XML and Semantic Web Technologies

II. XML / 4. XML Schema

Lars Schmidt-Thieme

Information Systems and Machine Learning Lab (ISMLL)
Institute of Economics and Information Systems
& Institute of Computer Science
University of Hildesheim
http://www.ismll.uni-hildesheim.de

II. XML / 4. XML Schema

1. **Constraining Document Structure**

2. **Datatypes**

3. **Integrity Constraints**

4. **Schema Modularization**

5. **Namespaces in XML Schema**

6. **RELAX NG**

7. **First Applications**

XML Schema

The XML Schema recommendation consists of 3 parts:

0. Primer (non-normative)

1. Structures: XML Schema definition language (schema components & their XML representation)

2. Datatypes: datatype language.

version:

- Version 1.0, 2nd edition, W3C Recommendation of 2004/10/28.
- Work on requirements for XML Schema 1.1 is under way.
- XML Schema 1.0 is a XML 1.0 application.
- Namespace is `http://www.w3.org/2001/XMLSchema`.

Schema Element

```
<schema
 version = ⟨token⟩
 targetNamespace = ⟨anyURI⟩
 >
 Content: ( ⟨include⟩ | ⟨import⟩ | ⟨redefine⟩ | ⟨annotation⟩ )*
         ( ⟨element⟩ | ⟨attribute⟩
           | ⟨simpleType⟩ | ⟨complexType⟩
           | ⟨group⟩ | ⟨attributeGroup⟩)
           | ⟨notation⟩ | ⟨annotation⟩ )*
</schema>
```

To identify the elements in a document as elements of a schema, the schema namespace has to be used:

```
<?xml version="1.0"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">
</xs:schema>
```

Figure 1: Empty schema document.

Linking Schemas to Documents (no namespaces)

To link a schema to a document (that does not use namespaces) the attribute

noNamespaceSchemaLocation

from the schema instance namespace

http://www.w3.org/2001/XMLSchema-instance

is used.

Its value is an URI to a resource containing the schema.

```
<?xml version="1.1"?>
<persons xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="empty.xsd">
  <person><sn>Doe</sn><fn>John</fn></person>
  <person><fn>Alice</fn><sn>Meier</sn></person>
  <person><fn>Bob</fn><sn>Miller</sn></person>
</persons>
```

Figure 2: Linking a schema to a document.

To validate a document w.r.t. a schema, call xerces as:

```
xerces -v -s persons-empty.xml
```

Schema Components

Schema Component	main XML elements	section
primary components		
Element declarations	`element`	1
Attribute declarations	`attribute`	1
Simple type definitions	`simpleType`	2
Complex type definitions	`complexType`	2
secondary components		
Model group definitions	`group`	4
Attribute group definitions	`attributeGroup`	4
Identity-constraint definitions	`key`, `keyref`, `unique`	3
Notation declarations	`notation`	–
helper components		
Particles	`element`, `group`, `any`	1
Model groups	`all`, `choice`, `sequence`	1
Wildcards	`any`, `anyAttribute`	1
Attribute Uses	`attribute`	1
Annotations	`annotation`	–

Top-level of type hierarchy

Basically, a XML Schema associates

- each element with a simple or complex type and
- each attribute of every element with a simple type.

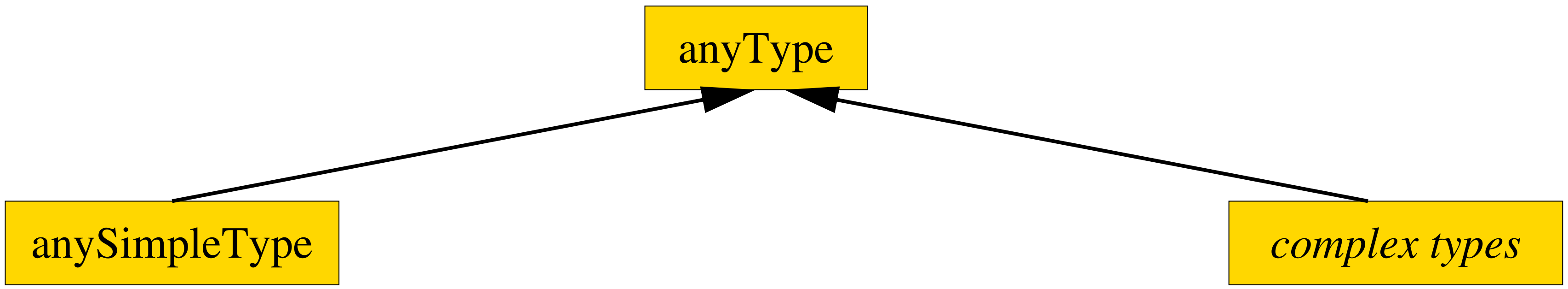


Figure 3: XML Schema Type hierarchy (top-level).

