XML
and Content Management

Lecture 3: Modelling XML Documents: XML Schema

Maciej Ogrodniczuk, Patryk Czarnik

MIMUW, Oct 18, 2010

DTD example (recall)

```
<!ELEMENT glossary (entry)+>
```

DTD example (recall)

```
<!ELEMENT glossary (entry)+>

<!ELEMENT entry (term, description)>
```

DTD example (recall)

```
<!ELEMENT glossary (entry)+>

<!ELEMENT entry (term, description)>
<!ATTLIST entry
          id ID #REQUIRED
          lastMod NMTOKEN #IMPLIED>
```

DTD example (recall)

```
<!ELEMENT glossary (entry)+>

<!ELEMENT entry (term, description)>
<!ATTLIST entry
          id ID #REQUIRED
          lastMod NMTOKEN #IMPLIED>

<!ELEMENT term (#PCDATA)>
```

DTD example (recall)

```
<!ELEMENT glossary (entry)+>

<!ELEMENT entry (term, description)>
<!ATTLIST entry
          id ID #REQUIRED
          lastMod NMTOKEN #IMPLIED>

<!ELEMENT term (#PCDATA)>

<!ELEMENT description (#PCDATA | link)*>
```

DTD example (recall)

```
<!ELEMENT glossary (entry)+>

<!ELEMENT entry (term, description)>
<!ATTLIST entry
          id ID #REQUIRED
          lastMod NMTOKEN #IMPLIED>

<!ELEMENT term (#PCDATA)>

<!ELEMENT description (#PCDATA | link)*>

<!ELEMENT link (#PCDATA)>
```

DTD example (recall)

```
<!ELEMENT glossary (entry)+>

<!ELEMENT entry (term, description)>
<!ATTLIST entry
          id ID #REQUIRED
          lastMod NMTOKEN #IMPLIED>

<!ELEMENT term (#PCDATA)>

<!ELEMENT description (#PCDATA | link)*>

<!ELEMENT link (#PCDATA)>
<!ATTLIST link
          term IDREF #REQUIRED>
```

DTD drawbacks

- Limited control over document structure

DTD drawbacks

- Limited control over document structure
- Too general data types: not possible to limit text content to numbers, dates etc.

DTD drawbacks

- Limited control over document structure
- Too general data types: not possible to limit text content to numbers, dates etc.
- Poor methods of controlling number of elements occurrence:

```
<!ELEMENT stanza (line, line, line, line,
                  line?, line?, line?, line?)>
```

DTD drawbacks

- Limited control over document structure
- Too general data types: not possible to limit text content to numbers, dates etc.
- Poor methods of controlling number of elements occurrence:

```
<!ELEMENT stanza (line, line, line, line,
                  line?, line?, line?, line?)>
```

- Little abilities to modularly relate similar definitions. The existing ones (mainly parametric entities) operates on text level, not logical level, hence they have many limitations.

DTD drawbacks

- Limited control over document structure
- Too general data types: not possible to limit text content to numbers, dates etc.
- Poor methods of controlling number of elements occurrence:

```
<!ELEMENT stanza (line, line, line, line,
                  line?, line?, line?, line?)>
```

- Little abilities to modularly relate similar definitions. The existing ones (mainly parametric entities) operates on text level, not logical level, hence they have many limitations.
- Non-XML syntax (SGML roots)

The need of strict text content control

Modern applications of XML

- Electronic data interchange, WebServices, ...
- Databases rather than text documents
- Format (and even values) of text fields as a part of document structure

The need of strict text content control

Modern applications of XML

- Electronic data interchange, WebServices, ...
- Databases rather than text documents
- Format (and even values) of text fields as a part of document structure

Example

```
<order>
  <products-number>Not too few and not too
many</products-number>
</order>
```

The need of strict text content control

Modern applications of XML

- Electronic data interchange, WebServices, ...
- Databases rather than text documents
- Format (and even values) of text fields as a part of document structure

Example

```
<order>
  <products-number>Not too few and not too
many</products-number>
</order>
```

Example

```
<order>
  <products-number>-999</products-number>
</order>
```

XML Schema specification

- 1999: requirements for new standard published by W3C
- 2001: XML Schema becomes a W3C Recommendation, three parts:
  - *XML Schema Part 0: Primer*,
  - *XML Schema Part 1: Structures*,
  - *XML Schema Part 2: Datatypes*.
- 2004: second edition of the recommendation

XML Schema — main structure

- XML Schema extends DTD capabilities (automatic conversion from DTD to XML Schema available)
  - without entities.

XML Schema — main structure

- XML Schema extends DTD capabilities (automatic conversion from DTD to XML Schema available)
  - without entities.
- A schema is an XML document.

XML Schema — main structure

- XML Schema extends DTD capabilities (automatic conversion from DTD to XML Schema available)
  - without entities.
- A schema is an XML document.
- Elements (and other components) relevant to the standard in XML Schema namespace `http://www.w3.org/2001/XMLSchema` (prefix `xsd` used in the rest of lecture).

XML Schema — main structure

- XML Schema extends DTD capabilities (automatic conversion from DTD to XML Schema available)
  - without entities.
- A schema is an XML document.
- Elements (and other components) relevant to the standard in XML Schema namespace `http://www.w3.org/2001/XMLSchema` (prefix `xsd` used in the rest of lecture).
- Root element is `<xsd:schema>`.

XML Schema — main structure

- XML Schema extends DTD capabilities (automatic conversion from DTD to XML Schema available)
  - without entities.
- A schema is an XML document.
- Elements (and other components) relevant to the standard in XML Schema namespace `http://www.w3.org/2001/XMLSchema` (prefix `xsd` used in the rest of lecture).
- Root element is `<xsd:schema>`.
- Root element contains "global definitions", such as `<xsd:element>` or `<xsd:attribute>`.

XML Schema — trivial example

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="glossary">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="entry" minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="entry">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="term"/>
        <xs:element ref="description"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID"/>
      <xs:attribute name="lastMod" type="xs:dateTime"/>
    </xs:complexType>
  </xs:element>


  <xs:element name="term" type="xs:string"/>

  <xs:element name="description" type="xs:string"/>
</xs:schema>
```

Types in XML Schema

- Types assigned to elements and attributes
- Set of predefined types
- Means for defining author's own types

Types in XML Schema

- Types assigned to elements and attributes
- Set of predefined types
- Means for defining author's own types

Assigning types to elements

