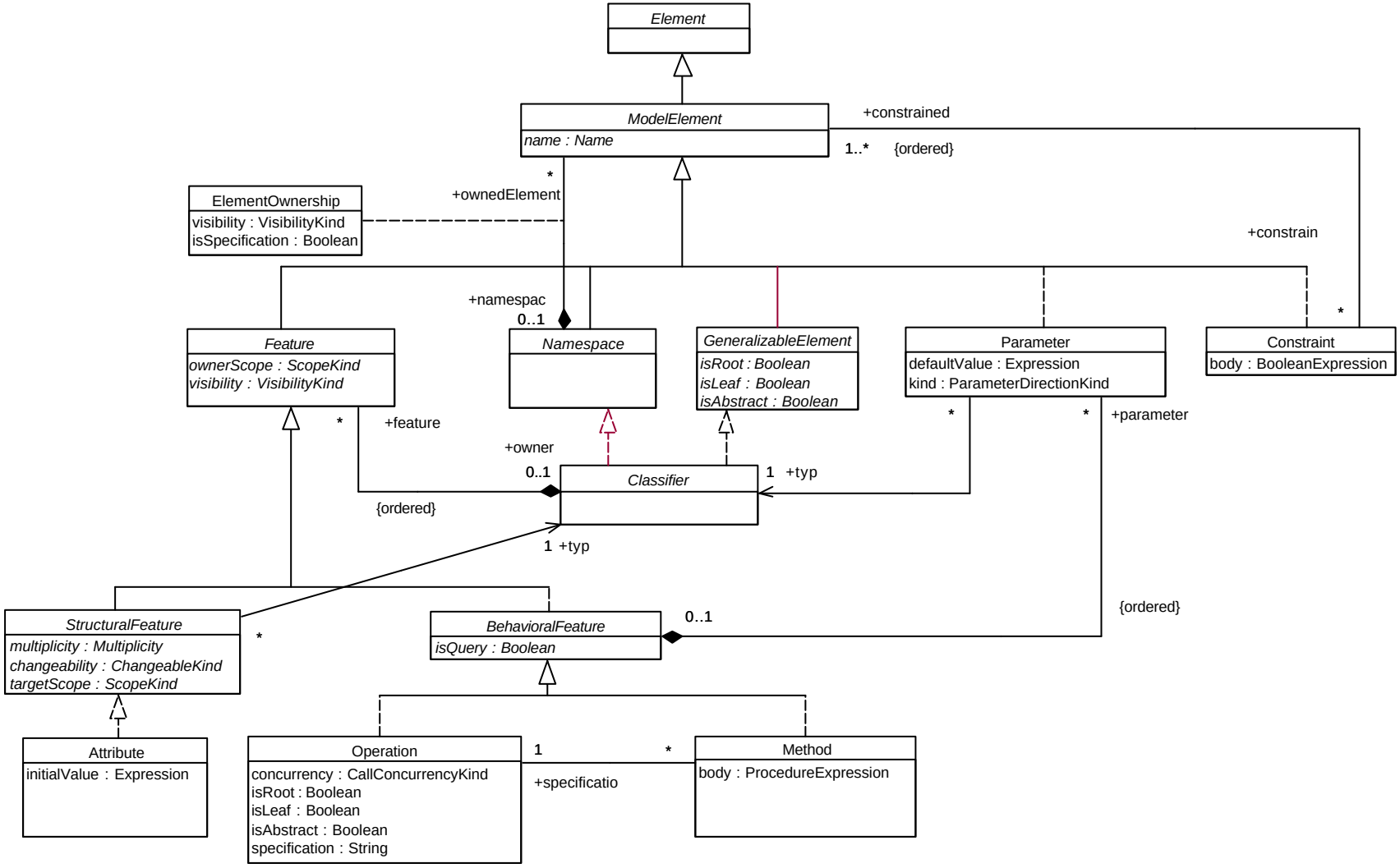
```
<!ELEMENT XMI  (XMI.header? , XMI.content? , XMI.difference* , XMI.extensions* )>
<!ATTLIST XMI  xmi.version CDATA  #FIXED '1.1'
          timestamp   CDATA  #IMPLIED
          verified    (true | false )  #IMPLIED
          xmlns:UML   CDATA  #IMPLIED >
<!– ____________________________________________________________ -->
<!–                                                              -->
<!– XMI.header contains documentation and identifies the model,    -->
<!– metamodel, and metametamodel                                  -->
<!– ____________________________________________________________ -->
<!ELEMENT XMI.header  (XMI.documentation? , XMI.model* ,
XMI.metamodel* , XMI.metametamodel* , XMI.import* )>

<!– ____________________________________________________________ -->
<!–                                                              -->
<!– documentation for transfer data                               -->
<!– ____________________________________________________________ -->
<!ELEMENT XMI.documentation  (#PCDATA | XMI.owner | XMI.contact
| XMI.longDescription | XMI.shortDescription | XMI.exporter
 | XMI.exporterVersion | XMI.notice )*>
```

XMI[UML]-Document

```
<XMI xmi.version="1.0" timestamp="Sat May 13 16:07:52 GMT+02:00 2000">
  <XMI.header>
    <XMI.documentation>
      <XMI.exporter>XMI Toolkit</XMI.exporter>
      <XMI.exporterVersion>1.05</XMI.exporterVersion>
    </XMI.documentation>
    <XMI.metamodel xmi.name="UML" xmi.version="1.1"/>
  </XMI.header>
  <XMI.content>
    <Model_Management.Model xmi.id="_1"
xmi.uuid="DCE:DDBE7E10-28D7-11D4-9EDF-0060978E3286:1">
<Foundation.Core.ModelElement.name>SimpleTree</Foundation.Core.ModelElement.name>
      <XMI.extension xmi.extender="IXT" xmi.extenderID="">
        <ixts s="Rose">
          <ixttv t="quid" v="391D039901A0"/>
          <ixttv t="file_name" v="SimpleTree.mdl"/>
        </ixts>
      </XMI.extension>
      <XMI.extension xmi.id="_1.8"
xmi.uuid="DCE:DDBE7E30-28D7-11D4-9EDF-0060978E3286:1"
xmi.extender="Rose" xmi.extenderID="Rose">
```

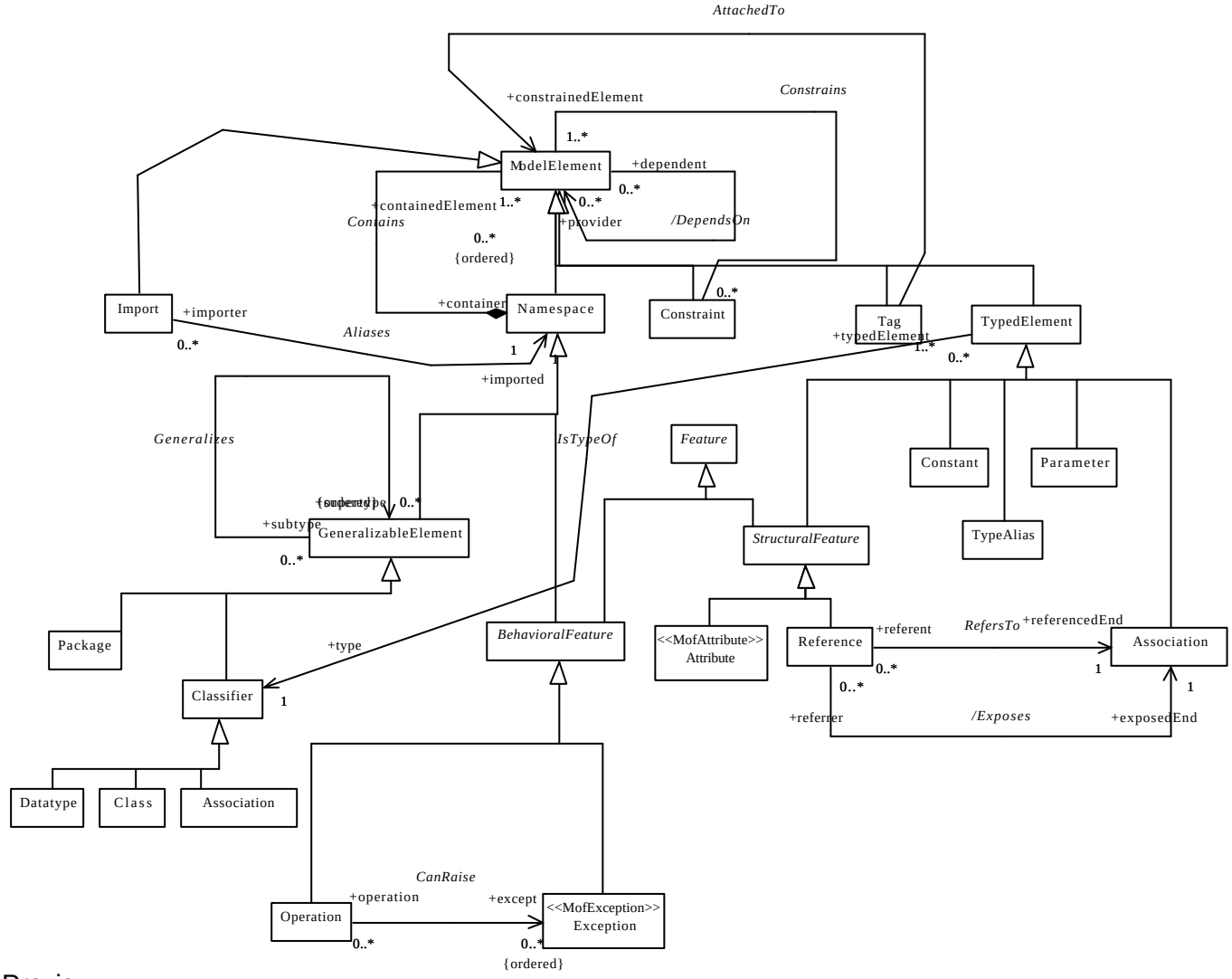
Modell und Metadatenaustausch mit XMI

UML-Metamodell (package *backbone*)



Modell und Metadatenaustausch mit XMI

MOF-Meta-Metamodell *(package model)*

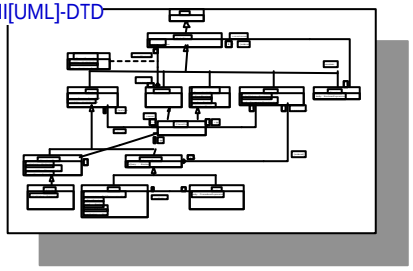


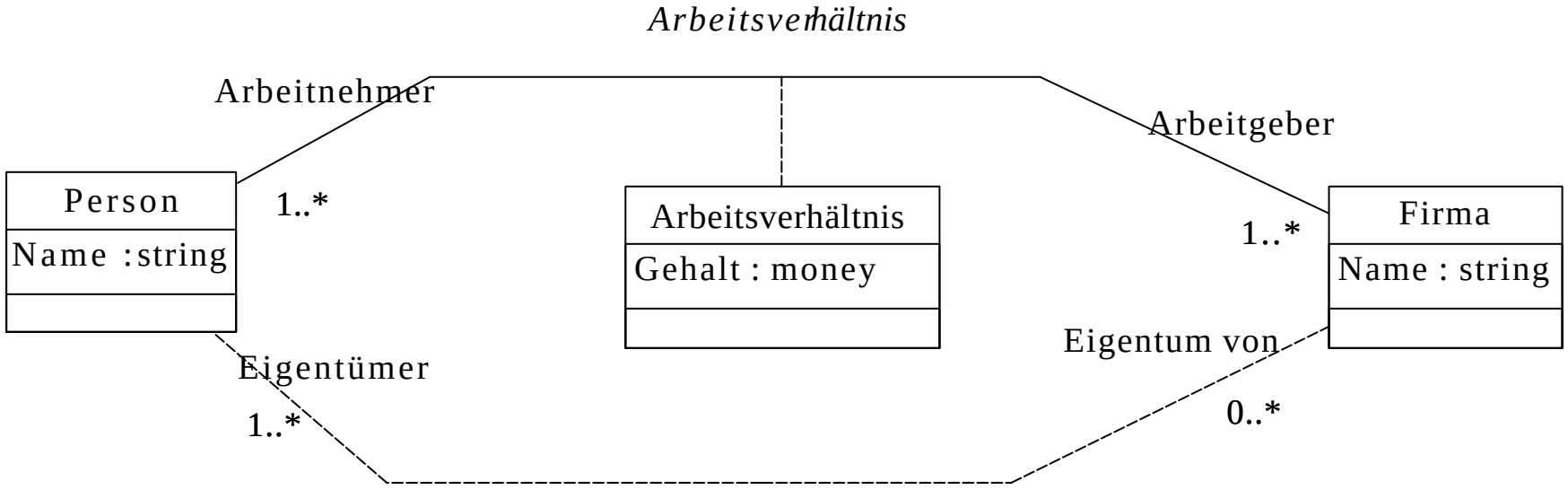
Modell und Metadatenaustausch mit XMI

XMI-Encoding eines UML-Klassendiagramms

M^{3+}	MetaObject Facilities Meta MetaModel	
M^2	UML & other MetaModels	UML & others as XML DTDs
M^1	Model	UML & other Models as XML Documents
M^0	Instances	