Simple types:

- strings, numeric, dates, or
- flat list of those (i.e., no nested lists).

Complex types: rich description of

- attributes and
- element contents.

Global Element Declaration

```
<element
 name = ⟨NCName⟩
 type = ⟨QName⟩
 default = ⟨string⟩
 fixed = ⟨string⟩
 >
 Content: ( ⟨simpleType⟩ | ⟨complexType⟩ )? ( ⟨unique⟩ | ⟨key⟩ | ⟨keyref⟩ )*
</element>
```

⟨NCName⟩ = non-colonized name (i.e., without ":"s);
⟨QName⟩ = qualified name (i.e., maybe with namespace).

The contents type of the element can be specified

- either by the type attribute (named type)
- or by declarations in the content of the element.

The default and fixed attribute allow the specification of a default / fixed value (if the empty literal is a valid literal of the content type).

Minimal Schema

```
<?xml version="1.1"?>
<persons xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="persons-minimal.xsd">
  <person><sn>Doe</sn><fn>John</fn></person>
  <person><fn>Alice</fn><sn>Meier</sn></person>
  <person><fn>Bob</fn><sn>Miller</sn></person>
</persons>
```

Figure 4: Example document.

```
<?xml version="1.0"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="persons"/>
</xs:schema>
```

Figure 5: Minimal schema `persons-minimal.xsd` s.t. the example document is valid w.r.t. that schema.

Element Declaration / Default Values

```
<?xml version="1.1"?>
<favorite-director xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="favorite-director.xsd"/>
```

Figure 6: Sample document with empty root element.

```
<?xml version="1.0"?>
<xs:schema version="1.0"
   xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="favorite-director" type="xs:string" default="Charlie Chaplin"/>
</xs:schema>
```

Figure 7: Schema with default value for an element.

```
<?xml version="1.1" encoding="UTF-8"?>
<favorite-director xsi:noNamespaceSchemaLocation="favorite-director.xsd">
 Charlie Chaplin</favorite-director>
```

Figure 8: Parsed document.

Complex Type Definition

```
<complexType
 name = ⟨NCName⟩
 mixed = ⟨boolean⟩ : false
 >
 Content: ⟨simpleContent⟩ | ⟨complexContent⟩
    | ( ( ⟨all⟩ | ⟨choice⟩ | ⟨sequence⟩ | ⟨group⟩ )?
       ( ⟨attribute⟩ | ⟨attributeGroup⟩ )* ⟨anyAttribute⟩? )
</complexType>
```

complexType can be used either

- anonymously, nested inside another element
  (e.g., the element element; name attribute must not be given), or
- named as top-level element
  (i.e., directly in the schema element; name attribute must be given).

Setting the mixed attribute to true allows mixed content
(i.e., arbitrary character data between the elements specified in the element content).

Complex Type Definition / Example

```
<?xml version="1.1"?>
<persons xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="persons-mixed.xsd">
  Doe, John
  Alice Meier
  Bob Miller
</persons>
```

Figure 9: Example document (valid).

```
<?xml version="1.0"?>
<xs:schema version="1.0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="persons">
    <xs:complexType mixed="true"/>
  </xs:element>
</xs:schema>
```

Figure 10: Schema with nested type.

```
<?xml version="1.0"?>
<xs:schema version="1.0"
    xmlns:xs="http://www.w3.org/2001/XML
  <xs:element name="persons"
              type="personsType"/>

  <xs:complexType name="personsType"
                  mixed="true"/>
</xs:schema>
```

Figure 11: Schema with referenced named type.

Model Groups / Sequences

```
<sequence
  maxOccurs = (⟨nonNegativeInteger⟩ | unbounded)  : 1
  minOccurs = ⟨nonNegativeInteger⟩ : 1
  >
  Content: ( ⟨element⟩ | ⟨choice⟩ | ⟨sequence⟩ | ⟨any⟩ | ⟨group⟩ )*
</sequence>
```

Every member model group must occur (as often as specified for the member) and in that order.