```
<xsd:element name="entry" type="TEntry"/>

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

<xsd:element name="term" type="xsd:string"/>
```

Types: simple vs complex

Simple types

- Text strings, numbers, date and time, etc.
- Values written as text
- Applied to elements and attributes

Types: simple vs complex

Simple types

- Text strings, numbers, date and time, etc.
- Values written as text
- Applied to elements and attributes

Complex types

- Structures (subelements, attributes)
- Document tree fragments
- Applied to elements only

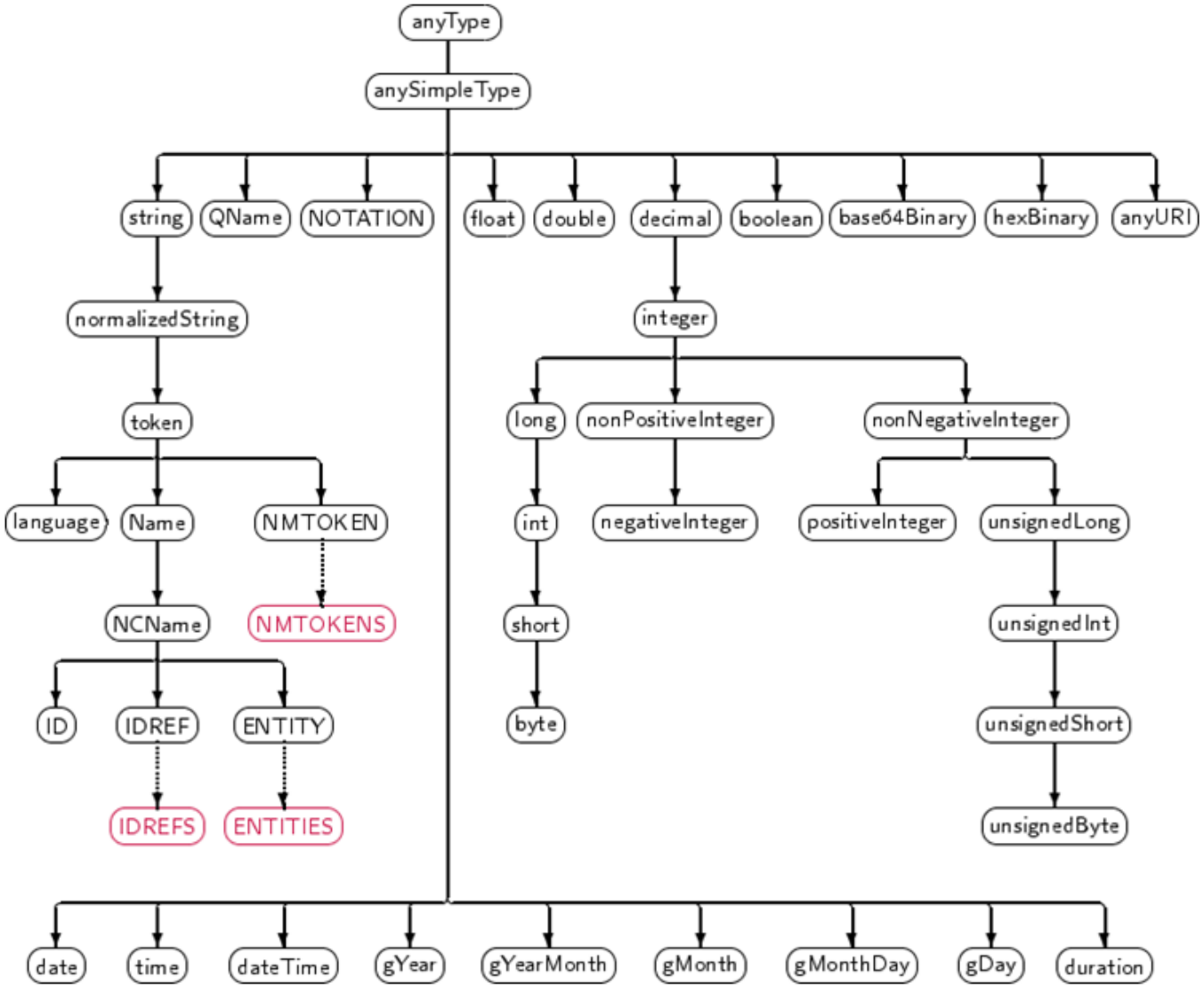
XML Schema predefined types

- `boolean` — logical value (`true`, `false`, `0`, or `1`),
- `string` — characters string,
- `normalizedString` — string to be normalized during processing; every white space character is replaced with space character (`#x20`,)
- `token` — like above; additionally, leading and trailing white spaces are removed,
- `hexBinary`, `base64Binary` — binary data in hexadecimal or base64 notation,

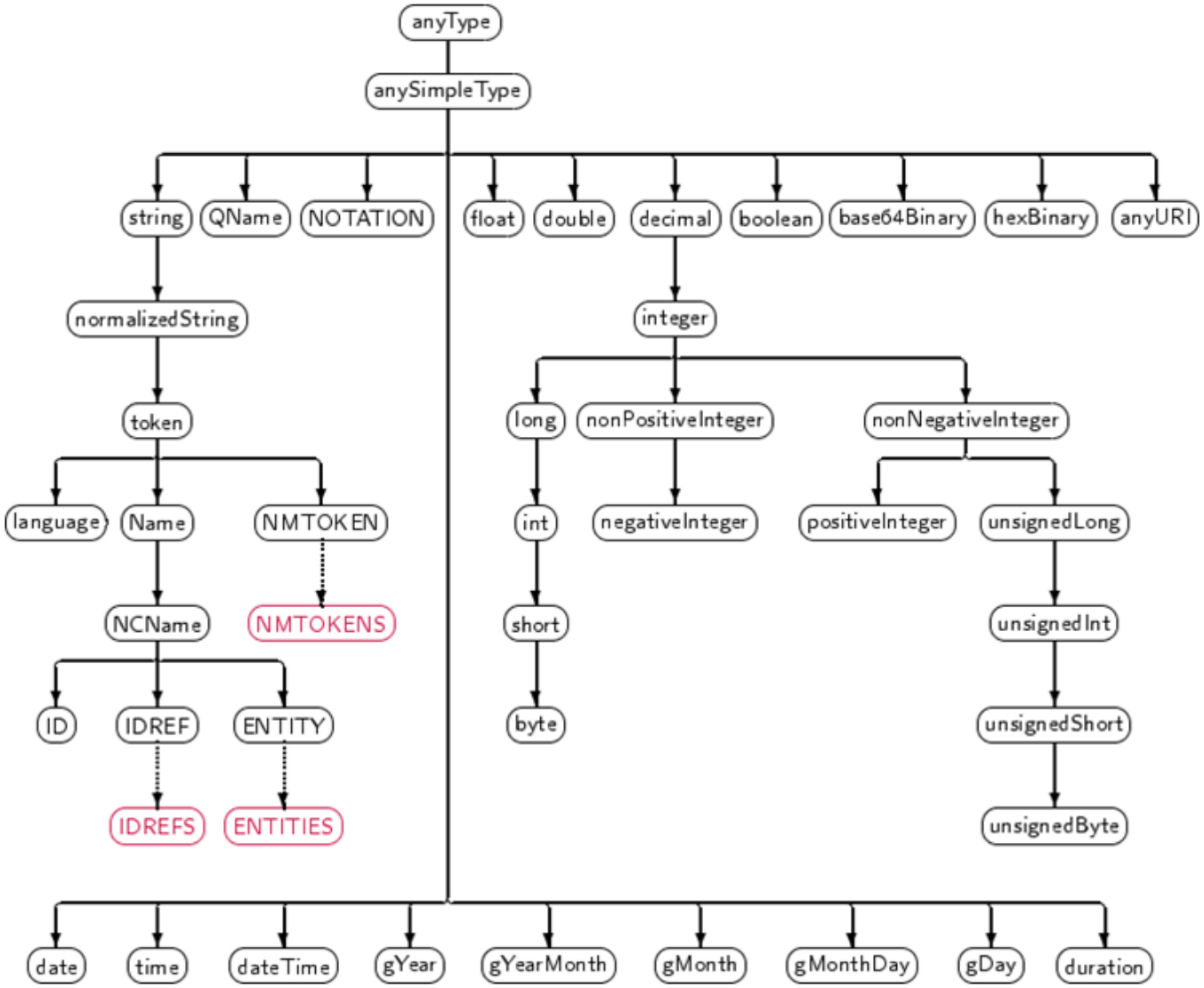