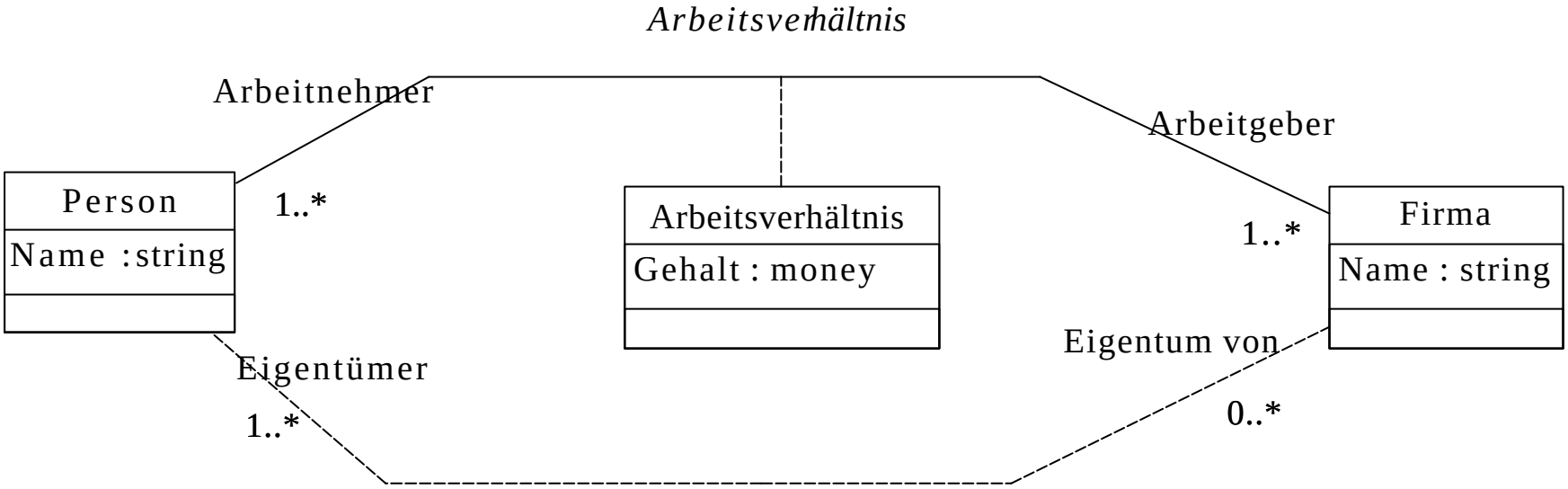
XMI[UML]-DTD





Modell und Metadatenaustausch mit XMI

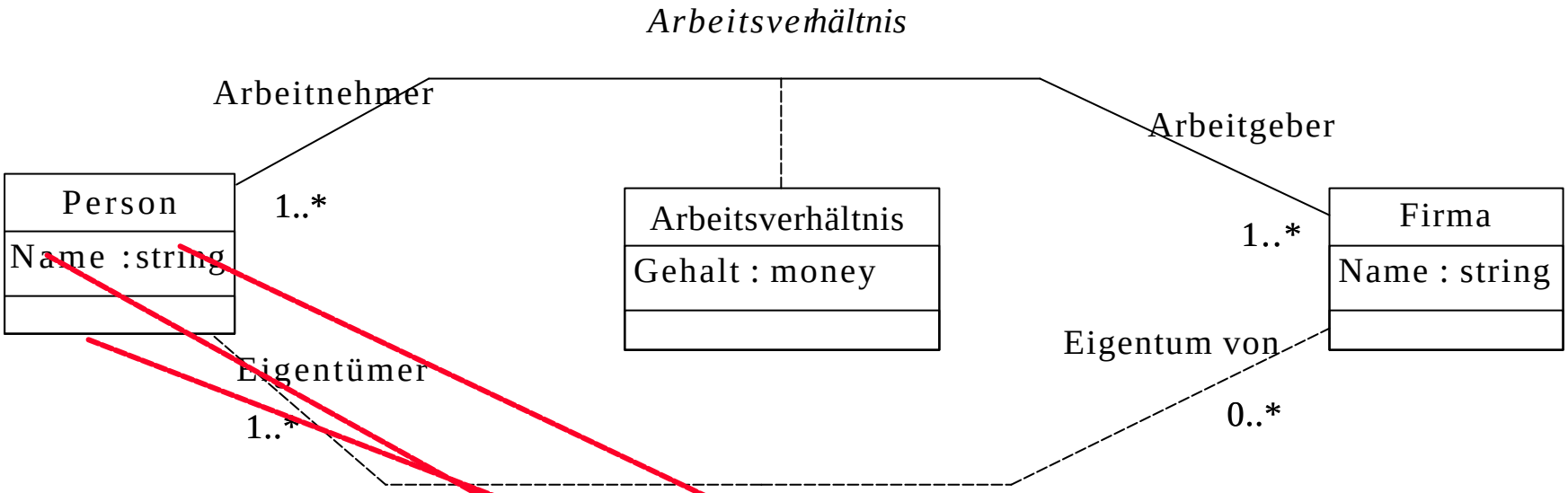
XMI-Encoding eines UML-Klassendiagramms




```
<XMI timestamp="2000-10-09T17:00:00" verified="true” xmi.version=“1.1”>
  <XMI.header>
    <XMI.model xmi.name="SimpleClassModel”/>
    <XMI.metamodel xmi.name="UML” xmi.version=“1.3”/>
</XMI.header>
```

Modell und Metadatenaustausch mit XMI

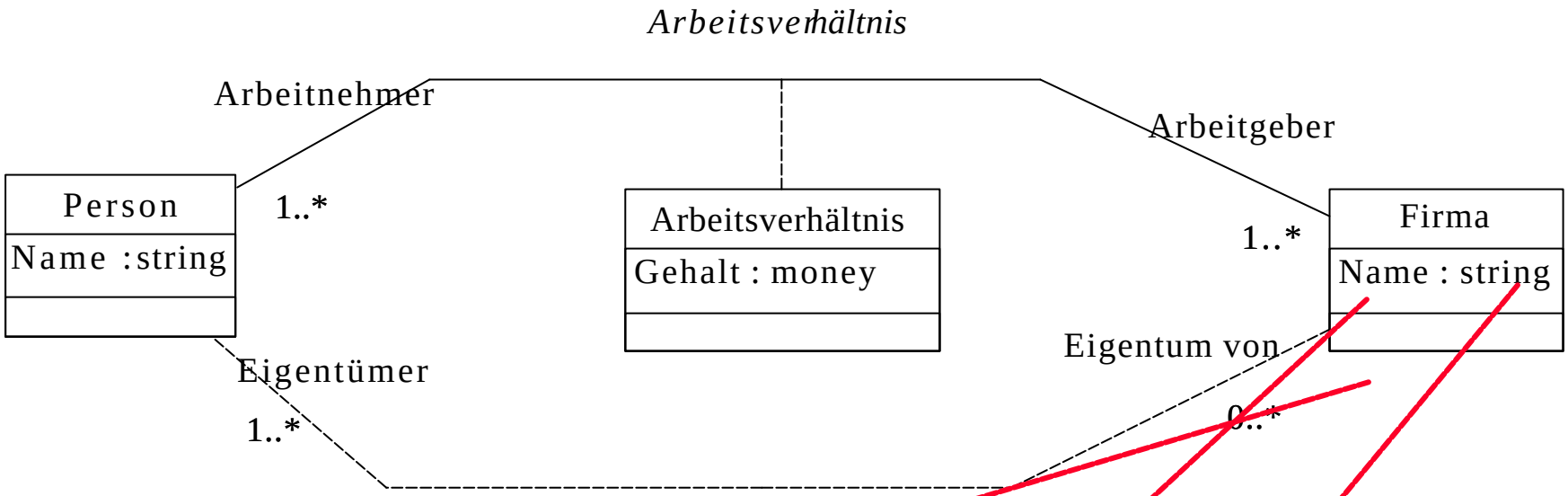
XMI-Encoding eines UML-Klassendiagramms




```
<XMI.content>
    <UML:Class name="Person" xmi.id="Person">
      <UML:Classifier.feature>
        <UML:Attribute name="Name" type="string"/>
      </UML:Classifier.feature>
    </UML:Class>
```

Modell und Metadatenaustausch mit XMI

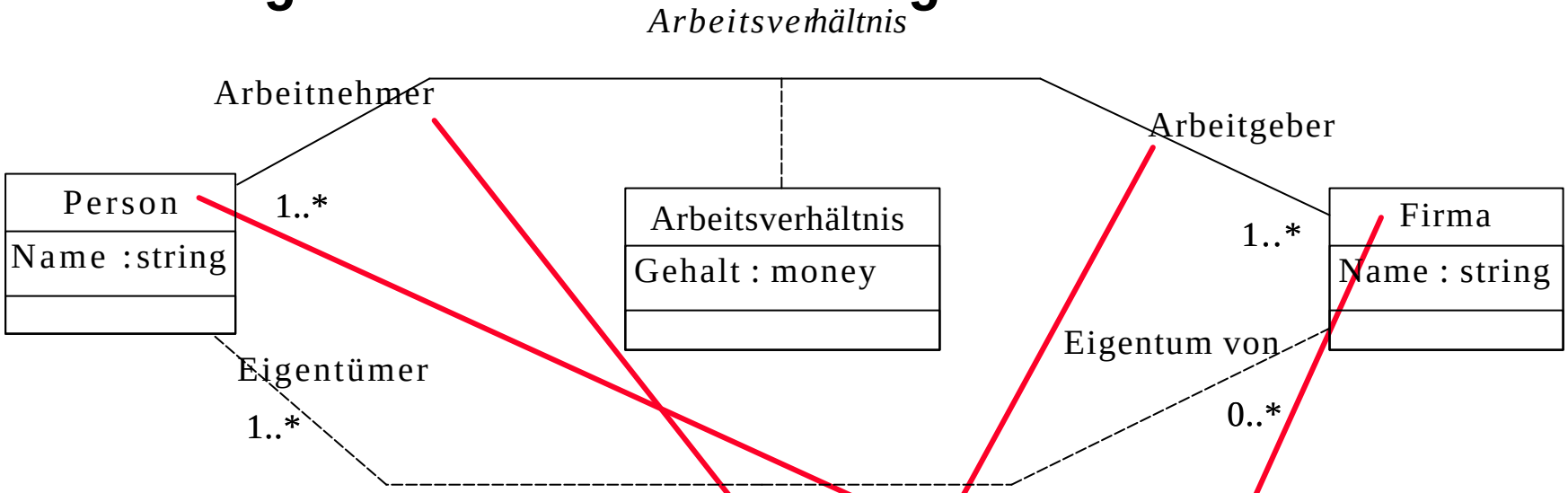
XMI-Encoding eines UML-Klassendiagramms




```
<UML:Class name="Firma" xmi.id="Firma">
  <UML:Classifier.feature>
    <UML:Attribute name="Name" type="string" />
  </UML:Classifier.feature>
</UML:Class>
```

Modell und Metadatenaustausch mit XMI

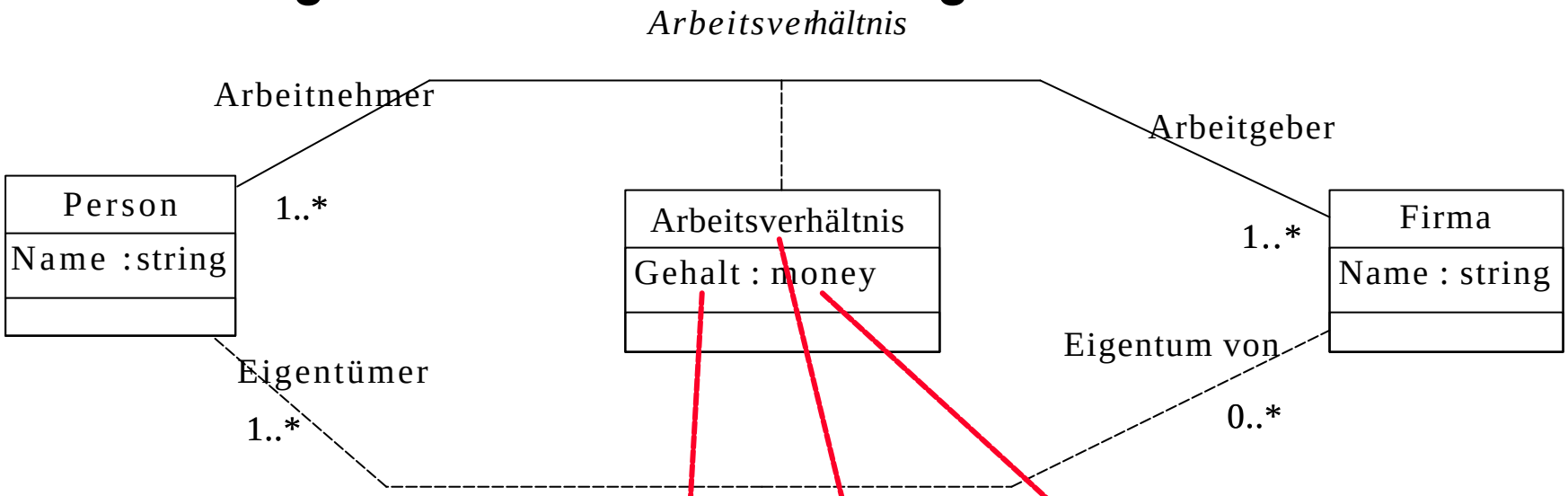
XMI-Encoding eines UML-Klassendiagramms




```
<UML:Association>
  <UML:Association.connection>
    <UML:AssociationEnd name="Arbeitnehmer" type="Person"/>
    <UML:AssociationEnd name="Arbeitgeber" type="Firma"/>
  </UML:Association.connection>
</UML:Association>
```