The model group as a whole must occur as often as specified in the sequence element.

Model Groups / Local Element Declaration and Element References

Nested local element declaration:

```
<element
 name = ⟨NCName⟩
 type = ⟨QName⟩
 default = ⟨string⟩
 fixed = ⟨string⟩
 maxOccurs = (⟨nonNegativeInteger⟩ | unbounded)  : 1
 minOccurs = ⟨nonNegativeInteger⟩ : 1
 >
 Content: ( ⟨simpleType⟩ | ⟨complexType⟩ )? ( ⟨unique⟩ | ⟨key⟩ | ⟨keyref⟩ )*
</element>
```

Element reference (to globally declared element):

```
<element
 ref = ⟨QName⟩
 maxOccurs = (⟨nonNegativeInteger⟩ | unbounded)  : 1
 minOccurs = ⟨nonNegativeInteger⟩ : 1
 />
```

minOccurs and maxOccurs allow the specification of cardinality constraints.

```
<?xml version="1.1"?>
<test xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="occur.xsd">
 <a/><a/><a/><b/>
 <a/><a/><a/><b/>
</test>
```

Figure 12: Sample Document.

```
<?xml version="1.0"?>
<xs:schema version="1.0"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="test">
  <xs:complexType>
   <xs:sequence minOccurs="2" maxOccurs="2">
    <xs:element name="a" minOccurs="2" maxOccurs="3"/>
    <xs:element name="b" minOccurs="0" maxOccurs="1"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Figure 13: Schema with sequence model group.

```
<?xml version="1.1"?>
<test xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="occur.xsd">
  <a/><a/>
  <a/><a/><a/><b/>
</test>
```

Figure 14: Another Sample Document.

```
<?xml version="1.0"?>
<xs:schema version="1.0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="test">
    <xs:complexType>
      <xs:sequence minOccurs="2" maxOccurs="2">
        <xs:element name="a" minOccurs="2" maxOccurs="3"/>
        <xs:element name="b" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 15: Schema with sequence model group.

Model Groups / Choices

<choice
 maxOccurs = (⟨*nonNegativeInteger*⟩ | unbounded) : 1
 minOccurs = ⟨*nonNegativeInteger*⟩ : 1
 >
 Content: (⟨*element*⟩ | ⟨*choice*⟩ | ⟨*sequence*⟩ | ⟨*any*⟩ | ⟨*group*⟩)*
</choice>

- Exactly one of the member model groups must occur (as often as specified for the member).

- The model group as a whole must occur as often as specified in the choice element.

- In effect: there must occur minOccurs to maxOccurs member model groups (in any order).

Model Groups / Choices / Example

```
<?xml version="1.1"?>
<article xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="article.xsd">
  <title>What <em>others</em> say</title>
  A <strong>short overview</strong> of basic and
  most important XML technologies is given in ...

  <em>Also</em> useful is ...
</article>
```

Figure 16: Sample Document.

```
<?xml version="1.0"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="article">
  <xs:complexType mixed="true">
   <xs:choice minOccurs="0" maxOccurs="unbounded">
    <xs:element ref="strong"/>
    <xs:element ref="em"/>
    <xs:element name="title">
     <xs:complexType mixed="true">
      <xs:choice minOccurs="0" maxOccurs="unbounded">
       <xs:element ref="strong"/>
       <xs:element ref="em"/>
      </xs:choice>
     </xs:complexType>
    </xs:element>
   </xs:choice>
  </xs:complexType>
 </xs:element>
 <xs:element name="strong" type="xs:string"/>
 <xs:element name="em" type="xs:string"/>
</xs:schema>
```

Figure 17: Schema with choice model group.

Model Groups / Collections

```
<all
  maxOccurs = 1 : 1
  minOccurs = (0 | 1) : 1
  >
  Content: ⟨element⟩*
</all>
```

minOccurs and maxOccurs must be 0 or 1 for member model groups.

Each of the member model groups must occur
(exactly once or at least once, as specified for the member) in arbitrary order.

The model group as a whole must occur as often as specified in the all element
(exactly once or at least once).

Model Groups / Collections / Example (1/2)

```
<?xml version="1.1"?>
<persons xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="persons-perm.xsd">
  <person><sn>Doe</sn><fn>John</fn></person>
  <person><fn>Alice</fn><sn>Meier</sn></person>
  <person><fn>Bob</fn><sn>Miller</sn></person>
</persons>
```

Figure 18: Sample Document.