XML Schema predefined types

- `decimal` — rational number (positive or negative) in decimal notation,
- `float` — 32-bit floating point number (including special values: `-INF`, `INF`, and `NaN`),
- `double` — 64-bit floating point number (including `-INF`, `INF`, and `NaN`),
- `integer` — most general type for integer numbers,
- `long`, `int`, `short`, `byte` — integer values from appropriate intervals (like in Java),
- `date`, `time`, `dateTime`, `duration`, `gYearMonth`, ...
- `ID`, `IDREF`, `IDREFS`, `ENTITY`, `ENTITIES`, `NOTATION`, `NMTOKEN`, `NMTOKENS`, `CDATA`, `language`, `Name`, `normalizedString`, `token`, `uriReference`...

Predefined types

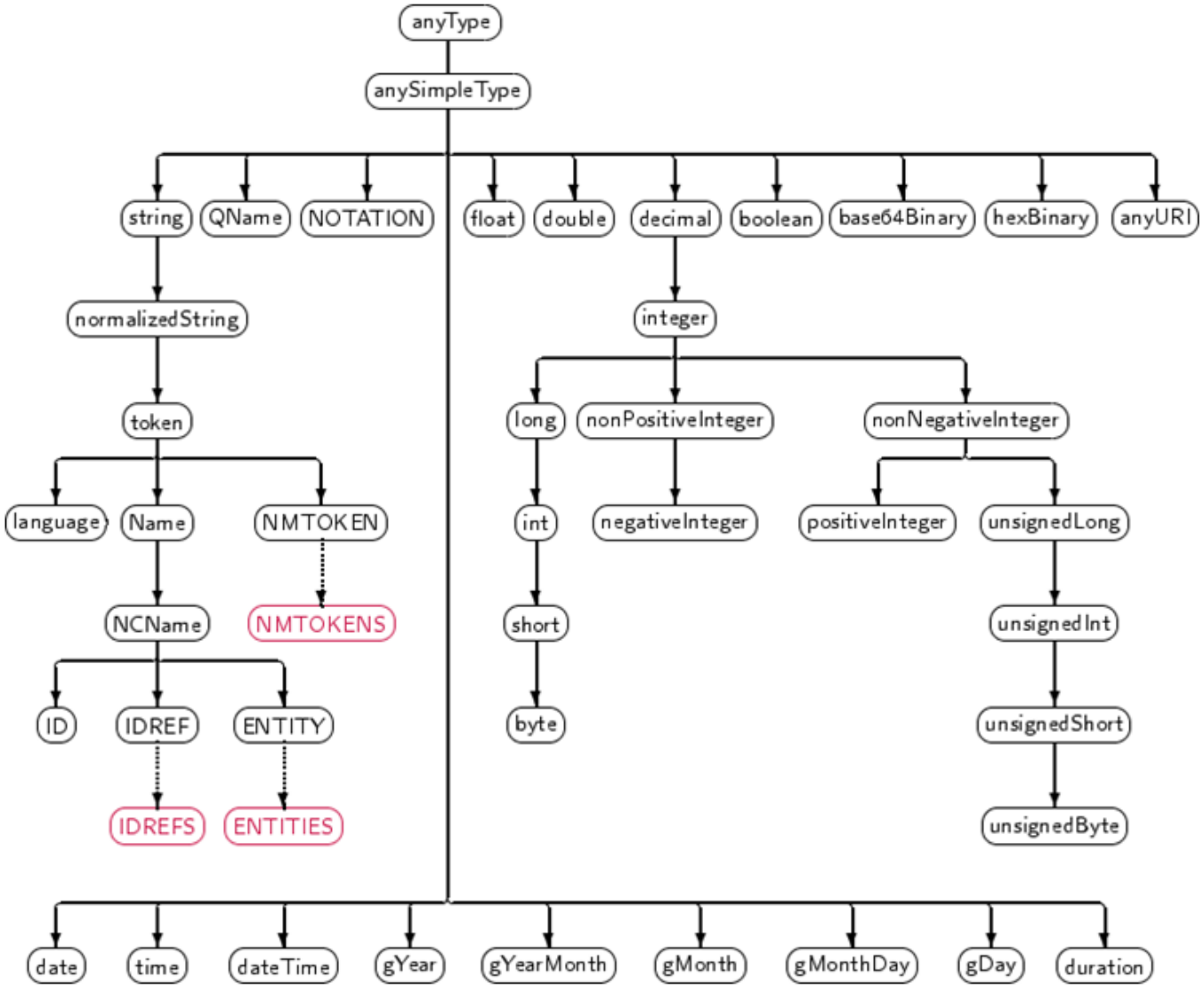


Predefined types



basic types

Predefined types

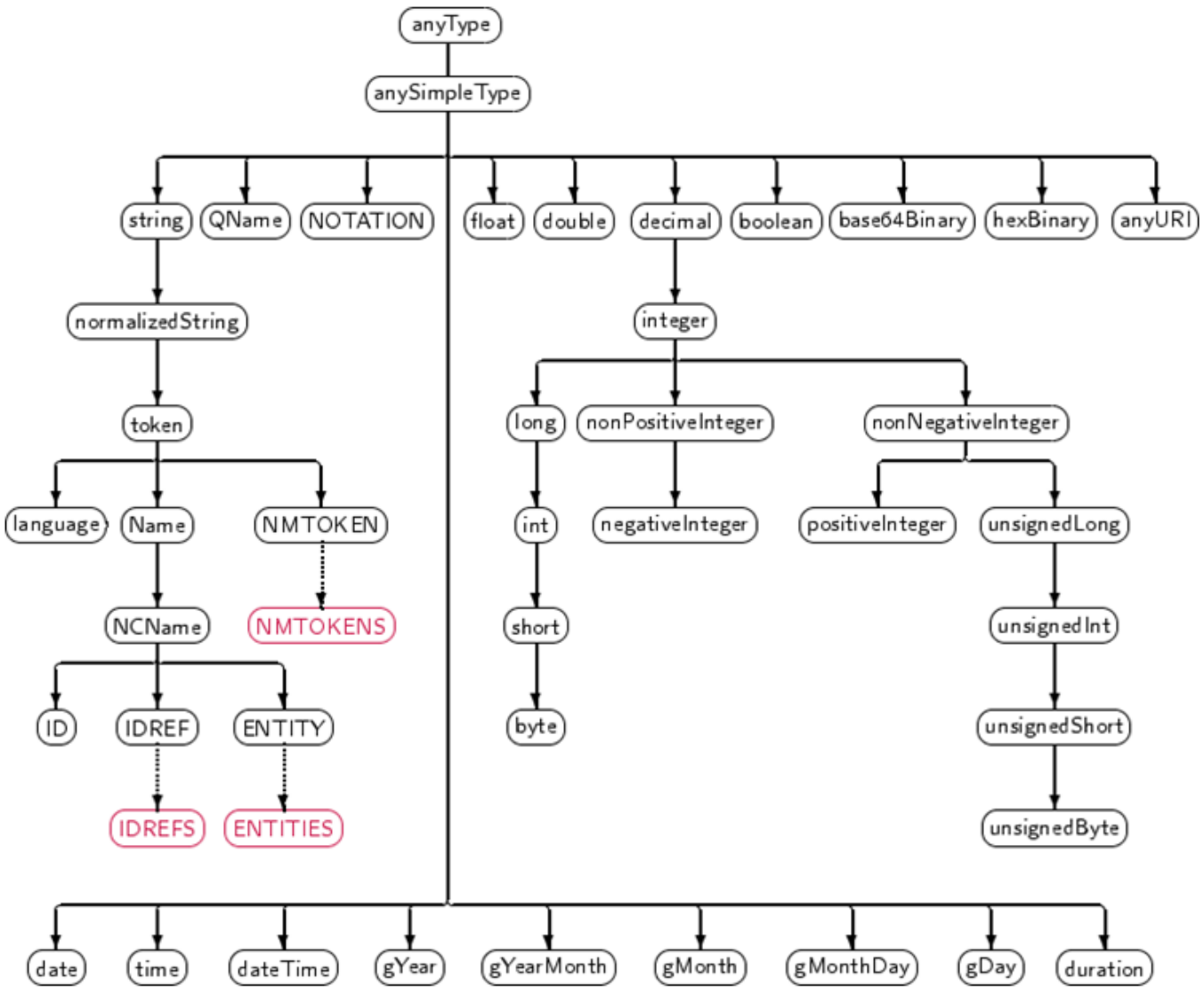


basic types

primitive types

primitive types

Predefined types



basic types

primitive types

derived types

primitive types

Basic types xsd:anyType and xsd:anySimpleType

xsd:anyType

- Can be used as element type:

```
<xsd:element name="codeFragment"
    type="xsd:anyType"/>
```

- Any character and element content (including mixed content) allowed

Basic types xsd:anyType and xsd:anySimpleType

xsd:anyType

- Can be used as element type:

```
<xsd:element name="codeFragment"
    type="xsd:anyType"/>
```

- Any character and element content (including mixed content) allowed
- Default type for elements:
  xsd:anyType is used if an element declaration does not give any type.

Basic types xsd:anyType and xsd:anySimpleType

xsd:anyType

- Can be used as element type:

```
<xsd:element name="codeFragment"
    type="xsd:anyType"/>
```

- Any character and element content (including mixed content) allowed
- Default type for elements:
  xsd:anyType is used if an element declaration does not give any type.

xsd:anySimpleType

- Any character content allowed
- Can be used as element or attribute type
- Default type for attributes

Defining new simple types

One can define new simple types basing on predefined types and using **facets** (pol. *aspekty*).

Defining new simple types

One can define new simple types basing on predefined types and using **facets** (pol. *aspekty*).

Important facets:

- `list` — lists of space-separated values (like `NMTOKENS`),
- `union` — set-theoretic union of several simple types,

Defining new simple types

One can define new simple types basing on predefined types and using **facets** (pol. *aspekty*).

Important facets:

- `list` — lists of space-separated values (like `NMTOKENS`),
- `union` — set-theoretic union of several simple types,
- `minInclusive`, `maxInclusive`, `minExclusive`, `maxExclusive` — narrowing the interval of allowed numeric values,
- `pattern` — restricting set of allowed character strings with regular expressions (`.`, `a?`, `a+`, `a*`, `(a|b)`, `[a-c]`), `(ab){2,}`...)
- `enumeration` — enumerating the set of allowed values,
- `length`, `minLength`, `maxLength` — length of a string; for types defined as list applies to number of elements.

Example: Defining simple type using pattern

- `<xsd:simpleType>` element introduces new simple type.
- `<xsd:restriction>` element defines the type by specifying (or narrowing) "constraining facets" of type given in base attribute.
- `<xsd:pattern>` element specifies `pattern` facet.

Simple type restriction example

```
<xsd:element name="postalCode">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="\d{2}-\d{3}"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

Example: Defining simple type using pattern

- `<xsd:simpleType>` element introduces new simple type.
- `<xsd:restriction>` element defines the type by specifying (or narrowing) "constraining facets" of type given in base attribute.
- `<xsd:pattern>` element specifies `pattern` facet.

Simple type restriction example

```
<xsd:element name="postalCode">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="\d{2}-\d{3}"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

Example: Defining simple type using pattern

- `<xsd:simpleType>` element introduces new simple type.
- `<xsd:restriction>` element defines the type by specifying (or narrowing) "constraining facets" of type given in base attribute.
- `<xsd:pattern>` element specifies pattern facet.

Simple type restriction example

```
<xsd:element name="postalCode">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="\d{2}-\d{3}"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

Types: anonymous vs named

Anonymous type

- Defined in place of use ("inside" element or attribute)
- Used for one element or attribute
- Use it to make the schema more compact (only if a type is to be used in one place)

Types: anonymous vs named

Anonymous type

- Defined in place of use ("inside" element or attribute)
- Used for one element or attribute
- Use it to make the schema more compact (only if a type is to be used in one place)

Named type

- Defined globally (in `<xsd:schema>`)
- May be used for many elements or attributes
- Use it to make the schema more modular (whenever a type may be used is several places)

Types: anonymous vs named

Anonymous type

- Defined in place of use ("inside" element or attribute)
- Used for one element or attribute
- Use it to make the schema more compact (only if a type is to be used in one place)

Named type

- Defined globally (in `<xsd:schema>`)
- May be used for many elements or attributes
- Use it to make the schema more modular (whenever a type may be used is several places)

Named type example

```
<xsd:simpleType name="TPostalCodePL">
  ...
</xsd:simpleType>
<xsd:element name="postalCode" type="TPostalCodePL"/>
```

Constraining facets: intervals

DTD

- Not possible to restrict text content of elements
- Not understand numeric values of attributes

Constraining facets: intervals

DTD

- Not possible to restrict text content of elements
- Not understand numeric values of attributes

XML Schema

```
<xsd:simpleType name="LottoNumber_type">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="1"/>
    <xsd:maxInclusive value="49"/>
  </xsd:restriction>
</xsd:simpleType>
```

Constraining facets: intervals

DTD

- Not possible to restrict text content of elements
- Not understand numeric values of attributes

XML Schema

```
<xsd:simpleType name="LottoNumber_type">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="1"/>
    <xsd:maxInclusive value="49"/>
  </xsd:restriction>
</xsd:simpleType>
```

Constraining facets: enumeration

DTD

- Not possible to restrict text content of elements

Constraining facets: enumeration

DTD

- Not possible to restrict text content of elements

XML Schema

```
<xsd:simpleType name="TSex">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="female"/>
    <xsd:enumeration value="male"/>
  </xsd:restriction>
</xsd:simpleType>
```

Constraining facets: enumeration

DTD

- Not possible to restrict text content of elements

XML Schema

```
<xsd:simpleType name="TSex">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="female"/>
    <xsd:enumeration value="male"/>
  </xsd:restriction>
</xsd:simpleType>
```

Restrictions to restrictions

- Different facets — conjunction
- Multiple entries for the same facet (`pattern` and `enumeration` only) — disjunction

Restrictions to restrictions

- Different facets — conjunction
- Multiple entries for the same facet (`pattern` and `enumeration` only) — disjunction
- A facet in a restriction can not be more general than in base type.

Incorrect restriction

```
<xsd:simpleType name="TByte9">
  <xsd:restriction base="xsd:byte">
    <xsd:minInclusive value="-256"/>
    <xsd:maxInclusive value="255"/>
  </xsd:restriction>
</xsd:simpleType>
```

Lists

- **List** of simple values separated with white space
- **Lists** are simple types, **sequences** are complex types

Lists

- **List** of simple values separated with white space
- **Lists** are simple types, **sequences** are complex types

List further restricted by `length` facet

```
<xsd:simpleType name="TLottoNumberList">
  <xsd:list itemType="LottoNumber_type"/>
</xsd:simpleType>

<xsd:simpleType name="TLotto">
  <xsd:restriction base="TLottoNumberList">
    <xsd:length value="6"/>
  </xsd:restriction>
</xsd:simpleType>
```

Lists

- **List** of simple values separated with white space
- **Lists** are simple types, **sequences** are complex types

List further restricted by `length` facet

```
<xsd:simpleType name="TLottoNumberList">
  <xsd:list itemType="LottoNumber_type"/>
</xsd:simpleType>

<xsd:simpleType name="TLotto">
  <xsd:restriction base="TLottoNumberList">
    <xsd:length value="6"/>
  </xsd:restriction>
</xsd:simpleType>
```

List in document

```
<lotto>3 13 5  15 48 3</lotto>
```

Anonymous base of derivation — lists

One can use anomous simple type as item type of a list.

List defined basing on anonymous simple type

```
<xsd:simpleType name="TLottoNumberList">
  <xsd:list>
    <xsd:simpleType>
      <xsd:restriction base="xsd:integer">
        <xsd:minInclusive value="1"/>
        <xsd:maxInclusive value="49"/>
      </xsd:restriction>
    <xsd:simpleType>
  </xsd:list>
</xsd:simpleType>
```

Anonymous base of derivation — restriction

Anonymous simple type can also be base to extension.

Restricting an anonymous simple type (here a list)

```
<xsd:simpleType name="TLotto">
  <xsd:restriction>
    <xsd:simpleType>
      <xsd:list itemType="LottoNumber_type"/>
    </xsd:simpleType>
    <xsd:length value="6"/>
  </xsd:restriction>
</xsd:simpleType>
```

Anonymous base of derivation — combined

Lotto defined at once, using anonymous simple types

```
<xsd:simpleType name="TLotto">
  <xsd:restriction>
    <xsd:simpleType>
      <xsd:list>
        <xsd:restriction base="xsd:integer">
          <xsd:minInclusive value="1"/>
          <xsd:maxInclusive value="49"/>
        </xsd:restriction>
      </xsd:list>
    </xsd:simpleType>
    <xsd:length value="6"/>
  </xsd:restriction>
</xsd:simpleType>
```

Unions

Union of values sets. A value is a member of union type if and only if it is a member of any of component types.

Union of named simple types

```
<xsd:simpleType name="TNumericSize">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="32"/>
    <xsd:maxInclusive value="62"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="TLetterSize">
  <xsd:restriction base="xsd:token">
    <xsd:enumeration value="S"/>
    <xsd:enumeration value="M"/>
    <xsd:enumeration value="L"/>
    <xsd:enumeration value="XL"/>
    <xsd:enumeration value="XXL"/>
  </xsd:restriction>
</xsd:simpleType>
```

Unions

Union of values sets. A value is a member of union type if and only if it is a member of any of component types.

Union of named simple types

```
<xsd:simpleType name="TNumericSize">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="32"/>
    <xsd:maxInclusive value="62"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="TLetterSize">
  <xsd:restriction base="xsd:token">
    <xsd:enumeration value="S"/>
    <xsd:enumeration value="M"/>
    <xsd:enumeration value="L"/>
    <xsd:enumeration value="XL"/>
    <xsd:enumeration value="XXL"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="TSize">
  <xsd:union memberTypes="TNumericSize TLetterSize"/>
</xsd:simpleType>
```

Unions

Unions can also be defined basing on anonymous types.