Modell und Metadatenaustausch mit XMI

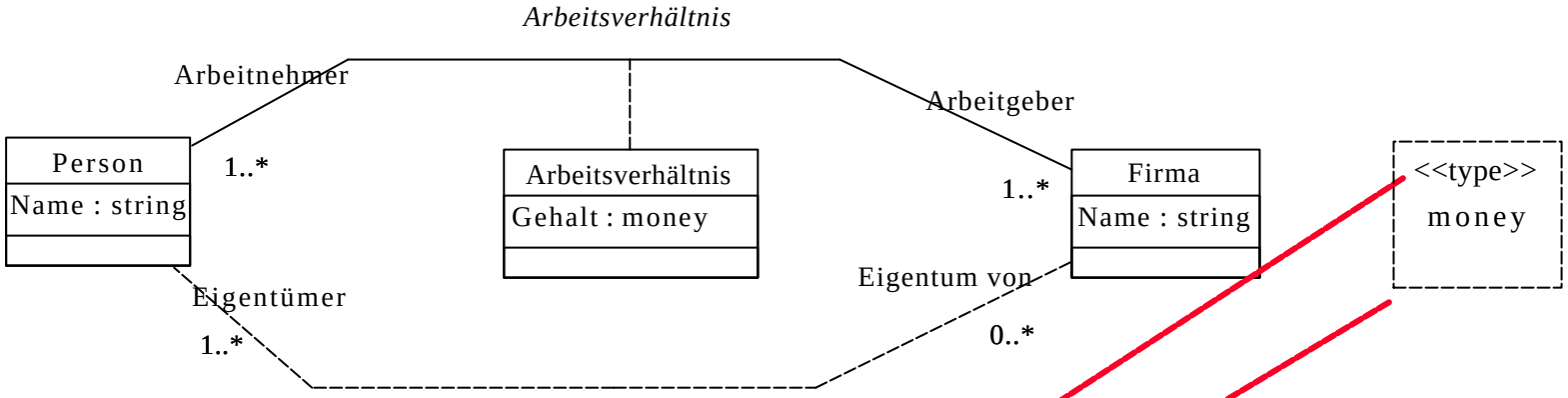
XMI-Encoding eines UML-Klassendiagramms




```
<UML:AssociationClass name="Arbeitsverhältnis">
  <UML:Classifier.feature>
    <UML:Attribute name="Gehalt" multiplicity="1..1" type="money"/>
  </UML:Classifier.feature>
</UML:AssociationClass>
```

Modell und Metadatenaustausch mit XMI

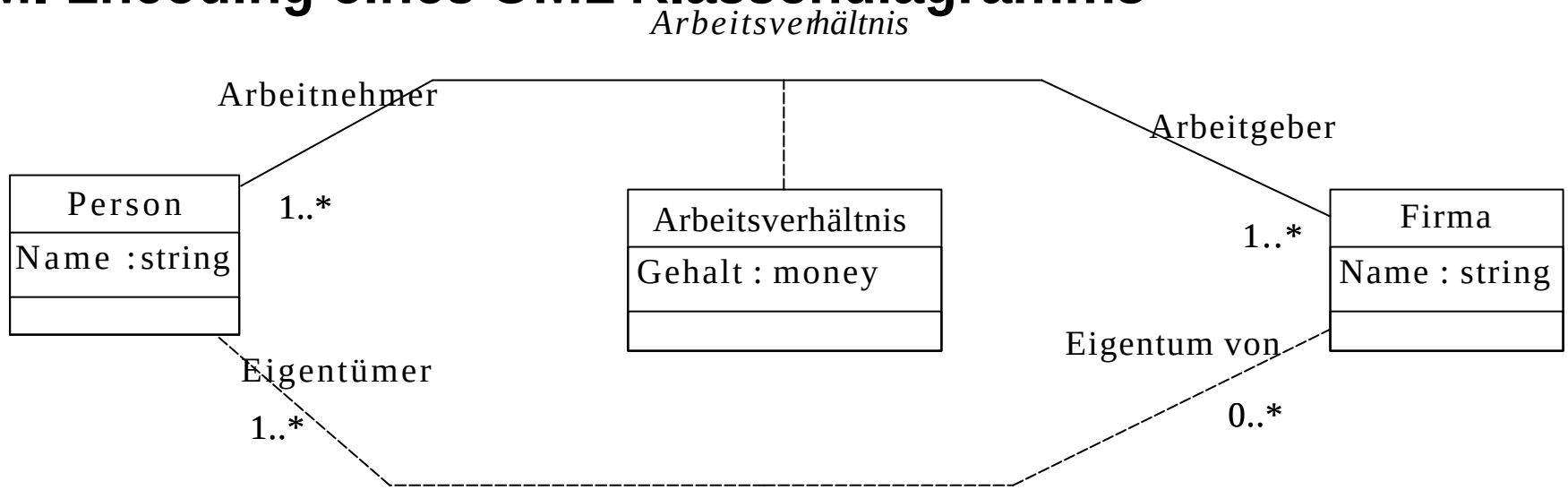
XMI-Encoding eines UML-Klassendiagramms




```
<UML:Stereotype name="type" xmi.id="type”/>
<UML:Class name="money" stereotype="type" xmi.id="money”/>
```

Modell und Metadatenaustausch mit XMI

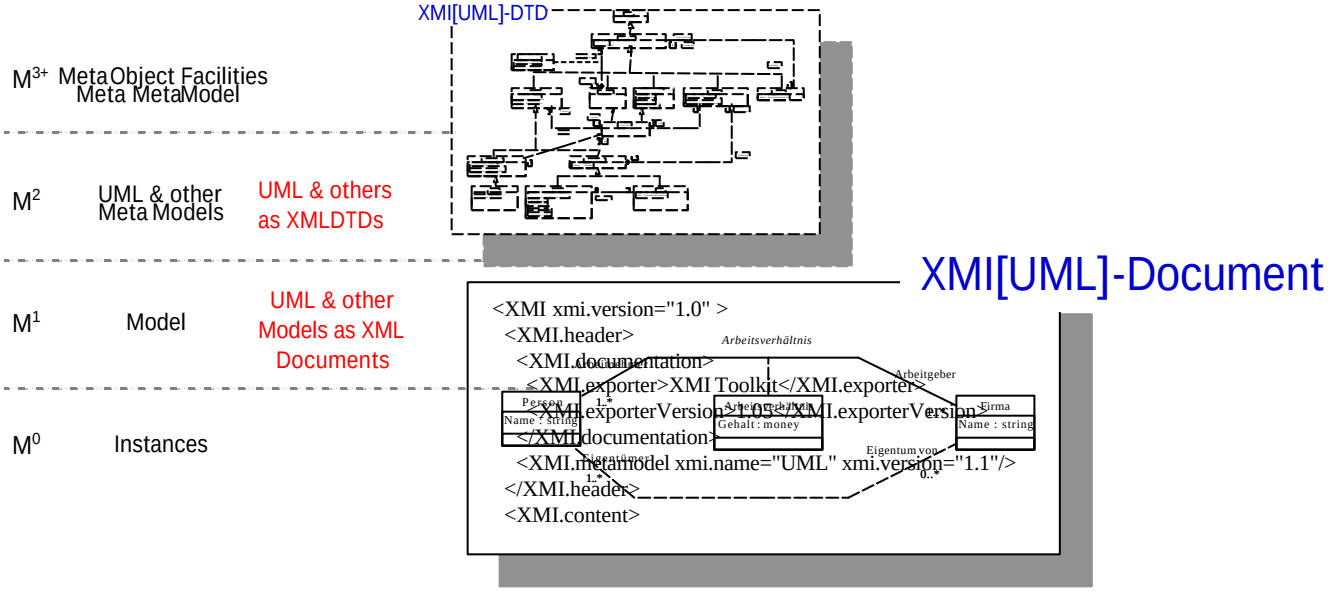
XMI-Encoding eines UML-Klassendiagramms




```
<UML:Association>
  <UML:Association.connection>
    <UML:AssociationEnd name="Eigentümer" multiplicity="1..*"  type="Person"/>
  <UML:AssociationEnd name="Eigentum von" multiplicity="1..*" type="Firma"/>
  </UML:Association.connection>
</UML:Association>
```

Modell und Metadatenaustausch mit XMI

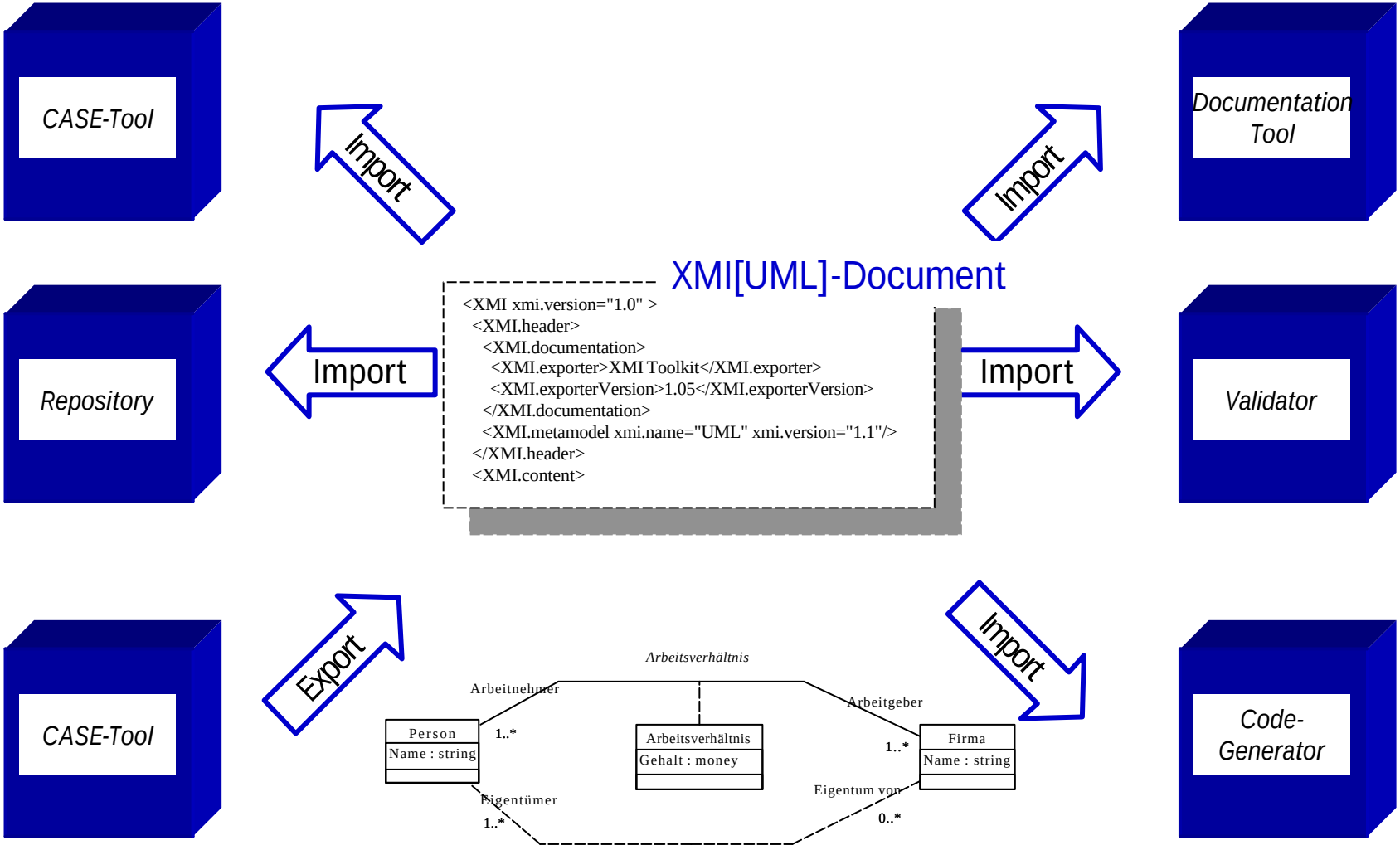
XMI-Encoding eines UML-Klassendiagramms



- Modellierungswerkzeugunabhängige Darstellung beliebiger UML-Modelle
- Modellvalidierung (*modeling guidelines*)
- Werkzeugunabhängige Codegenerierung (-> XSL(T))
- Design-Pattern Libraries