Model Groups / Collections / Example (2/2)

```
<?xml version="1.0"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="persons">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="person" maxOccurs="unbounded">
     <xs:complexType>
      <xs:all>
       <xs:element name="fn" type="xs:string"/>
       <xs:element name="sn" type="xs:string"/>
      </xs:all>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Figure 19: Schema with collection model group.

Model Groups / Collections / Restrictions

The use of the all model group is severly restricted:

- it can only contain element members (but not nested sequence, choice, etc. members),
- the cardinality restrictions of its element members have to be 0 or 1,
- it can only occur as top-level model-group of a complex type (but not nested in a sequence, choice, etc. model group).

Thus, one cannot express with XML Schema, e.g., that a book element should consist of

- one or more authors,
- one title, and
- one year

in arbitrary order (unless one enumerates all permutations).

Model Groups / Element Wildcards

```
<any
 namespace = ##any | ##other
                  | List of ( ⟨anyURI⟩ | ##targetNamespace | ##local )  : ##any
 processContents = (strict | skip | lax) : strict
 maxOccurs = (⟨nonNegativeInteger⟩ | unbounded)  : 1
 minOccurs = ⟨nonNegativeInteger⟩ : 1
 />
```

##any: any namespace.

##targetNamespace: namespace that is defined by the actual schema (see section 4),

##other: any namespace except the target namespace,

##local: empty namespace.

strict: matches any valid element.

skip: matches any element (no validation, just well-formed XML).

lax: matches any valid element as well as any element with undefined name ("validate where you can, don’t worry when you can’t").

```
<?xml version="1.1"?>
<notes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="notes-simple.xsd">
 <todo>complete exercise sheet #2.</todo>
 <deadline>Wed. 19.5.</deadline>
</notes>
```

Figure 20: A Sample Document.

```
<?xml version="1.0"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="notes">
  <xs:complexType mixed="true">
   <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:any namespace="##targetNamespace"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="em" type="xs:string"/>
 <xs:element name="todo" type="xs:string"/>
</xs:schema>
```

Figure 21: Schema with element wildcard.

Attribute Declaration

a) Global or local attribute declaration:

```
<attribute
  name = ⟨NCName⟩
  type = ⟨QName⟩
  default = ⟨string⟩
  fixed = ⟨string⟩
  use = (optional | prohibited | required) : optional
  >
  Content: ⟨simpleType⟩?
</attribute>
```

b) Attribute reference (to gloablly declared attribute):

```
<attribute
  ref = ⟨QName⟩
  default = ⟨string⟩
  fixed = ⟨string⟩
  use = (optional | prohibited | required) : optional
  />
```

Attribute Wildcard

c) Attribute wildcard:

```
<anyAttribute
  namespace = ##any | ##other
             | List of (⟨anyURI⟩ | ##targetNamespace | ##local) : ##any
  processContents = (strict | lax | skip) : strict
  />
```

The attributes of attribute declarations, references, and wildcards
have the same semantics
as in corresponding constructs for elements.

```
<?xml version="1.1"?>
<books xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="books-att.xsd">
 <book isbn="isbn-0-596-00420-6" year="2003">
  <author>Erik T. Ray</author><title>Learning XML</title></book>
 <book isbn="isbn-1-565-92580-7" year="1999">
  <author>Norman Walsh and Leonard Muellner</author>
  <title>DocBook: The Definitive Guide</title></book>
</books>
```

Figure 22: A Sample Document.

```
    <xs:element name="book">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="author" minOccurs="1" maxOccurs="unbounded" type
       <xs:element name="title" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="year" type="xs:gYear"/>
      <xs:attribute name="isbn" type="xs:string"/>
     </xs:complexType>
    </xs:element>
```

Figure 23: Schema with attributes (excerpt).

Global vs. Local Elements and Attributes

Global: located at top-level (i.e., direct children of schema element).

Local: located at deeper level (i.e., nested inside another declaration).

Parsers match start elements (e.g., root elements) only with global elements.

Unique Particle Attribution

Particles contribute to the content model of an element:

- local element declarations and references (**element**),
- model group references (**group**),
- element wildcards (**any**).

Elements in instance documents must be attributed to a unique particle during validation
(unique particle attribution constraint; XML Schema Part 1, appendix H).

Particles are identified by their position in (the parse tree of) the schema and linked to instances by the API-independent so called **post schema-validation infoset**.

Unique Particle Attribution / Example (1/2)

```
<?xml version="1.1"?>
<test xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="occur.xsd">
  <a/><a/>
  <a/><a/><a/><b/>
</test>
```

Figure 24: Sample document.