Union of anonymous simple types

```
<xsd:simpleType name="TSize">
  <xsd:union>
    <xsd:simpleType>
      <xsd:restriction base="xsd:integer">
        <xsd:minInclusive value="34"/>
        <xsd:maxInclusive value="62"/>
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType>
      <xsd:restriction base="xsd:token">
        <xsd:enumeration value="S"/>
        <xsd:enumeration value="M"/>
        <xsd:enumeration value="L"/>
        <xsd:enumeration value="XL"/>
        <xsd:enumeration value="XXL"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:union>
</xsd:simpleType>
```

Default and fixed values

- XML Schema can provide default value for attributes and elements using `default` or `fixed` attributes.
- Fixed value — default and the only one allowed
- DTD can provide default and fixed values for attributes only!
- Default values are used (only) by XML processors taking schema into account.

Default and fixed values

- XML Schema can provide default value for attributes and elements using `default` or `fixed` attributes.
- Fixed value — default and the only one allowed
- DTD can provide default and fixed values for attributes only!
- Default values are used (only) by XML processors taking schema into account.

Fixed and default values

```
<xsd:element name="greeting" type="xsd:string"
    default="Hello,"/>

<xsd:attribute name="payment" type="TPaymentMeans"
    default="cash"/>
```

Rules for default and fixed values

Attributes

- attribute occurs, no matter what is its value $\rightarrow$ the actual (document) value is used
- attribute does not occur $\rightarrow$ the default (schema) value is used

Rules for default and fixed values

Attributes

- attribute occurs, no matter what is its value → the actual (document) value is used
- attribute does not occur → the default (schema) value is used

Elements

- element occurs and is not empty → the actual (document) value is used
- element occurs and is empty → the default (schema) value is used
- element does not occur → the default value is not used

Complex types

Application

Complex types apply to elements and specify:

- content model
- attributes

Complex types

Application

Complex types apply to elements and specify:

- content model
- attributes

Definition example

```
<xsd:complexType name="TPerson">
  <xsd:all>
    <xsd:element name="fname" type="xsd:string"/>
    <xsd:element name="lname" type="xsd:string"/>
  </xsd:all>
  <xsd:attribute name="sex" type="TSex"/>
</xsd:complexType>
```

Model groups

- Content model specified with **model groups**

Model groups

- Content model specified with **model groups**
- Three kinds of model groups:

  `<xsd:sequence>` all elements (or subgroups) in the given order (comma in DTD)

Model groups

- Content model specified with **model groups**
- Three kinds of model groups:

`<xsd:sequence>`	all elements (or subgroups) in the given order (comma in DTD)
`<xsd:choice>`	one from given elements or subgroups (| in DTD)

Model groups

- Content model specified with **model groups**
- Three kinds of model groups:

`<xsd:sequence>`	all elements (or subgroups) in the given order (comma in DTD)
`<xsd:choice>`	one from given elements or subgroups (| in DTD)
`<xsd:all>`	all elements in any order (no corresponding construct in DTD)

Model groups

- Content model specified with **model groups**
- Three kinds of model groups:

  `<xsd:sequence>` all elements (or subgroups) in the given order (comma in DTD)

  `<xsd:choice>` one from given elements or subgroups (| in DTD)

  `<xsd:all>` all elements in any order (no corresponding construct in DTD)
- Sequences and choices may be nested

Model groups

- Content model specified with **model groups**
- Three kinds of model groups:

  `<xsd:sequence>` all elements (or subgroups) in the given order (comma in DTD)

  `<xsd:choice>` one from given elements or subgroups (| in DTD)

  `<xsd:all>` all elements in any order (no corresponding construct in DTD)
- Sequences and choices may be nested
- Elements and subgroups can be repeated
  - number of occurrences under control

Number of occurrences

- Elements can be repeated within boundaries specified with `minOccurs` and `maxOccurs` attributes.
- Special value `unbounded` for upper limit
- Default occurrence number interval: [1, 1]
- `minOccurs` and `maxOccurs` may be also used for nested groups.

Controlling number of occurrences — examples

DTD

```
<!ELEMENT person (title?, name+, surname, address*)>
```

Controlling number of occurrences — examples

DTD

```
<!ELEMENT person (title?, name+, surname, address*)>
```

XML Schema

```
<xsd:element name="person">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="title" minOccurs="0"/>
      <xsd:element name="name"
                   maxOccurs="unbounded"/>
      <xsd:element name="surname"/>
      <xsd:element name="address" minOccurs="0"
                   maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Controlling number of occurrences — examples

DTD

```
<!ELEMENT person (title?, name+, surname, address*)>
```

XML Schema

```
<xsd:element name="person">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="title" minOccurs="0"/>
      <xsd:element name="name"
                   maxOccurs="unbounded"/>
      <xsd:element name="surname"/>
      <xsd:element name="address" minOccurs="0"
                   maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Controlling number of occurrences — examples

DTD

```
<!ELEMENT person (title?, name+, surname, address*)>
```

XML Schema

```
<xsd:element name="person">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="title" minOccurs="0"/>
      <xsd:element name="name"
                   maxOccurs="unbounded"/>
      <xsd:element name="surname"/>
      <xsd:element name="address" minOccurs="0"
                   maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Controlling number of occurrences — examples

DTD

```
<!ELEMENT person (title?, name+, surname, address*)>
```

XML Schema

```
<xsd:element name="person">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="title" minOccurs="0"/>
      <xsd:element name="name"
                   maxOccurs="unbounded"/>
      <xsd:element name="surname"/>
      <xsd:element name="address" minOccurs="0"
                   maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Number of occurrences — further examples

1. `<xsd:element name="a" minOccurs="2" maxOccurs="5"/>`

Number of occurrences — further examples

1. `<xsd:element name="a" minOccurs="2" maxOccurs="5"/>`
   Element may occur from 2 to 5 times (any content).

Number of occurrences — further examples

1. `<xsd:element name="a" minOccurs="2" maxOccurs="5"/>`
   Element may occur from 2 to 5 times (any content).
2. `<xsd:element name="b"/>`

Number of occurrences — further examples

1. `<xsd:element name="a" minOccurs="2" maxOccurs="5"/>`
   Element may occur from 2 to 5 times (any content).
2. `<xsd:element name="b"/>`
   Element must occur exactly one time (any content).

Number of occurrences — further examples

1. `<xsd:element name="a" minOccurs="2" maxOccurs="5"/>`
   Element may occur from 2 to 5 times (any content).
2. `<xsd:element name="b"/>`
   Element must occur exactly one time (any content).
3. `<xsd:element name="c" minOccurs="0" fixed="24"/>`

Number of occurrences — further examples

1. `<xsd:element name="a" minOccurs="2" maxOccurs="5"/>`
   Element may occur from 2 to 5 times (any content).
2. `<xsd:element name="b"/>`
   Element must occur exactly one time (any content).
3. `<xsd:element name="c" minOccurs="0" fixed="24"/>`
   Element may occur at least one time. If it occurs as an empty element, a validating parser will fill it with content "24". If it occurs as a non-empty element, its content must be equal to 24. If it does not occur in a document, the parser will not insert an element.

Number of occurrences — further examples

1. `<xsd:element name="a" minOccurs="2" maxOccurs="5"/>`
   Element may occur from 2 to 5 times (any content).
2. `<xsd:element name="b"/>`
   Element must occur exactly one time (any content).
3. `<xsd:element name="c" minOccurs="0" fixed="24"/>`
   Element may occur at least one time. If it occurs as an empty element, a validating parser will fill it with content "24". If it occurs as a non-empty element, its content must be equal to 24. If it does not occur in a document, the parser will not insert an element.
4. `<xsd:element name="d" maxOccurs="unbounded" default="24"/>`

Number of occurrences — further examples

1. `<xsd:element name="a" minOccurs="2" maxOccurs="5"/>`
   Element may occur from 2 to 5 times (any content).
2. `<xsd:element name="b"/>`
   Element must occur exactly one time (any content).
3. `<xsd:element name="c" minOccurs="0" fixed="24"/>`
   Element may occur at least one time. If it occurs as an empty element, a validating parser will fill it with content "24". If it occurs as a non-empty element, its content must be equal to 24. If it does not occur in a document, the parser will not insert an element.
4. `<xsd:element name="d" maxOccurs="unbounded" default="24"/>`
   Element must occur exactly once. If it occurs as an empty element, the (validating) parser will fill it with content "24". If it occurs as a non-empty element, the document value is used.