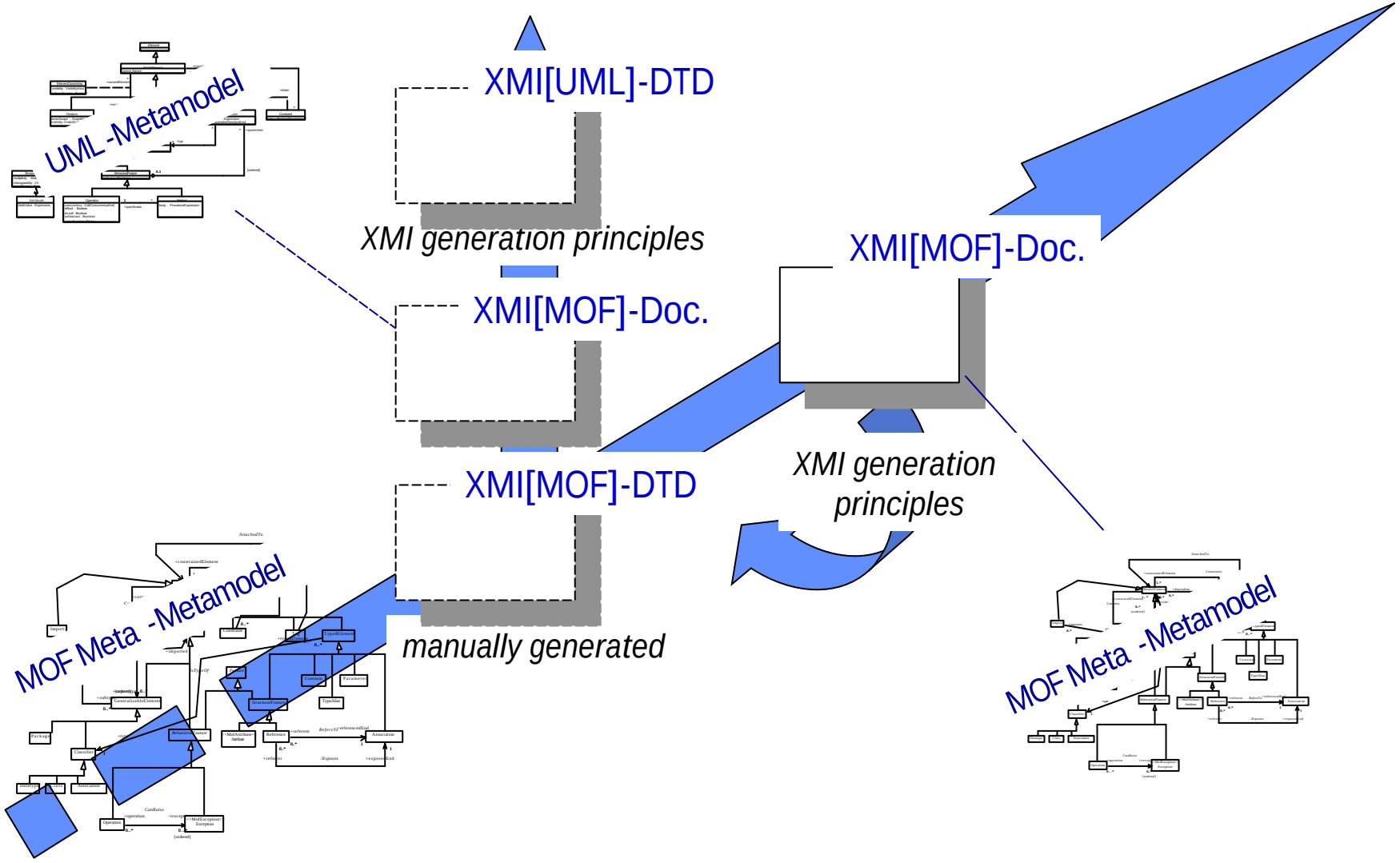
Modell und Metadatenaustausch mit XMI

XMI-Encoding eines UML-Klassendiagramms



Modell und Metadatenaustausch mit XMI

Entwicklung von XMI ...



XML Metadata Interchange Format

IBM XMI Toolkit

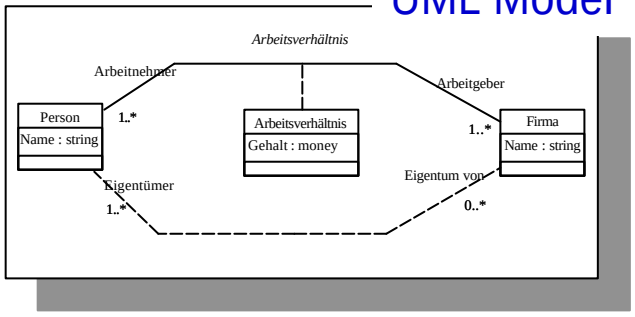
XMI[UML]-DTD / Schema

```
<!ELEMENT XMI  (XMI.header? , XMI.content? ,
<!ATTLIST XMI  xmi.version CDATA  #FIXED '1.
          timestamp   CDATA  #IMPLIED
          verified    (true | false )  #IMPLIED
          xmlns:UML   CDATA  #IMPLIED >
<!-- ____________________________________________________________-->
<!--                                                            -->
<!-- XMI.header contains documentation and identifies the model,  -->
<!-- metamodel, and metametamodel                                 -->
<!-- ____________________________________________________________-->
<!ELEMENT XMI.header  (XMI.documentation? , XMI.model* ,
XMI.metamodel* , XMI.metametamodel* , XMI.import* )>

<!-- ____________________________________________________________-->
<!--                                                            -->
<!-- documentation for transfer data                              -->
<!-- ____________________________________________________________-->
<!ELEMENT XMI.documentation  (#PCDATA | XMI.owner | XMI.contact
| XMI.longDescription | XMI.shortDescription | XMI.exporter
| XMI.exporterVersion | XMI.notice )*>
```

XMI[UML]-Document

```
<XMI xmi.version="1.0" >
  <XMI.header>
    <XMI.documentation>
      <XMI.exporter>XMI Toolkit</XMI.exporter>
      <XMI.exporterVersion>1.05</XMI.exporterVersion>
    </XMI.documentation>
    <XMI.metamodel xmi.name="UML" xmi.version="1.1"/>
  </XMI.header>
  <XMI.content>
```

UML Model



Rational Rose

CORBA IDL

...

Modell und Metadatenaustausch mit XMI

Warum XMI?

- XMI ist *verabschiedeter OMG-Standard*
  - Die OMG ist mit über 800 Mitgliedern das weltgrößte Software-Konsortium
- XMI integriert die *Unified Modeling Language* (UML)
- XMI basiert auf der *Meta Object Facility* (MOF)
- XMI-Anwendungsgebiete
  - Austausch von UML-Modellen

    (Rational Rose, Together, MID's Innovator, Software through Pictures, OTW)
  - Middleware-bezogene Komponenteninformation (CORBA Components)
  - Austausch von Data Warehouse Models (CWM)
  - Vertikale Datenintegration (e.g. clincal information)
- Auf Basis von XMI lassen sich beliebige DTDs aus UML-Datenmodellen generieren
- XMI reagiert auf zukünftige Entwicklungen im Umfeld XML (XML Schema)

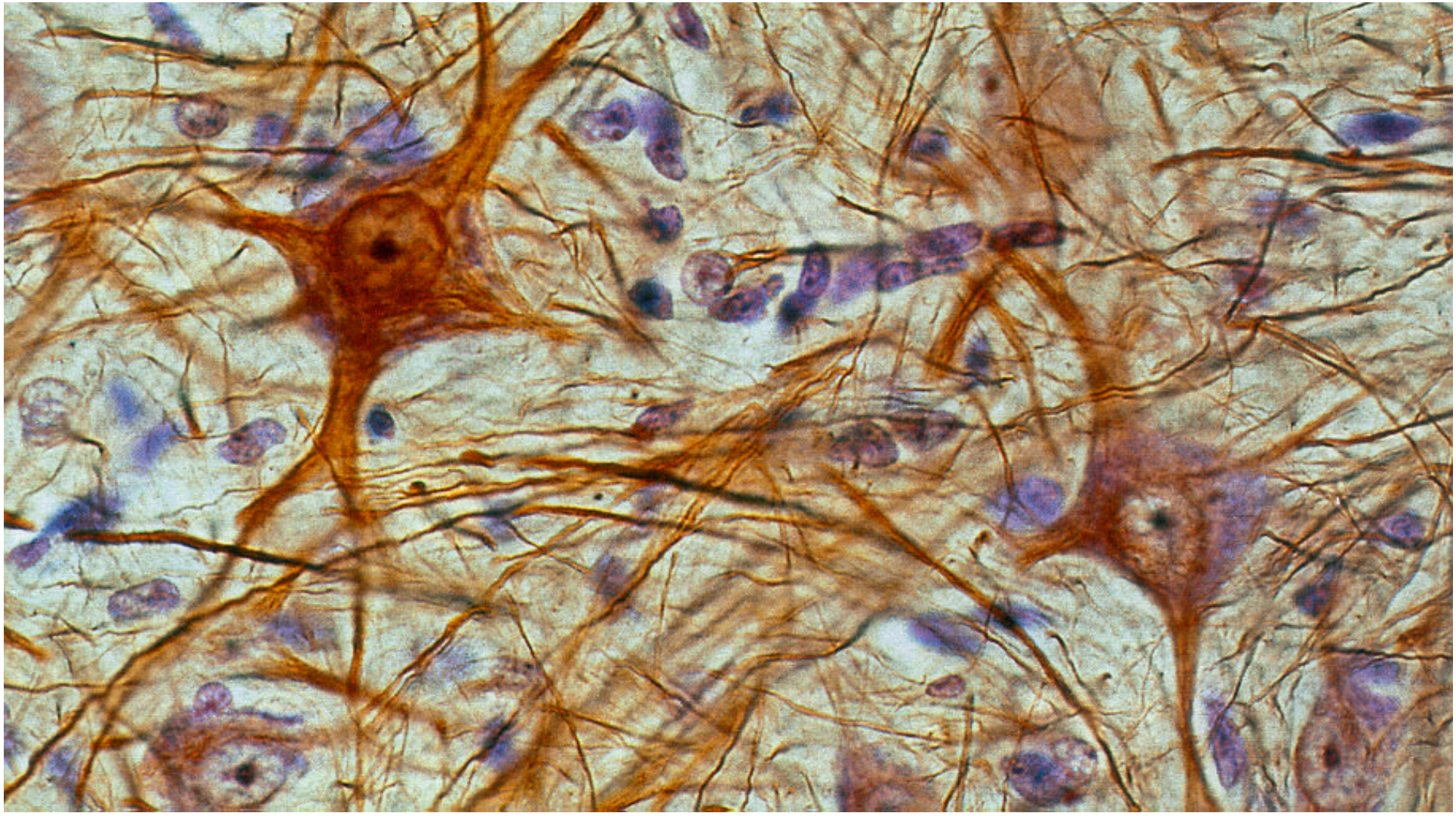
- *Meta Data Coalition* (MDC) hat Mapping MDC's OIM auf XMI angekündigt. Eine XMI-DTD für OIM existiert (XMI[OIM])

Modell und Metadatenaustausch mit XMI

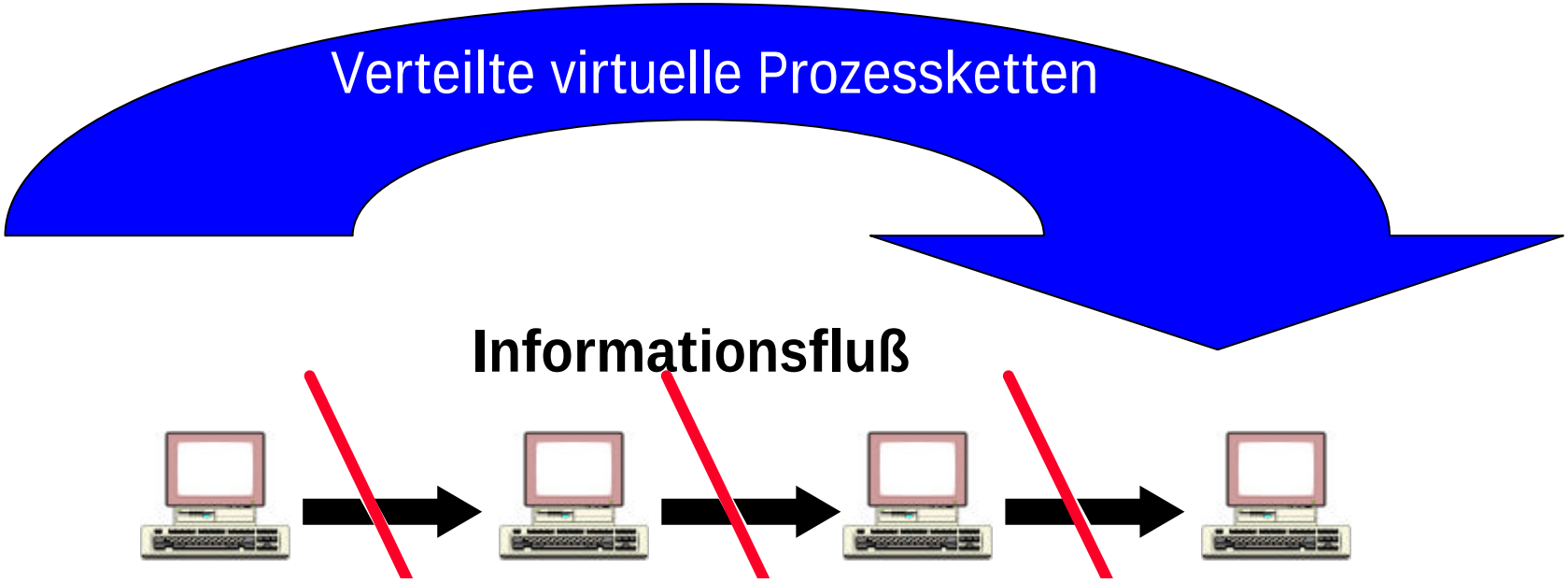
Erfahrungen aus dem praktischen Einsatz

- XMI v1.1 ist stabil und einsetzbar
  - DTD ist schlanker als v1.0 => kleinere Dokumente, weniger Overhead
  - Struktur intuitiver und flexibler
- Werkzeugunterstützung wächst stetig
- automatisierte DTD-Generierung beschleunigt XML-Integration in laufenden Entwicklungsprozeß (inbesondere bei Schemaänderungen)
- XMI als XML-Sprache eröffnet Zugang zur XML-Sprachfamilie (XSL(T)!)