```
<?xml version="1.0"?>
<xs:schema version="1.0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="test">
    <xs:complexType>
      <xs:sequence minOccurs="2" maxOccurs="2">
        <xs:element name="a" minOccurs="2" maxOccurs="3"/>
        <xs:element name="b" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 25: Schema allowing different "groupings" of a particle.

Unique Particle Attribution / Example (2/2)

```
<?xml version="1.0"?>
<xs:schema version="1.0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="test">
    <xs:complexType>
      <xs:sequence minOccurs="1" maxOccurs="1">
        <xs:element name="a" minOccurs="2" maxOccurs="3"/>
        <xs:element name="b" minOccurs="0" maxOccurs="1"/>
        <xs:element name="a" minOccurs="2" maxOccurs="3"/>
        <xs:element name="b" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 26: Schema with different particles matching elements in the sample document.

II. XML / 4. XML Schema

1. Constraining Document Structure

2. Datatypes

3. Integrity Constraints

4. Schema Modularization

5. Namespaces in XML Schema

6. RELAX NG

7. First Applications

Type system

A datatype is a 3-tuple, consisting of

1. a set of distinct values (**value space**),
2. a set of lexical representations (**lexical space**), i.e., a set of literals, and
3. a set of defining aspects that characterize properties of the value space, individual values or lexical items (**facets**).

A **canonical lexical representation** is a one-to-one mapping between the value space and a subset of the lexical space.

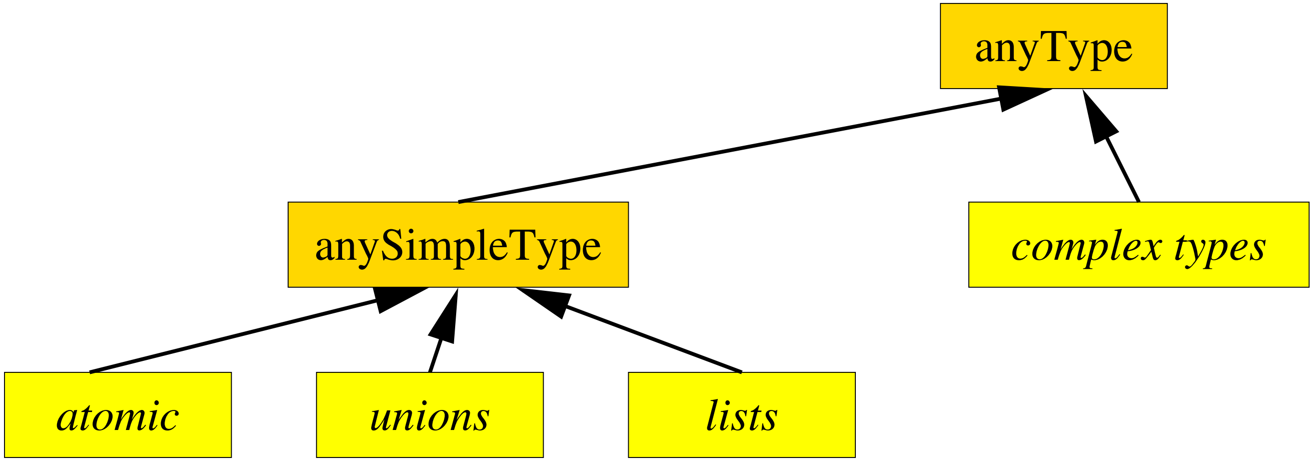


Figure 27: XML Schema type hierarchy (first two levels).

Primitive Atomic Types

XML Schema Datatypes defines 19 primitive atomic datatypes.

Thereby means