Choice

DTD

```
<!ELEMENT vehicle (train | aeroplane | car)>
```

Choice

DTD

```
<!ELEMENT vehicle (train | aeroplane | car)>
```

XML Schema

```
<xsd:element name="vehicle">
  <xsd:complexType>
    <xsd:choice>
      <xsd:element name="train"/>
      <xsd:element name="aeroplane"/>
      <xsd:element name="car"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

Choice

DTD

```
<!ELEMENT vehicle (train | aeroplane | car)>
```

XML Schema

```
<xsd:element name="vehicle">
  <xsd:complexType>
    <xsd:choice>
      <xsd:element name="train"/>
      <xsd:element name="aeroplane"/>
      <xsd:element name="car"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

Free combination of elements

Without occurrence control

DTD

```
<!ELEMENT vehicle (train | aeroplane | car)*>
```

Free combination of elements

Without occurrence control

DTD

```
<!ELEMENT vehicle (train | aeroplane | car)*>
```

XML Schema

```
<xsd:element name="vehicle">
  <xsd:complexType>
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="train"/>
      <xsd:element name="aeroplane"/>
      <xsd:element name="car"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

Free combination of elements

Without occurrence control

DTD

```
<!ELEMENT vehicle (train | aeroplane | car)*>
```

XML Schema

```
<xsd:element name="vehicle">
  <xsd:complexType>
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="train"/>
      <xsd:element name="aeroplane"/>
      <xsd:element name="car"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

All elements in any order

DTD

No such construct!

All elements in any order

DTD

No such construct!

XML Schema

```
<xsd:element name="book">
  <xsd:complexType>
    <xsd:all>
      <xsd:element name="title"/>
      <xsd:element name="author"/>
      <xsd:element name="publisher" minOccurs="0"/>
    </xsd:all>
  </xsd:complexType>
</xsd:element>
```

All elements in any order

DTD

No such construct!

XML Schema

```
<xsd:element name="book">
  <xsd:complexType>
    <xsd:all>
      <xsd:element name="title"/>
      <xsd:element name="author"/>
      <xsd:element name="publisher" minOccurs="0"/>
    </xsd:all>
  </xsd:complexType>
</xsd:element>
```

All elements in any order

Restrictions:

- cannot contain other groups, only elements,
- cannot be nested in other group,
- elements cannot occur more than once, `maxOccurs` ≤ 1.

Empty content

DTD

```
<!ELEMENT hr EMPTY>
```

Empty content

DTD

```
<!ELEMENT hr EMPTY>
```

XML Schema

```
<xsd:element name="hr">
  <xsd:complexType/>
</xsd:element>
```

Any content

Usually used to insert content from another namespace (e.g. nested XHTML, MathML, or SVG)

Any content

Usually used to insert content from another namespace (e.g. nested XHTML, MathML, or SVG)

For such purposes, XML Schema provides element `<xsd:any>` with two special attributes:

- `namespace` — namespace of allowed element; special values `#any` and `#other`
- `processContents` — should the inserted content be validated

  `skip` no validation
  `lax` validation if definition (schema) of a particular element available
  `strict` mandatory validation (default value)

Any content

DTD

```
<!ELEMENT nestedXHTML ANY>
```

Any content

DTD

```
<!ELEMENT nestedXHTML ANY>
```

XML Schema

```
<xsd:element name="nestedXHTML">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:any
           namespace="http://www.w3.org/1999/xhtml"
           maxOccurs="unbounded"
           processContents="skip"/>
      </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

Any attributes

XML Schema provides means to allow occurrence of any attributes (from a given namespace). It would be useful e.g. to enable XLink.

Any attributes

XML Schema provides means to allow occurrence of any attributes (from a given namespace). It would be useful e.g. to enable XLink.

Example

```
<xsd:element name="person">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="website">
        <xsd:complexType>
          <xsd:anyAttribute
             namespace="http://www.w3.org/1999/xlink"/>
        </xsd:complexType>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Defining attributes

Occurrence of attributes is controlled by attribute `use`, which possible values are:

- `required` must occur
- `optional` may occur (default value)
- `prohibited` must not occur (useful in type restrictions)

Defining attributes: DTD vs XML Schema

DTD

```
<!ELEMENT person EMPTY>
<!ATTLIST person pesel CDATA #REQUIRED
                 nip CDATA #IMPLIED>
                 species CDATA #FIXED "Homo sapiens">
```

Defining attributes: DTD vs XML Schema

DTD

```
<!ELEMENT person EMPTY>
<!ATTLIST person pesel CDATA #REQUIRED
                 nip CDATA #IMPLIED>
                 species CDATA #FIXED "Homo sapiens">
```

XML Schema

```
<xsd:element name="person">
  <xsd:complexType>
    <xsd:attribute name="pesel" use="required"/>
    <xsd:attribute name="nip"/>
    <xsd:attribute name="species"
                   fixed="Homo sapiens"/>
  </xsd:complexType>
</xsd:element>
```

Defining attributes: DTD vs XML Schema

DTD

```
<!ELEMENT person EMPTY>
<!ATTLIST person pesel CDATA #REQUIRED
                 nip CDATA #IMPLIED>
                 species CDATA #FIXED "Homo sapiens">
```

XML Schema

```
<xsd:element name="person">
  <xsd:complexType>
    <xsd:attribute name="pesel" use="required"/>
    <xsd:attribute name="nip"/>
    <xsd:attribute name="species"
                   fixed="Homo sapiens"/>
  </xsd:complexType>
</xsd:element>
```

Defining attributes: DTD vs XML Schema

DTD

```
<!ELEMENT person EMPTY>
<!ATTLIST person pesel CDATA #REQUIRED
                 nip CDATA #IMPLIED>
                 species CDATA #FIXED "Homo sapiens">
```

XML Schema

```
<xsd:element name="person">
  <xsd:complexType>
    <xsd:attribute name="pesel" use="required"/>
    <xsd:attribute name="nip"/>
    <xsd:attribute name="species"
                   fixed="Homo sapiens"/>
  </xsd:complexType>
</xsd:element>
```

Defining attributes: DTD vs XML Schema

DTD

```
<!ELEMENT person EMPTY>
<!ATTLIST person pesel CDATA #REQUIRED
                 nip CDATA #IMPLIED>
                 species CDATA #FIXED "Homo sapiens">
```

XML Schema

```
<xsd:element name="person">
  <xsd:complexType>
    <xsd:attribute name="pesel" use="required"/>
    <xsd:attribute name="nip"/>
    <xsd:attribute name="species"
                   fixed="Homo sapiens"/>
  </xsd:complexType>
</xsd:element>
```

Defining attributes: DTD vs XML Schema ctd.

DTD

```
<!ATTLIST person sex (woman | man) #REQUIRED>
```

Defining attributes: DTD vs XML Schema ctd.

DTD

```
<!ATTLIST person sex (woman | man) #REQUIRED>
```

XML Schema

```
<xsd:element name="person">
  <xsd:complexType>
    <xsd:attribute name="sex" use="required">
      <xsd:simpleType>
        <xsd:restriction base="xsd:NMTOKEN">
          <xsd:enumeration value="woman"/>
          <xsd:enumeration value="man"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:complexType>
</xsd:element>
```

Defining attributes: DTD vs XML Schema ctd.

DTD

```
<!ATTLIST person sex (woman | man) #REQUIRED>
```

XML Schema

```
<xsd:element name="person">
  <xsd:complexType>
    <xsd:attribute name="sex" use="required">
      <xsd:simpleType>
        <xsd:restriction base="xsd:NMTOKEN">
          <xsd:enumeration value="woman"/>
          <xsd:enumeration value="man"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:complexType>
</xsd:element>
```

Defining attributes — further examples

1. `<xsd:attribute name="a" use="required"/>`

Defining attributes — further examples

1. `<xsd:attribute name="a" use="required"/>`
   Attribute must occur, it may take any value.

Defining attributes — further examples

1. `<xsd:attribute name="a" use="required"/>`
   Attribute must occur, it may take any value.