Fallstudie *Systemintegration mit XML*

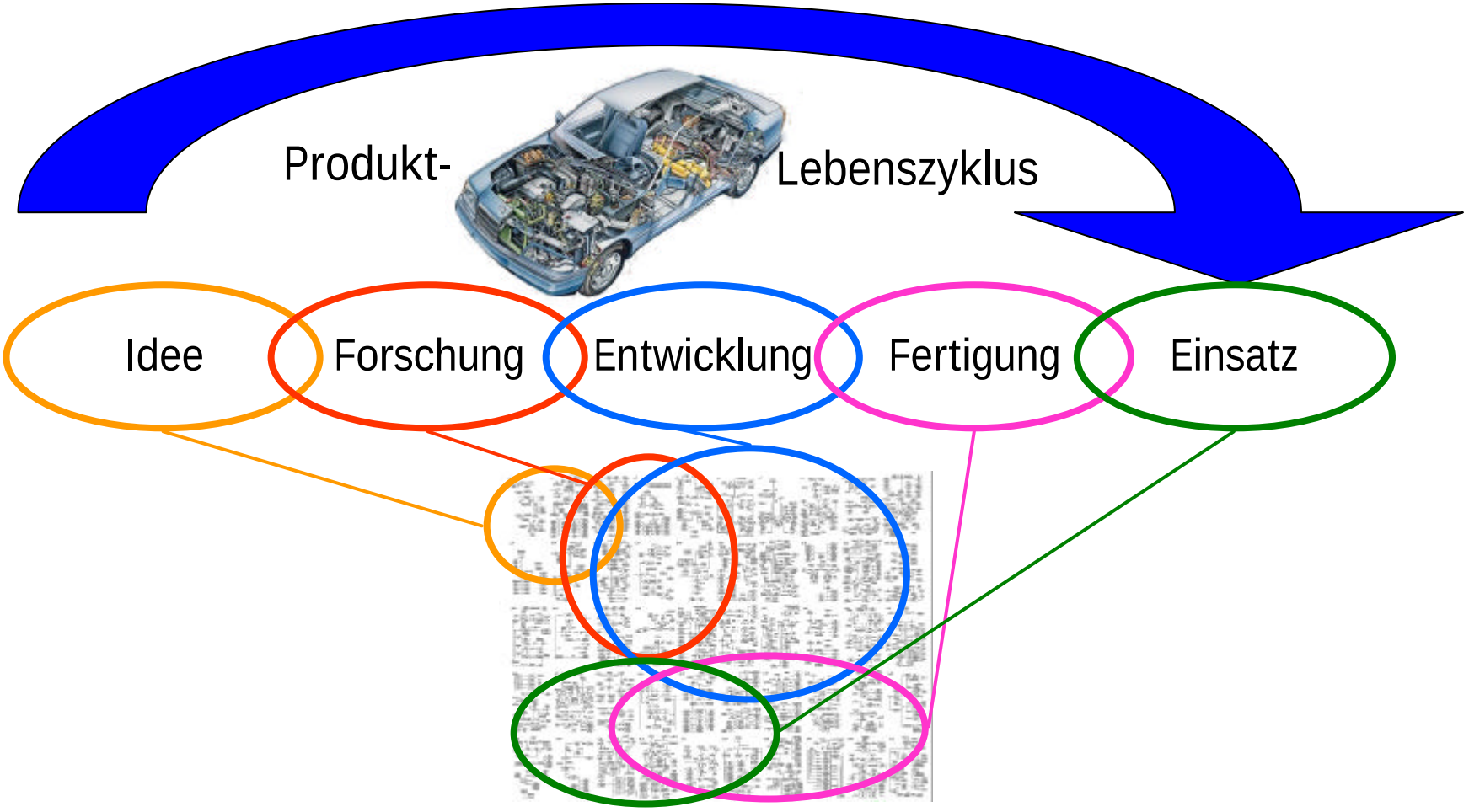


Fallstudie *Systemintegration mit XML*



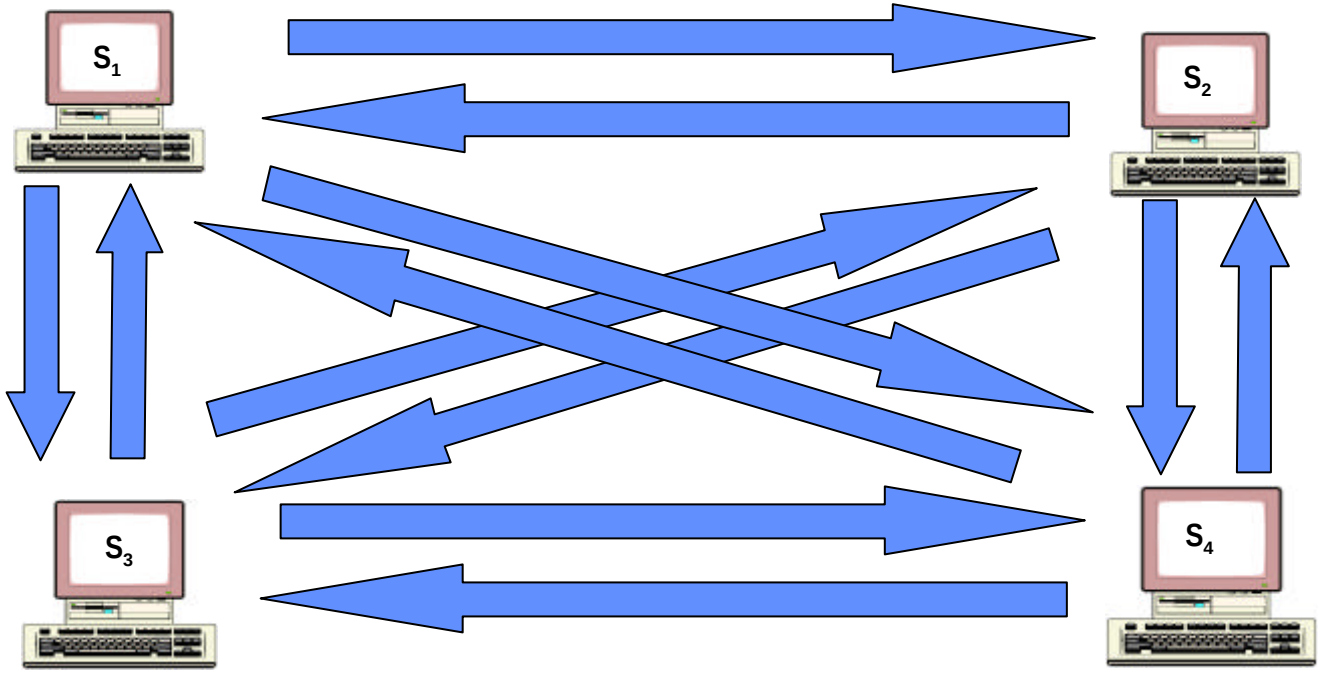
Dynamische verteilte virtuelle Prozesse und Prozeßketten bilden das erfolgskritische Rückrad moderner Unternehmen

Fallstudie *Systemintegration mit XML*

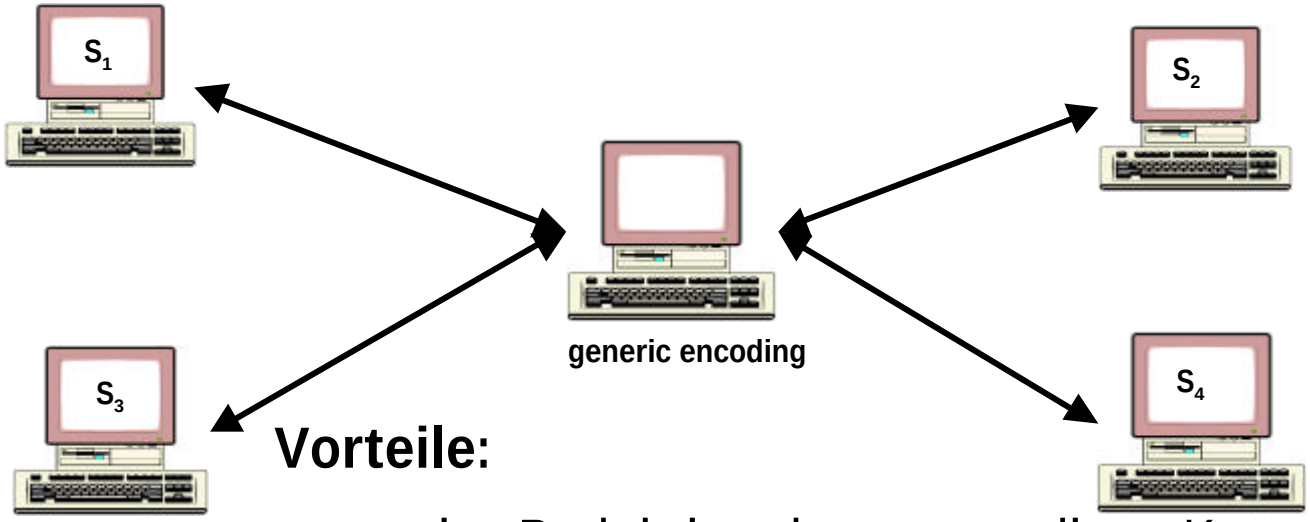


Integrierte Entwicklungsprozesse bedingen Paradigmenwechsel, vom klassischen Teilnehmer- hin zum Teilhaberbetrieb.

Fallstudie *Systemintegration mit XML*



Fallstudie *Systemintegration mit XML*



Vorteile:

- massive Reduktion der notwendigen Konvertierungsvorgänge (n^2-n vs. n)

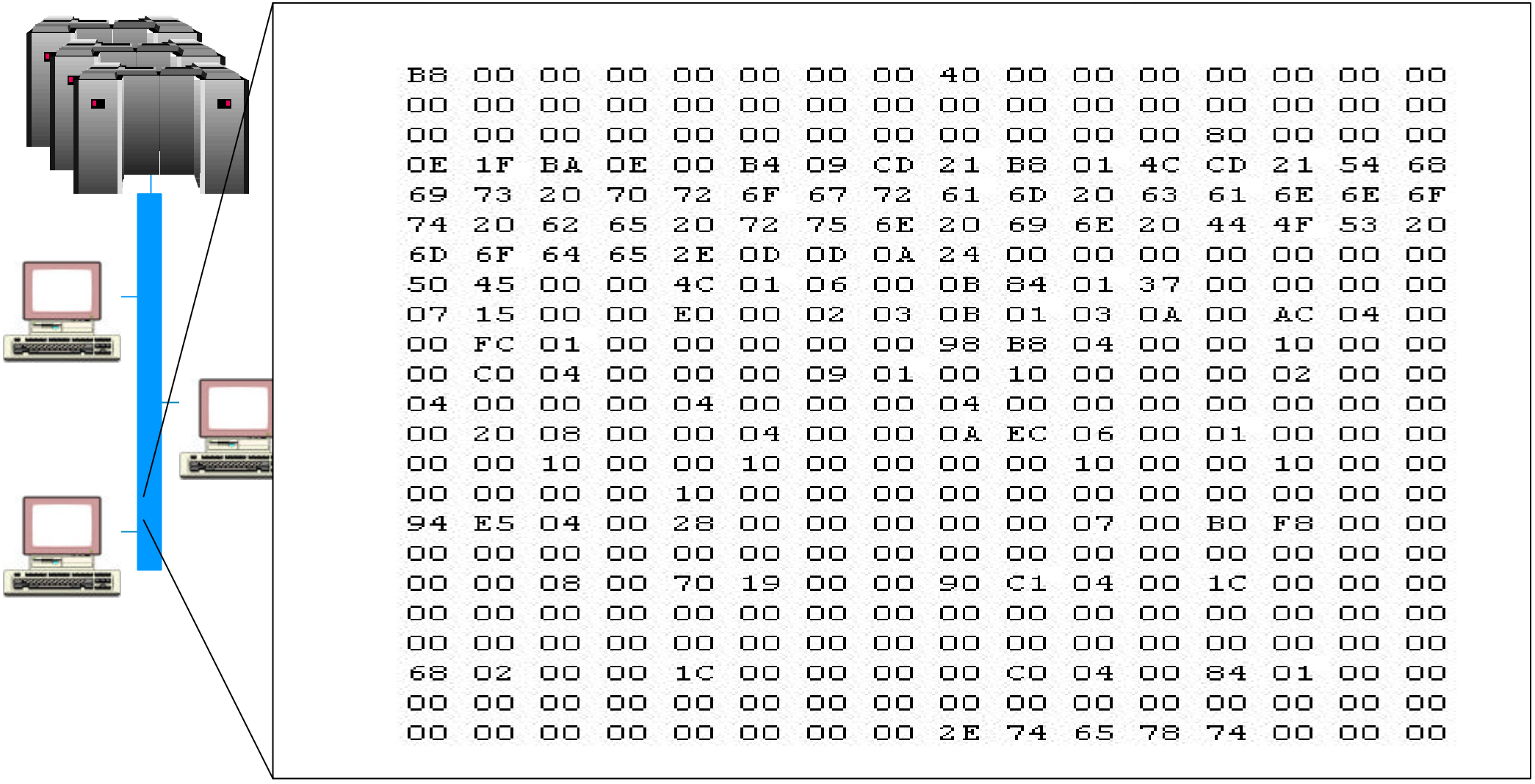
Nachteile:

-Informationsverlust durch Generalisierungsprozess

-Semantikverlust durch Abstraktion

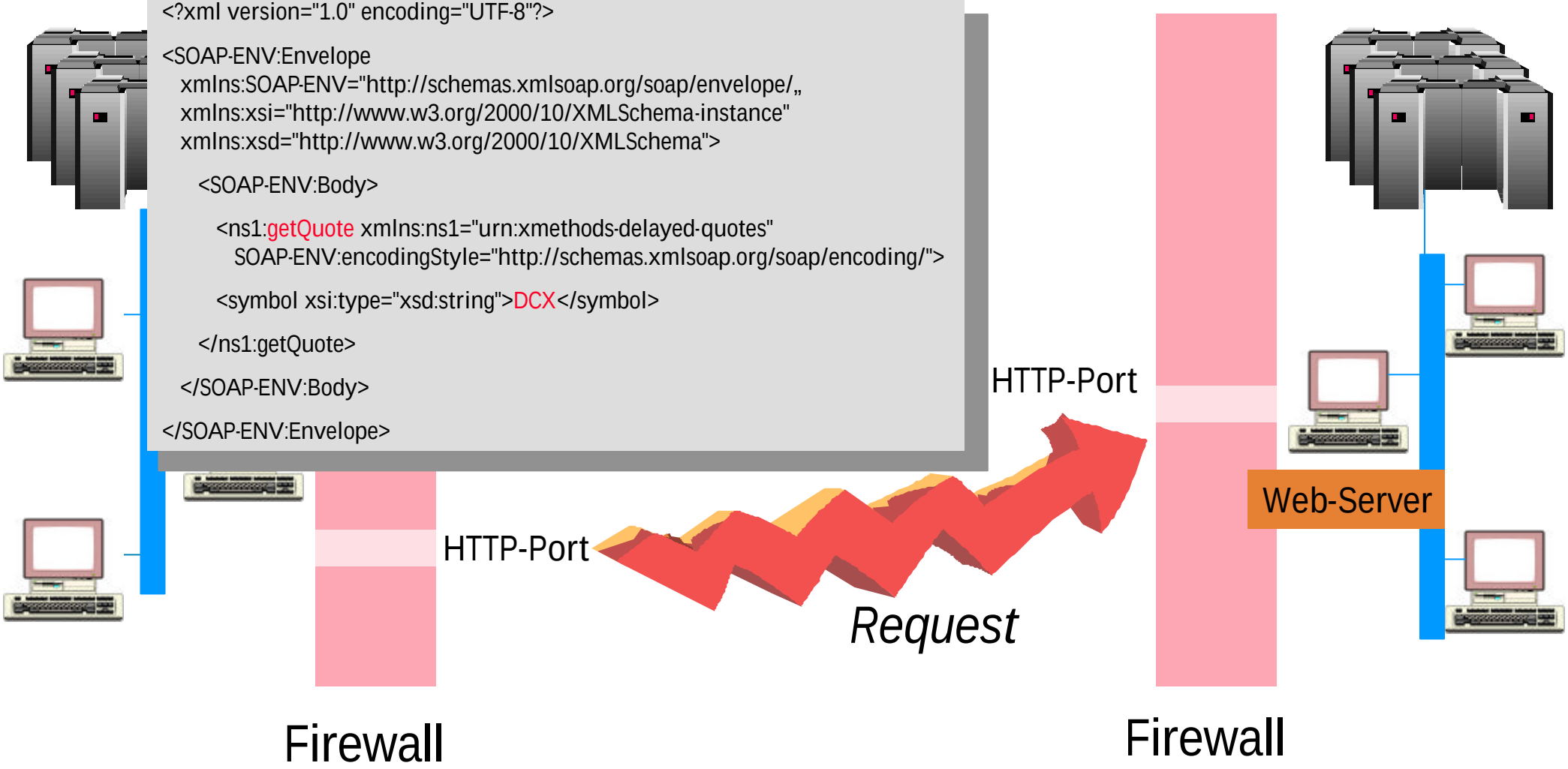
-Codierungsaufwand der Transformationsprozessoren

Fallstudie *Systemintegration mit XML*

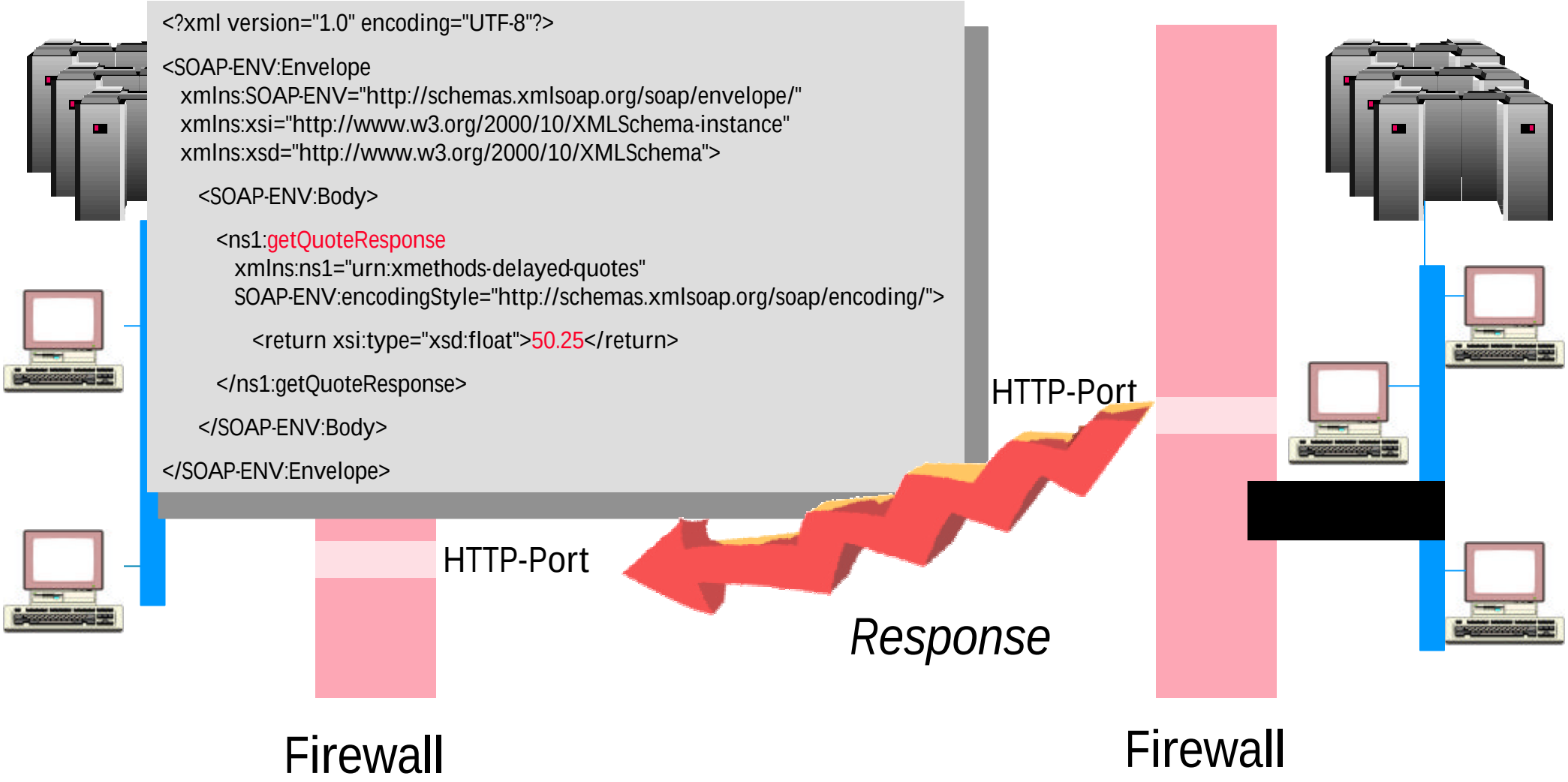


Fallstudie *Systemintegration mit XML*

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/„
  xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
    <SOAP-ENV:Body>
      <ns1:getQuote xmlns:ns1="urn:xmethods-delayed-quotes"
        SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <symbol xsi:type="xsd:string">DCX</symbol>
    </ns1:getQuote>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

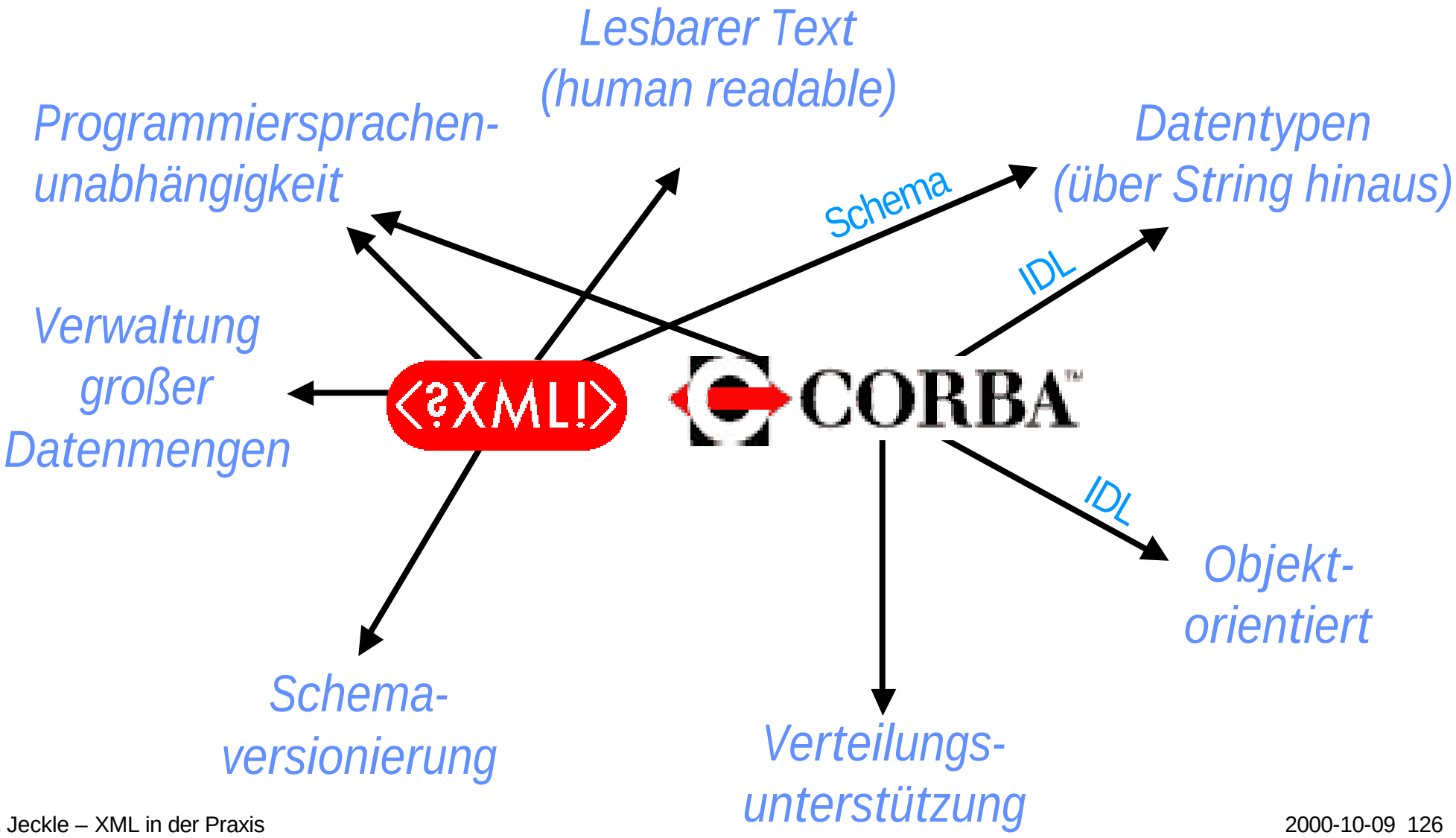


Fallstudie *Systemintegration mit XML*



Fallstudie *Systemintegration mit XML*

CORBA vs. XML



Fallstudie *Systemintegration mit XML*

Prozesskette und eingesetzte Techniken

- Modellierung: UML
- XML-Sprachgenerierung: XMI, XSD
- Systemkopplung: SOAP
- Interaktion: XML-basierte Anfragesprache
- Transformation: XSLT

Referenzen

XML-Schema:

W3C's XML-Schema Description Language (XSD):

- www.w3.org/TR/NOTE-xml-schema-req (*XML schema requirements*)

- www.w3.org/TR/xml-schema-0
- www.w3.org/TR/xml-schema-1
- www.w3.org/TR/xml-schema-2

Alternativvorschläge:

- www.w3.org/TR/dt4dtd
- www.w3.org/TR/NOTE-dcd
- www.w3.org/TR/NOTE-ddml
- www.brics.dk/DSD/
- www.w3.org/TR/NOTE-SOX/
- www.ascc.net/xml/resource/schematron/schematron.html
- www.w3.org/TR/1998/NOTE-XML-data-0105

Referenzen

XML-Schema (cont'd):

Sekundärliteratur:

- www.w3.org/TR/schema-arch
- www.lindamann.com/xml/XML%20Schemas%20NG%20Guide%20HTML.htm
- xml.com/pub/2000/02/23/xmldeviant/index.html?wwwrrr_20000223.txt
- www.w3.org/TR/NOTE-xml-schema-req
- www.iso.ch/cate/d19346.html *(ISO 11404)*

Werkzeuge:

- www.alphaworks.ibm.com/formula/xml
- xml.apache.org (*Xerces Parser*)

  Schema Editoren:

  - www.extensibility.com (*XML Authority*)
  - www.xmlspy.com

Referenzen

Namespace trouble:

- http://lists.xml.org/archives/xml-dev/200005/msg00329.html
- http://www.xml.com/pub/2000/05/17/deviant.html
- http://www.xmlhack.com/read.php?item=758
- http://lists.w3.org/Archives/Public/xml-uri/2000Sep/0083.html

jeckle.de - Netscape

File Edit View Go Communicator Help

Bookmarks Go to: http://www.jeckle.de

Mario Jeckle...
Dialog...
über diese Seiten...
suchen...