atomic: that the value space has no internal structure.

primitive: that the datatype is not derived formally from another datatype.

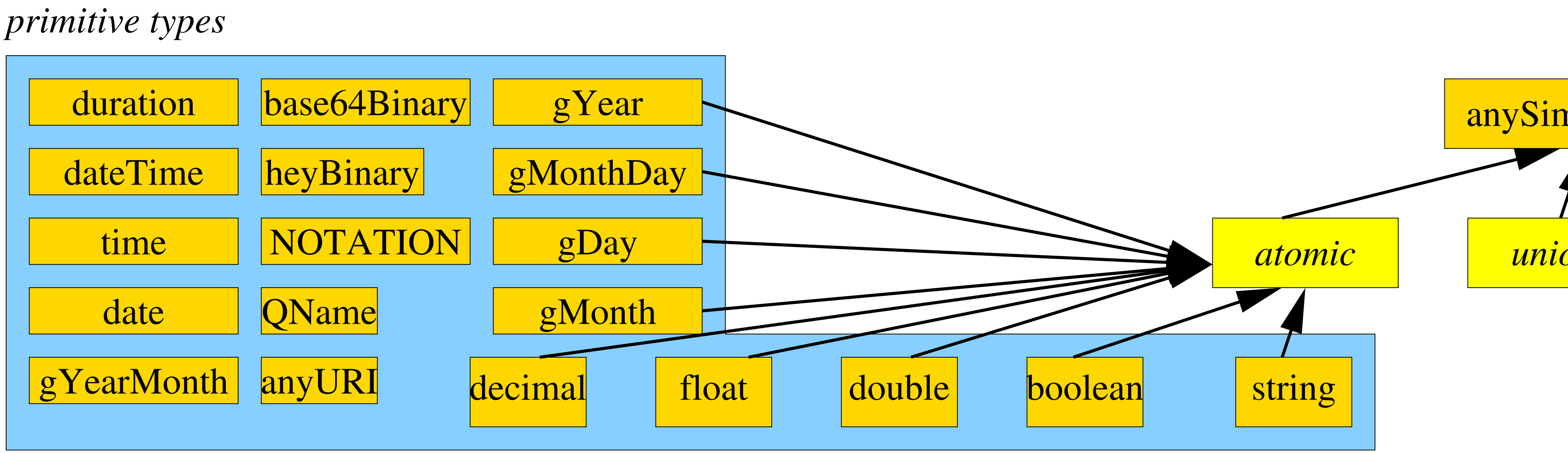


Figure 28: XML Schema built-in primitive atomic types.

Type Derivation

Beneath primitive types, XML Schema Datatypes provides three mechanisms for deriving types from other types:

restriction: a datatype which's value and lexical space is a
subset of its **base type**
(specified by a set of constraining facets)

list: a datatype which's value and lexical space consists of
lists / **tuples** of values/lexical items form the respective spaces of its **item type**.

union: a datatype which's value and lexical space is the
union of the respective spaces of its **member types**.

Constraining Facets

facet	string	anyURI	QName	boolean	decimal	float, double	duration, dateTime, …	hexBinary, base64Binary	NOTATION	lists	unions
length, minLength, maxLength	+	+	+					+	+	+	
totalDigits, fractionDigits					+						
pattern	+	+	+	+	+	+	+	+	+	+	+
enumeration	+	+	+		+	+	+	+	+	+	+
whiteSpace	+	+	+	+				+	+	+	
minInclusive, maxInclusive, minExclusive, maxExclusive					+	+	+				

Simple Type Definitions

<simpleType name = ⟨*NCName*⟩>
 Content: ⟨*restriction*⟩ | ⟨*list*⟩ | ⟨*union*⟩
</simpleType>

<restriction base = ⟨*QName*⟩>
 Content: ⟨*simpleType*⟩?
  (⟨*length*⟩ | ⟨*minLength*⟩ | ⟨*maxLength*⟩
  | ⟨*totalDigits*⟩ | ⟨*fractionDigits*⟩
  | ⟨*pattern*⟩ | ⟨*enumeration*⟩ | ⟨*whiteSpace*⟩
  | ⟨*minInclusive*⟩ | ⟨*maxInclusive*⟩ | ⟨*minExclusive*⟩ | ⟨*maxExclusive*⟩)*
</restriction>

The base type can be specified by

- either **base** attribute
- or nested **simpleType**-element.

Values of constraining facets are specified with an attribute **value**.

Derivation by Restriction / Example