2. `<xsd:attribute name="b"/>`

Defining attributes — further examples

1. `<xsd:attribute name="a" use="required"/>`
   Attribute must occur, it may take any value.

2. `<xsd:attribute name="b"/>`
   Attribute is optional, it may take any value.

Defining attributes — further examples

1. `<xsd:attribute name="a" use="required"/>`
   Attribute must occur, it may take any value.

2. `<xsd:attribute name="b"/>`
   Attribute is optional, it may take any value.

3. `<xsd:attribute name="c" fixed="24"/>`

Defining attributes — further examples

1. `<xsd:attribute name="a" use="required"/>`
   Attribute must occur, it may take any value.

2. `<xsd:attribute name="b"/>`
   Attribute is optional, it may take any value.

3. `<xsd:attribute name="c" fixed="24"/>`
   Attribute is optional. If not given in document it take the value of 24; If given in document, it must have the value of 24.

Defining attributes — further examples

1. `<xsd:attribute name="a" use="required"/>`
   Attribute must occur, it may take any value.

2. `<xsd:attribute name="b"/>`
   Attribute is optional, it may take any value.

3. `<xsd:attribute name="c" fixed="24"/>`
   Attribute is optional. If not given in document it take the value of 24; If given in document, it must have the value of 24.

4. `<xsd:attribute name="d" default="24"/>`

Defining attributes — further examples

1. `<xsd:attribute name="a" use="required"/>`
   Attribute must occur, it may take any value.

2. `<xsd:attribute name="b"/>`
   Attribute is optional, it may take any value.

3. `<xsd:attribute name="c" fixed="24"/>`
   Attribute is optional. If not given in document it take the value of 24; If given in document, it must have the value of 24.

4. `<xsd:attribute name="d" default="24"/>`
   Attribute is optional. If not given in document it take the value of 24; If given in document, it takes the actual value from document (even if it is empty string).

Mixed content model

DTD

```
<!ELEMENT text (#PCDATA | strong | link)*>
```

Mixed content model

DTD

```
<!ELEMENT text (#PCDATA | strong | link)*>
```

XML Schema

```
<xsd:element name="text">
  <xsd:complexType mixed="true">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="strong"/>
      <xsd:element name="link"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

Mixed content model

DTD

```
<!ELEMENT text (#PCDATA | strong | link)*>
```

XML Schema

```
<xsd:element name="text">
  <xsd:complexType mixed="true">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="strong"/>
      <xsd:element name="link"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

Mixed content model — XML Schema capabilities

XML Schema is capable to specify mixed models more precisely than DTD. In particular, it is possible to specify:

- order of elements (using `<xsd:sequence>`),
- number of occurrence
  (using `minOccurs` and `maxOccurs`).

Generally, a normal element content model is constructed and then arbitrary text is permitted to occur between subelements.

Global and local definitions

Elements and attributes defined directly in `<xsd:schema>` children are global; the rest are local.

Global and local definitions

Elements and attributes defined directly in `<xsd:schema>` children are global; the rest are local.

One can use globally defined components in any place, using `ref` attribute:

```
<xsd:element name="footnote">
  <xsd:complexType>
    <xsd:attribute name="nr"/>
  </xsd:complexType>
</xsd:element>

<xsd:element name="title">
  <xsd:complexType mixed="true">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element ref="footnote"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

Global vs local

- Global definitions allow author to modularize schema and avoid code duplication.
- Global element can always be used as root element of a document.
  - Use named groups or named complex types instead of global elements to prevent elements from being root elements.
- Local definitions allows author to define elements with the same name and different types.
- For some namespace-related reasons, in typical applications, attributes should be defined locally.

Note: Global and local definitions in DTD

In DTD

- All elemets are global.
- Element types are anonymous (except special types `ANY` and `EMPTY`).
- All attributes are local to their elements.
- Attribute types are predefined or anonymous.

Element groups
Named, global model groups

DTD

```
<!ENTITY % formatting 'b | i | u'>
```

Element groups
Named, global model groups

DTD

```
<!ENTITY % formatting 'b | i | u'>
```

XML Schema

```
<xsd:group name="formatting">
  <xsd:choice>
    <xsd:element ref="b"/>
    <xsd:element ref="i"/>
    <xsd:element ref="u"/>
  </xsd:choice>
</xsd:group>
```

Element groups
Named, global model groups

DTD

```
<!ENTITY % formatting 'b | i | u'>
```

XML Schema

```
<xsd:group name="formatting">
  <xsd:choice>
    <xsd:element ref="b"/>
    <xsd:element ref="i"/>
    <xsd:element ref="u"/>
  </xsd:choice>
</xsd:group>
```

Element groups
Named, global model groups

DTD

```
<!ENTITY % formatting 'b | i | u'>
```

XML Schema

```
<xsd:group name="formatting">
  <xsd:choice>
    <xsd:element ref="b"/>
    <xsd:element ref="i"/>
    <xsd:element ref="u"/>
  </xsd:choice>
</xsd:group>
```

DTD

```
<!ENTITY % time-attrs 'valid-from CDATA #IMPLIED
                       valid-to CDATA #IMPLIED'>

<!ENTITY % common-attrs 'status CDATA #IMPLIED
                        %time-attrs;'>
```

DTD

```
<!ENTITY % time-attrs 'valid-from CDATA #IMPLIED
                       valid-to CDATA #IMPLIED'>

<!ENTITY % common-attrs 'status CDATA #IMPLIED
                        %time-attrs;'>
```

XML Schema

```
<xsd:attributeGroup name="time-attrs">
  <xsd:attribute name="valid-from"/>
  <xsd:attribute name="valid-to"/>
</xsd:attributeGroup

<xsd:attributeGroup name="common-attrs">
  <xsd:attribute name="status"/>
  <xsd:attributeGroup ref="time-attrs"/>
</xsd:attributeGroup
```

DTD

```
<!ENTITY % time-attrs 'valid-from CDATA #IMPLIED
                       valid-to CDATA #IMPLIED'>

<!ENTITY % common-attrs 'status CDATA #IMPLIED
                       %time-attrs;'>
```

XML Schema

```
<xsd:attributeGroup name="time-attrs">
  <xsd:attribute name="valid-from"/>
  <xsd:attribute name="valid-to"/>
</xsd:attributeGroup

<xsd:attributeGroup name="common-attrs">
  <xsd:attribute name="status"/>
  <xsd:attributeGroup ref="time-attrs"/>
</xsd:attributeGroup
```

Constraints on uniqueness and references

Two mechanisms:

Constraints on uniqueness and references

Two mechanisms:

1. special attribute types `xsd:ID` and `xsd:IDREF` — like in DTD

Constraints on uniqueness and references

Two mechanisms:

1. special attribute types `xsd:ID` and `xsd:IDREF` — like in DTD
2. XML Schema **identity constraints**:

Constraints on uniqueness and references

Two mechanisms:

1. special attribute types `xsd:ID` and `xsd:IDREF` — like in DTD
2. XML Schema **identity constraints**:
   - `<xsd:unique>`

Constraints on uniqueness and references

Two mechanisms:

1. special attribute types `xsd:ID` and `xsd:IDREF` — like in DTD
2. XML Schema **identity constraints**:
   - `<xsd:unique>`
   - `<xsd:key>`

Constraints on uniqueness and references

Two mechanisms:

1. special attribute types `xsd:ID` and `xsd:IDREF` — like in DTD
2. XML Schema **identity constraints**:
   - `<xsd:unique>`
   - `<xsd:key>`
   - `<xsd:keyref>`

Identity constraints: definitions

Three components of identity constraint:

Identity constraints: definitions

Three components of identity constraint:

scope element the constraint is defined within

Identity constraints: definitions

Three components of identity constraint:

scope element the constraint is defined within

selector points elements the constraint applies to

Identity constraints: definitions

Three components of identity constraint:

- **scope** element the constraint is defined within
- **selector** points elements the constraint applies to
- **fields** points elements and attributes which values must be unique (for nodes selected by selector, within a given scope)

Identity constraints: definitions

Three components of identity constraint:

- **scope** element the constraint is defined within
- **selector** points elements the constraint applies to
- **fields** points elements and attributes which values must be unique (for nodes selected by selector, within a given scope)

Selectors and fields specified with (a subset of) XPath language

Become an XPath guru in 30 seconds

XPath is a language for addressing elements and attributes within a document tree.

Become an XPath guru in 30 seconds

XPath is a language for addressing elements and attributes within a document tree.

Syntax — almost like in file system:

Become an XPath guru in 30 seconds

XPath is a language for addressing elements and attributes within a document tree.

Syntax — almost like in file system:

- `book/chapter` — choose all `chapter` subelements from all `book` elements being children of current element,

Become an XPath guru in 30 seconds

XPath is a language for addressing elements and attributes within a document tree.

Syntax — almost like in file system:

- `book/chapter` — choose all `chapter` subelements from all `book` elements being children of current element,
- `*/chapter` — `chapter` subelements of all children of current element,

Become an XPath guru in 30 seconds

XPath is a language for addressing elements and attributes within a document tree.

Syntax — almost like in file system:

- `book/chapter` — choose all `chapter` subelements from all `book` elements being children of current element,
- `*/chapter` — `chapter` subelements of all children of current element,
- `@nr` — `nr` attribute of current element.

Become an XPath guru in 30 seconds

XPath is a language for addressing elements and attributes within a document tree.

Syntax — almost like in file system:

- `book/chapter` — choose all `chapter` subelements from all `book` elements being children of current element,
- `*/chapter` — `chapter` subelements of all children of current element,
- `@nr` — `nr` attribute of current element.

Namespaces complicate things a bit,

Become an XPath guru in 30 seconds

XPath is a language for addressing elements and attributes within a document tree.

Syntax — almost like in file system:

- `book/chapter` — choose all `chapter` subelements from all `book` elements being children of current element,
- `*/chapter` — `chapter` subelements of all children of current element,
- `@nr` — `nr` attribute of current element.

Namespaces complicate things a bit,
In fact, XPath is more complex — lectures 6–8.

Identity constraints: unique identifier