Unified Modeling Language (UML)
eXtensible Markup Language (XML)
XML Metadata Interchange (XMI)

Vorträge und Publikationen
Vorlesungen
GOOAL.net
XML-Arbeitskreis

Internet Search Engines
Mersennesche Primzahlen
Feedback
Rotkreuz Mitgliederverwaltung

jeckle.de

DAIMLERCHRYSLER

Tutorial: XML in der Praxis

Appendix -- Hintergrundinformation

Inhaltsübersicht

Namespace trouble

I	Mail von *Pope32767*, die Namespaceproblem benennt
II-IV	Antwort von Tim Berners-Lee
V-IX	XML Schema-Darstellung des Beispiels
X-XV	Mit XMI Generierte DTD

Namespace trouble (I)

Subject: W3C XML "Coordination" Hassle

From: "Pope 32767" <pope32767@hotmail.com>
To: xml-dev@xml.org
Date: Sat, 13 May 2000 01:54:41 GMT

The mind virus that has infected the W3C's XML Activity
is all about what namespaces mean when they are not regular
URLs. The Namespaces Reccommendation said that two namespaces
were the same if they matched exactly char-by-char. (The
attribute values not the prefixes, that is). It
also said that they were URI references. Those two ideas
conflict because thesame-looking relative URL means different things
depending on what document it's in, but strings are strings
no matter what the contextis.

That leads to 3 ideas: forbid relative URLs in namespace
declarations, always convert relative to absolute before
comparing, or just say that namespace names aren't URLs at
all but just strings that look like them (keep the exact-match
idea). All these ideas either break
existing documents or existing software or both and they
are incompatible with each other.

Different groups in the W3C have decided different things
and they are changing their positions all the time and
shifting back and forth. The XML Plenary is supposed to
clean up messes like this. However there is no consensus
anywhere with maybe half the people who take any one
position finding all the other positions absolutely intolerable.
I am not going to name names here.

TimBL who is SUPPOSED to be the final authority has said that
he is not going to decide so that he can make his input just
like any other W3C member. That means everything is stuck
and nobody can say what is going to happen. In my opinion
this SUCKS BIG TIME.

I am not going to say inflamatory thing like the process is
hosed or whatever since I am sure a way out will be found
but lots of people are going to be unhappy with it no matter
what it is. But eventually I think people will get tired
of arguing about it and just accept some answer just to have
a definite answer after all.

I am posting this because I don't think it should all be
kept undercover, instead outside groups like XML-Dev need
to use some pressure where ever they can (over a beer or
whatever) to get the mess cleaned up and the Working Drafts
moved on. I realize this means muck raking journalists
may get a hold of it and run scare headlines like "XML Doomed!"
or somethign stupid like that, I can't help it. Somehow
XML Activity has to **** or get off the pot.

-- Pope 32767

Namespace trouble (II)

Re: W3C XML "Coordination" Hassle

From: Tim Berners-Lee (timbl@w3.org)
Date: Mon, May 15 2000

Message-ID: <011d01bfbe79$0d860370$a60a1712@col.w3.org>
From: "Tim Berners-Lee" <timbl@w3.org>
To: " \"Pope 32767 \"" <pope32767@hotmail.com>
Cc: <tbl@w3.org>, <janet@w3.org>, "Ann Navarro" <ann@webgeek.com>, <xml-dev@xml.org>, <xml-uri@w3.org>
Date: Mon, 15 May 2000 10:22:50 -0400
Subject: Re: W3C XML "Coordination" Hassle

There is a publicly archived list, xml-uri@w3.org, for discussion of this
topic.
See http://lists.w3.org/Archives/Public/xml-uri/
I have put a point of view in a welcome message,
http://lists.w3.org/Archives/Public/xml-uri/2000May/0000.html

See also comments below.
Tim

-----Original Message-----
From: Ann Navarro <ann@webgeek.com>
To: tbl@w3.org <tbl@w3.org>; janet@w3.org <janet@w3.org>
Date: Sunday, May 14, 2000 10:02 AM
Subject: Fwd: W3C XML "Coordination" Hassle

>>X-Originating-IP: [32.100.253.120]
>>From: "Pope 32767" <pope32767@hotmail.com>
>>To: xml-dev@xml.org
>>Subject: W3C XML "Coordination" Hassle
>>Date: Sat, 13 May 2000 01:54:41 GMT
>>Sender: owner-xml-dev@xml.org
>>
>>The mind virus that has infected the W3C's XML Activity
>>is all about what namespaces mean when they are not regular
>>URLs. The Namespaces Reccommendation said that two namespaces
>>were the same if they matched exactly char-by-char. (The
>>attribute values not the prefixes, that is). It
>>also said that they were URI references. Those two ideas
>>conflict because thesame-looking relative URL means different things
>>depending on what document it's in, but strings are strings
>>no matter what the contextis.

Precicely.

Namespace trouble (III)

>>That leads to 3 ideas: forbid relative URLs in namespace
>>declarations, always convert relative to absolute before
>>comparing, or just say that namespace names aren't URLs at
>>all but just strings that look like them (keep the exact-match
>>idea). All these ideas either break
>>existing documents or existing software or both and they
>>are incompatible with each other.

That just about sums it up.

>>Different groups in the W3C have decided different things
>>and they are changing their positions all the time and
>>shifting back and forth. The XML Plenary is supposed to
>>clean up messes like this.

While is anyone can clean something up it is in general to be welcomed, the XML-plenary is an ad-hoc group which covers all the working groups in the
XML for general discussion but is not part of the W3C process for such problems.

>> However there is no consensus
>>anywhere with maybe half the people who take any one
>>position finding all the other positions absolutely intolerable.
>>I am not going to name names here.
>>
>>TimBL who is SUPPOSED to be the final authority has said that
>>he is not going to decide so that he can make his input just
>>like any other W3C member.

I am supposed to give architectural guidance and be a judge of consensus. Here there seems as you say to be no consensus. However, in this case an arbitrary decision
is not, IMHO, a good idea. I feel quite strongly about the technical issues.
Levaing the consensus judgement to others leves me fee to argue the technical points.
(Using URIs is crucailly important for the architecture, and forbidding relative URIs
while workable is an inferior solution).

>> That means everything is stuck
>>and nobody can say what is going to happen. In my opinion
>>this SUCKS BIG TIME.

Namespace trouble (IV)

No one likes to be held up like this over a fundamental point, but making the
decision arbitrary to save time is not a solution IMHO. By saying the lack of decision
is a problem you speak for everyone.

>>I am not going to say inflamatory thing like the process is
>>hosed or whatever since I am sure a way out will be found
>>but lots of people are going to be unhappy with it no matter
>>what it is.

I have a (perhaps naif) trust in the outcome of a technically sound solution and
a relief of misunderstandings. Maybe my naivete will be tested
at this point.

>> But eventually I think people will get tired
>>of arguing about it and just accept some answer just to have
>>a definite answer after all.

I hope that does not become the way we as a community make decisions!

>>I am posting this because I don't think it should all be
>>kept undercover, instead outside groups like XML-Dev need
>>to use some pressure where ever they can (over a beer or
>>whatever) to get the mess cleaned up and the Working Drafts
>>moved on.

Given that a lot of the URI work has happened on IETF lists the
discussion has to be open to IETF input too.

>>I realize this means muck raking journalists
>>may get a hold of it and run scare headlines like "XML Doomed!"
>>or somethign stupid like that, I can't help it. Somehow
>>XML Activity has to **** or get off the pot.

There are plenty of open discussion lists
http://lists.w3.org/Archives/Public
already and although this is an important discussion I don't think it would
make above the fold.

Tim

XML-Schema – Beispiel (V)

```
<?xml version ="1.0"?>
<schema xmlns = "http://www.w3.org/2000/10/XMLSchema">
<element name = "Projektverwaltung">
    <complexType mixed = "false">
      <sequence>
        <element ref = "Projekt" minOccurs = "1" maxOccurs = "unbounded"/>
        <element ref = "Person" minOccurs = "1" maxOccurs = "unbounded"/>
        <element ref = "Projektmitarbeit"/>
      </sequence>
    </complexType>
  </element>
  <element name = "Projekt">
    <complexType mixed = "false">
      <sequence>
        <element ref = "Projektleiter"/>
      </sequence>
      <attribute name = "ID" use = "required" type = "ProjektNo"/>
      <attribute name = "start" type = "date"/>
      <attribute name = "budget" use = "fixed" value = "10000" type = "budgetMoney"/>
      <key name=“ProjektKey”><selector>Projekt><field>@ID</field><key>
    </complexType>
  </element>
```

XML-Schema – Beispiel (VI)

```
<simpleType name = "ProjektNo">
  <extension base="string"/>
  <pattern value = "P-\(19|20)\Nd{2,}-\Lu+\Nd{3,5}"/>
</simpleType>

<simpleType name = "budgetMoney">
  <extension base="float"/>
  <minExclusive value = "0"/>
  <maxInclusive value = "50000"/>
  <scale value = "2"/>
</simpleType>
```

```
<simpleType name = "GehaltsgruppeType">
  <extension base="string"/>
  <enumeration value = "1"/>
  <enumeration value = "1a"/>
  <enumeration value = "2"/>
</simpleType>

<simpleType name = "PersIDType">
  <extension base = "string"/>
  <pattern value = "h-\Nd{2,}-\Nd{5,}"/>
</simpleType>
```

XML-Schema – Beispiel (VII)

```
<element name = "Person">
  <complexType mixed = "false">
    <sequence>
      <element ref = "PersonType"/>
    </sequence>
  </complexType>
</element>

<element name = "PersonType">
  <complexType mixed = "false">
    <sequence>
      <element ref = "Vorname" minOccurs = "1" maxOccurs = "unbounded"/>
      <element ref = "Nachname" minOccurs = "1" maxOccurs = "unbounded"/>
    </sequence>
    <attribute name = "eingeordnet_in" use = "fixed" value = "1a" type = "GehaltsgruppeType"/>
    <attribute name = "PersonalD" type = "PersIDType"/>
    <key name="PersonKey">
      <selector>Person</selector>
      <field>@PersonalID</field>
    </key>
  </complexType>
</element>
```

XML-Schema – Beispiel (VIII)

```
<element name = "Vorname" type = "string"/>
<element name = "Nachname" type = "string"/>

<element name = "Projektmitarbeit">
 <complexType mixed = "false">
     <sequence>
       <element ref = "ausgefuehrtes_Projekt" minOccurs = "1" maxOccurs = "3">
         <keyref refer="ProjektKey"/>
       </element>
       <element ref = "Projektmitarbeiter" minOccurs = "1" maxOccurs = "30">
         <keyref refer="PersonKey"/>
       </element>
     </sequence>
   </complexType>
 </element>

 <element name = "Projektleiter">
   <complexType mixed = "false">
     <sequence>
       <element ref = "PersonType"/>
     </sequence>
   </complexType>
 </element>
```

XML-Schema – Beispiel (IX)

```
  <element name = "Projektmitarbeiter">
    <complexType mixed = "false">
      <sequence>
        <element ref = "PersonType"/>
      </sequence>
    </complexType>
  </element>
</schema>
```

SimpleTree.dtd (X)

```
<!-- XMI Automatic DTD Generation      ->
<!-- Date: Sat May 13 17:34:26 GMT+02:00 2000 ->
<!-- Metamodel: SimpleTree            ->

<!-- ____________________________________________________________ ->
<!-                                                  ->
<!-- XMI is the top-level XML element for XMI transfer text        -->
<!-- ____________________________________________________________ ->

<!ELEMENT XMI (XMI.header, XMI.content?, XMI.difference*,
            XMI.extensions*) >
<!ATTLIST XMI
          xmi.version CDATA #FIXED "1.0"
          timestamp CDATA #IMPLIED
          verified (true | false) #IMPLIED
>

<!-- ____________________________________________________________ ->
<!-                                                  ->
<!-- XMI.header contains documentation and identifies the model,    -->
<!- metamodel, and metametamodel                    ->
<!-- ____________________________________________________________ ->

<!ELEMENT XMI.header (XMI.documentation?, XMI.model*, XMI.metamodel*,
                  XMI.metametamodel*) >

<!-- ____________________________________________________________ ->
<!-                                                  ->
<!- documentation for transfer data                  -->
<!-- ____________________________________________________________ ->

<!ELEMENT XMI.documentation (#PCDATA | XMI.owner | XMI.contact |
                  XMI.longDescription | XMI.shortDescription |
                  XMI.exporter | XMI.exporterVersion |
                  XMI.notice)* >

<!ELEMENT XMI.owner ANY >

<!ELEMENT XMI.contact ANY >

<!ELEMENT XMI.longDescription ANY >

<!ELEMENT XMI.shortDescription ANY >

<!ELEMENT XMI.exporter ANY >

<!ELEMENT XMI.exporterVersion ANY >

<!ELEMENT XMI.exporterID ANY >

<!ELEMENT XMI.notice ANY >
```

SimpleTree.dtd (XI)

```
<!-- ____________________________________________________ ->
<!-                                                      ->
<!- XMI.element.att defines the attributes that each XML element   ->
<!- that corresponds to a metamodel class must have to conform to  ->
<!- the XMI specification.                                         ->
<!- ____________________________________________________ ->

<!ENTITY % XMI.element.att
        'xmi.id ID #IMPLIED xmi.label CDATA #IMPLIED xmi.uuid
         CDATA #IMPLIED ' >

<!- ____________________________________________________ ->
<!-                                                      ->
<!- XMI.link.att defines the attributes that each XML element that  ->
<!- corresponds to a metamodel class must have to enable it to      ->
<!- function as a simple XLink as well as refer to model            ->
<!- constructs within the same XMI file.                            ->
<!- ____________________________________________________ ->

<!ENTITY % XMI.link.att
        'xml:link CDATA #IMPLIED inline (true | false) #IMPLIED
         actuate (show | user) #IMPLIED href CDATA #IMPLIED role
         CDATA #IMPLIED title CDATA #IMPLIED show (embed | replace
         | new) #IMPLIED behavior CDATA #IMPLIED xmi.idref IDREF
         #IMPLIED xmi.uuidref CDATA #IMPLIED' >

<!- ____________________________________________________ ->
<!-                                                      ->
<!- XMI.model identifies the model(s) being transferred            ->
<!- ____________________________________________________ ->

<!ELEMENT XMI.model ANY >
<!ATTLIST XMI.model
        %XMI.link.att;
        xmi.name     CDATA #REQUIRED
        xmi.version  CDATA #IMPLIED>

<!- ____________________________________________________ ->
<!-                                                      ->
<!- XMI.metamodel identifies the metamodel(s) for the transferred   ->
<!- data                                                           ->
<!- ____________________________________________________ ->

<!ELEMENT XMI.metamodel ANY >
<!ATTLIST XMI.metamodel
        %XMI.link.att;
        xmi.name     CDATA #REQUIRED
        xmi.version  CDATA #IMPLIED>

<!- ____________________________________________________ ->
<!-                                                      ->
<!- XMI.metametamodel identifies the metametamodel(s) for the       ->
<!- transferred data                                               ->
<!- ____________________________________________________ ->

<!ELEMENT XMI.metametamodel ANY >
<!ATTLIST XMI.metametamodel
        %XMI.link.att;
        xmi.name     CDATA #REQUIRED
        xmi.version  CDATA #IMPLIED>
```

SimpleTree.dtd (XII)

```
<!-- ____________________________________________________ ->
<!-                                                      ->
<!- XMI.content is the actual data being transferred     ->
<!- ____________________________________________________ ->

<!ELEMENT XMI.content ANY >

<!- ____________________________________________________ ->
<!-                                                      ->
<!- XMI.extensions contains data to transfer that does not conform  ->
<!- to the metamodel(s) in the header                   ->
<!- ____________________________________________________ ->

<!ELEMENT XMI.extensions ANY >
<!ATTLIST XMI.extensions
        xmi.extender CDATA #REQUIRED>

<!- ____________________________________________________ ->
<!-                                                      ->
<!- extension contains information related to a specific model     ->
<!- construct that is not defined in the metamodel(s) in the header ->
<!- ____________________________________________________ ->

<!ELEMENT XMI.extension ANY >
<!ATTLIST XMI.extension
        %XMI.element.att;
        %XMI.link.att;
        xmi.extender   CDATA #REQUIRED
        xmi.extenderID CDATA #REQUIRED>

<!- ____________________________________________________ ->
<!-                                                      ->
<!- XMI.difference holds XML elements representing differences to a ->
<!- base model                                           ->
<!- ____________________________________________________ ->

<!ELEMENT XMI.difference (XMI.difference | XMI.delete | XMI.add |
                XMI.replace)* >
<!ATTLIST XMI.difference
        %XMI.element.att;
        %XMI.link.att;>

<!- ____________________________________________________ ->
<!-                                                      ->
<!- XMI.delete represents a deletion from a base model   ->
<!- ____________________________________________________ ->

<!ELEMENT XMI.delete EMPTY >
<!ATTLIST XMI.delete
        %XMI.element.att;
        %XMI.link.att;>

<!- ____________________________________________________ ->
<!-                                                      ->
<!- XMI.add represents an addition to a base model       ->
<!- ____________________________________________________ ->

<!ELEMENT XMI.add ANY >
<!ATTLIST XMI.add
        %XMI.element.att;
        %XMI.link.att;
        xmi.position CDATA "-1">
```

SimpleTree.dtd (XIII)

```
<!-- ____________________________________________________ ->
<!-                                                      ->
<!-- XMI.replace represents the replacement of a model construct    ->
<!-- with another model construct in a base model                   ->
<!-- ____________________________________________________ ->

<!ELEMENT XMI.replace ANY >
<!ATTLIST XMI.replace
        %XMI.element.att;
        %XMI.link.att;
        xmi.position CDATA "-1"
>

<!-- ____________________________________________________ ->
<!-                                                      ->
<!-- XMI.reference may be used to refer to data types not defined in ->
<!-- the metamodel                                        ->
<!-- ____________________________________________________ ->

<!ELEMENT XMI.reference ANY >
<!ATTLIST XMI.reference
        %XMI.link.att;
>
```

```
<!-- ____________________________________________________ ->
<!-                                                      ->
<!-- This section contains the declaration of XML elements          ->
<!-- representing data types                                        ->
<!-- ____________________________________________________ ->

<!ELEMENT XMI.TypeDefinitions ANY >
<!ELEMENT XMI.field ANY >
<!ELEMENT XMI.seqItem ANY >
<!ELEMENT XMI.octetStream (#PCDATA) >
<!ELEMENT XMI.unionDiscrim ANY >
<!ELEMENT XMI.enum EMPTY >
<!ATTLIST XMI.enum
        xmi.value CDATA #REQUIRED>

<!ELEMENT XMI.any ANY >
<!ATTLIST XMI.any
        %XMI.link.att;
        xmi.type CDATA #IMPLIED
        xmi.name CDATA #IMPLIED>

<!ELEMENT XMI.CorbaTypeCode (XMI.CorbaTcAlias | XMI.CorbaTcStruct |
                XMI.CorbaTcSequence | XMI.CorbaTcArray |
                XMI.CorbaTcEnum | XMI.CorbaTcUnion |
                XMI.CorbaTcExcept | XMI.CorbaTcString |
                XMI.CorbaTcWstring | XMI.CorbaTcShort |
                XMI.CorbaTcLong | XMI.CorbaTcUshort |
                XMI.CorbaTcUlong | XMI.CorbaTcFloat |
                XMI.CorbaTcDouble | XMI.CorbaTcBoolean |
                XMI.CorbaTcChar | XMI.CorbaTcWchar |
                XMI.CorbaTcOctet | XMI.CorbaTcAny |
                XMI.CorbaTcTypeCode | XMI.CorbaTcPrincipal |
                XMI.CorbaTcNull | XMI.CorbaTcVoid |
                XMI.CorbaTcLongLong |
                XMI.CorbaTcLongDouble) >
<!ATTLIST XMI.CorbaTypeCode
        %XMI.element.att;>
```

SimpleTree.dtd (XIV)

```
<!ELEMENT XMI.CorbaTcAlias (XMI.CorbaTypeCode) >
<!ATTLIST XMI.CorbaTcAlias
        xmi.tcName CDATA #REQUIRED
        xmi.tcId   CDATA #IMPLIED>

<!ELEMENT XMI.CorbaTcStruct (XMI.CorbaTcField)* >
<!ATTLIST XMI.CorbaTcStruct
        xmi.tcName CDATA #REQUIRED
        xmi.tcId   CDATA #IMPLIED>

<!ELEMENT XMI.CorbaTcField (XMI.CorbaTypeCode) >
<!ATTLIST XMI.CorbaTcField
        xmi.tcName CDATA #REQUIRED>

<!ELEMENT XMI.CorbaTcSequence (XMI.CorbaTypeCode |
                    XMI.CorbaRecursiveType) >
<!ATTLIST XMI.CorbaTcSequence
        xmi.tcLength CDATA #REQUIRED>

<!ELEMENT XMI.CorbaRecursiveType EMPTY >
<!ATTLIST XMI.CorbaRecursiveType
        xmi.offset CDATA #REQUIRED>

<!ELEMENT XMI.CorbaTcArray (XMI.CorbaTypeCode) >
<!ATTLIST XMI.CorbaTcArray
        xmi.tcLength CDATA #REQUIRED>

<!ELEMENT XMI.CorbaTcObjRef EMPTY >
<!ATTLIST XMI.CorbaTcObjRef
        xmi.tcName CDATA #REQUIRED
        xmi.tcId   CDATA #IMPLIED>

<!ELEMENT XMI.CorbaTcEnum (XMI.CorbaTcEnumLabel) >
<!ATTLIST XMI.CorbaTcEnum
        xmi.tcName CDATA #REQUIRED
        xmi.tcId   CDATA #IMPLIED>

<!ELEMENT XMI.CorbaTcEnumLabel EMPTY >
<!ATTLIST XMI.CorbaTcEnumLabel
        xmi.tcName CDATA #REQUIRED>

<!ELEMENT XMI.CorbaTcUnionMbr (XMI.CorbaTypeCode, XMI.any) >
<!ATTLIST XMI.CorbaTcUnionMbr
        xmi.tcName CDATA #REQUIRED>

<!ELEMENT XMI.CorbaTcUnion (XMI.CorbaTypeCode, XMI.CorbaTcUnionMbr*) >
<!ATTLIST XMI.CorbaTcUnion
        xmi.tcName CDATA #REQUIRED
        xmi.tcId   CDATA #IMPLIED>

<!ELEMENT XMI.CorbaTcExcept (XMI.CorbaTcField)* >
<!ATTLIST XMI.CorbaTcExcept
        xmi.tcName CDATA #REQUIRED
        xmi.tcId   CDATA #IMPLIED>

<!ELEMENT XMI.CorbaTcString EMPTY >
<!ATTLIST XMI.CorbaTcString
        xmi.tcLength CDATA #REQUIRED>

<!ELEMENT XMI.CorbaTcWstring EMPTY >
<!ATTLIST XMI.CorbaTcWstring
        xmi.tcLength CDATA #REQUIRED>
```

SimpleTree.dtd (XV)

```
<!ELEMENT XMI.CorbaTcFixed EMPTY >
<!ATTLIST XMI.CorbaTcFixed
        xmi.tcDigits CDATA #REQUIRED
        xmi.tcScale  CDATA #REQUIRED>

<!ELEMENT XMI.CorbaTcShort EMPTY >
<!ELEMENT XMI.CorbaTcLong EMPTY >
<!ELEMENT XMI.CorbaTcUshort EMPTY >
<!ELEMENT XMI.CorbaTcUlong EMPTY >
<!ELEMENT XMI.CorbaTcFloat EMPTY >
<!ELEMENT XMI.CorbaTcDouble EMPTY >
<!ELEMENT XMI.CorbaTcBoolean EMPTY >
<!ELEMENT XMI.CorbaTcChar EMPTY >
<!ELEMENT XMI.CorbaTcWchar EMPTY >
<!ELEMENT XMI.CorbaTcOctet EMPTY >
<!ELEMENT XMI.CorbaTcAny EMPTY >
<!ELEMENT XMI.CorbaTcTypeCode EMPTY >
<!ELEMENT XMI.CorbaTcPrincipal EMPTY >
<!ELEMENT XMI.CorbaTcNull EMPTY >
<!ELEMENT XMI.CorbaTcVoid  EMPTY >
<!ELEMENT XMI.CorbaTcLongLong EMPTY >
<!ELEMENT XMI.CorbaTcLongDouble EMPTY >
```

```
<!-- ____________________________________________________ -->
<!-- ____________________________________________________ -->
<!-- METAMODEL:  SimpleTree                               -->
<!-- ____________________________________________________ -->

<!-- ____________________________________________________ -->
<!-- ____________________________________________________ -->
<!-- METAMODEL CLASS:  Node                               -->
<!-- ____________________________________________________ -->

<!ELEMENT Node.name (#PCDATA | XMI.reference)* >

<!ELEMENT Node (Node.name?, XMI.extension*)? >
<!ATTLIST Node
        %XMI.element.att;
        %XMI.link.att;>

<!-- ____________________________________________________ -->
<!-- ____________________________________________________ -->
<!-- METAMODEL CLASS:  innerNode                          -->
<!-- ____________________________________________________ -->

<!ELEMENT  innerNode.node (Node | innerNode | leafNode)* >

<!ELEMENT  innerNode (Node.name?, XMI.extension*, innerNode.node*)? >
<!ATTLIST  innerNode
        %XMI.element.att;
        %XMI.link.att;>
```

SimpleTree.dtd (XVI)

```
<!-- ____________________________________________________________ ->
<!-                                                              ->
<!- METAMODEL CLASS:  leafNode                                   ->
<!-- ____________________________________________________________ ->


<!ELEMENT  leafNode.information (#PCDATA | XMI.reference)* >

<!ELEMENT  leafNode (Node.name?, leafNode.information?,
                 XMI.extension*)? >
<!ATTLIST  leafNode
          %XMI.element.att;
          %XMI.link.att;
>

<!ELEMENT  SimpleTree ((Node |  innerNode | leafNode)*) >
<!ATTLIST  SimpleTree
          %XMI.element.att;
          %XMI.link.att;
>
```