```
<accounts>
  <orders>
    <order>
      <number>125</number>
...
    </order>
    <order>
      <number>665</number>
...
    </order>
  </orders>
  <invoices>
  ...
  </invoices>
</accounts>
```

Identity constraints: unique identifier

```
<accounts>
  <orders>
    <order>
      <number>125</number>
...
    </order>
    <order>
      <number>665</number>
...
    </order>
  </orders>
  <invoices>
  ...
  </invoices>
</accounts>
```

Order numbers unique within document scope. They will be referred from invoices.

```
<xsd:key name="orderId">
   <xsd:selector
       xpath="orders
             /order"/>
   <xsd:field
       xpath="number"/>
</xsd:key>
```

Identity constraints: unique identifier

```
<accounts>
  <orders>
    <order>
      <number>125</number>
...
    </order>
    <order>
      <number>665</number>
...
    </order>
  </orders>
  <invoices>
  ...
  </invoices>
</accounts>
```

Order numbers unique within document scope. They will be referred from invoices.

```
<xsd:key name="orderId">
  <xsd:selector
       xpath="orders
             /order"/>
   <xsd:field
       xpath="number"/>
</xsd:key>
```

Identity constraints: unique identifier

```
<accounts>
  <orders>
    <order>
      <number>125</number>
...
    </order>
    <order>
      <number>665</number>
...
    </order>
  </orders>
  <invoices>
  ...
  </invoices>
</accounts>
```

Order numbers unique within document scope. They will be referred from invoices.

```
<xsd:key name="orderId">
   <xsd:selector
       xpath="orders
             /order"/>
  <xsd:field
       xpath="number"/>
</xsd:key>
```

Identity constraints: unique numbers and references

```
<accounts>
  ...
  <invoices>
    <invoice nr="123">
      <year>2007</year>
      <order-ref
          nr="125"/>...
    </invoice>
    <invoice nr="123">
      <year>2006</year>
      <order-ref
          nr="665"/>...
    </invoice>
  </invoices>
</accounts>
```

Identity constraints: unique numbers and references

```
<accounts>
  ...
  <invoices>
    <invoice nr="123">
      <year>2007</year>
      <order-ref
          nr="125"/>...
    </invoice>
    <invoice nr="123">
      <year>2006</year>
      <order-ref
          nr="665"/>...
    </invoice>
  </invoices>
</accounts>
```

Uniqueness of invoice numbers within a given year, in document scope:

```
<xsd:unique
     name="uniqueInvoiceNumber">

   <xsd:selector
       xpath="invoices
             /invoice"/>
  <xsd:field xpath="@nr"/>
  <xsd:field xpath="year"/>
</xsd:unique>
```

Identity constraints: unique numbers and references

```
<accounts>
  ...
  <invoices>
    <invoice nr="123">
      <year>2007</year>
      <order-ref
          nr="125"/>...
    </invoice>
    <invoice nr="123">
      <year>2006</year>
      <order-ref
          nr="665"/>...
    </invoice>
  </invoices>
</accounts>
```

Uniqueness of invoice numbers within a given year, in document scope:

```
<xsd:unique
     name="uniqueInvoiceNumber">

  <xsd:selector
       xpath="invoices
              /invoice"/>
  <xsd:field xpath="@nr"/>
  <xsd:field xpath="year"/>
</xsd:unique>
```

Identity constraints: unique numbers and references

```
<accounts>
  ...
  <invoices>
    <invoice nr="123">
      <year>2007</year>
      <order-ref
          nr="125"/>...
    </invoice>
    <invoice nr="123">
      <year>2006</year>
      <order-ref
          nr="665"/>...
    </invoice>
  </invoices>
</accounts>
```

Uniqueness of invoice numbers within a given year, in document scope:

```
<xsd:unique
     name="uniqueInvoiceNumber">

   <xsd:selector
       xpath="invoices
             /invoice"/>
  <xsd:field xpath="@nr"/>
  <xsd:field xpath="year"/>
</xsd:unique>
```

Identity constraints: unique numbers and references

```
<accounts>
  ...
  <invoices>
    <invoice nr="123">
      <year>2007</year>
      <order-ref
          nr="125"/>...
    </invoice>
    <invoice nr="123">
      <year>2006</year>
      <order-ref
          nr="665"/>...
    </invoice>
  </invoices>
</accounts>
```

Referencing order numbers:

```
<xsd:keyref
     name="orderRef"
     refer="orderId">
  <xsd:selector
       xpath="invoices
              /invoice
              /order-ref"/>
  <xsd:field xpath="@nr"/>
</xsd:keyref>
```

Identity constraints: unique numbers and references

```
<accounts>
  ...
  <invoices>
    <invoice nr="123">
      <year>2007</year>
      <order-ref
          nr="125"/>...
    </invoice>
    <invoice nr="123">
      <year>2006</year>
      <order-ref
          nr="665"/>...
    </invoice>
  </invoices>
</accounts>
```

Referencing order numbers:

```
<xsd:keyref
     name="orderRef"
     refer="orderId">
  <xsd:selector
       xpath="invoices
              /invoice
              /order-ref"/>
  <xsd:field xpath="@nr"/>
</xsd:keyref>
```

Identity constraints: unique numbers and references

```
<accounts>
  ...
  <invoices>
    <invoice nr="123">
      <year>2007</year>
      <order-ref
          nr="125"/>...
    </invoice>
    <invoice nr="123">
      <year>2006</year>
      <order-ref
          nr="665"/>...
    </invoice>
  </invoices>
</accounts>
```

Referencing order numbers:

```
<xsd:keyref
     name="orderRef"
     refer="orderId">
  <xsd:selector
       xpath="invoices
              /invoice
              /order-ref"/>
  <xsd:field xpath="@nr"/>
</xsd:keyref>
```

Identity constraints: all together

```
<xsd:element name="accounts" type="TAccounts">
  <xsd:unique name="uniqueInvoiceNumber">
    <xsd:selector xpath="invoices/invoice"/>
    <xsd:field xpath="@nr"/>
    <xsd:field xpath="year"/>
  </xsd:unique>
  <xsd:keyref name="orderRef"
              refer="orderId">
    <xsd:selector xpath="invoices/invoice/order-ref"/>

    <xsd:field xpath="@nr"/>
  </xsd:keyref>
  <xsd:key name="orderId">
    <xsd:selector xpath="orders/order"/>
    <xsd:field xpath="number"/>
  </xsd:key>
</xsd:element>
```