```
<?xml version="1.0"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:simpleType name="keyword">
  <xs:restriction base="xs:string"><xs:maxLength value="100"/></xs:restriction>
 </xs:simpleType>
 <xs:simpleType name="smallnumber">
  <xs:restriction base="xs:decimal">
   <xs:totalDigits value="3"/>
   <xs:fractionDigits value="0"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:simpleType name="odd-smallnumber">
  <xs:restriction base="smallnumber">
   <xs:pattern value="([1-9][0-9]*)?[13579]"/>
  </xs:restriction>
 </xs:simpleType>
</xs:schema>
```

Figure 29: Schema with type derivations by restriction.

Built-in Derived Atomic Types

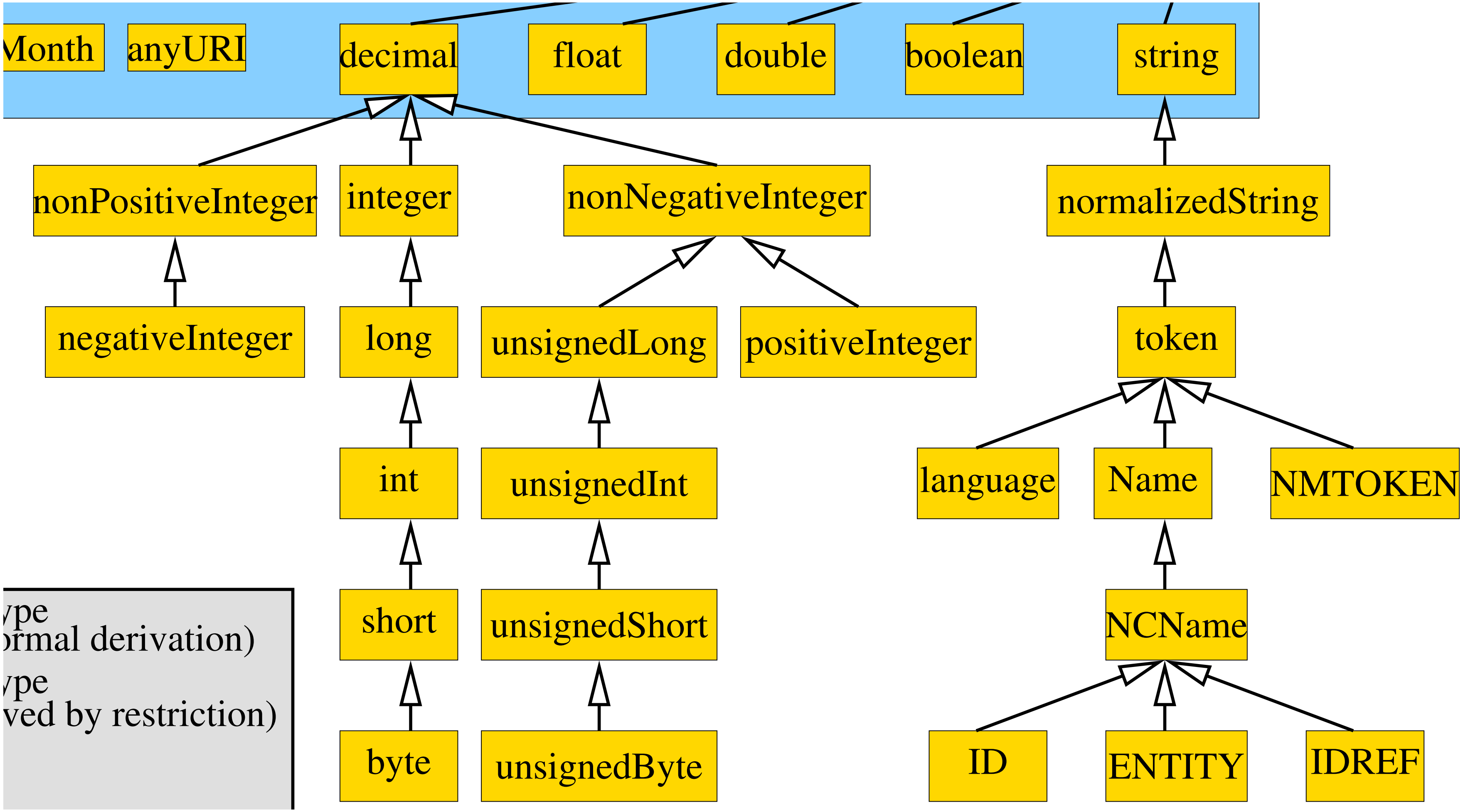


Figure 30: XML Schema built-in derived atomic types.

Derivation by List and Union

<list itemType = ⟨*QName*⟩>
 Content: ⟨*simpleType*⟩?
</list>

<union memberTypes = List of ⟨*QName*⟩>
 Content: ⟨*simpleType*⟩*
</union>

There are no built-in union types.

Built-in List Types

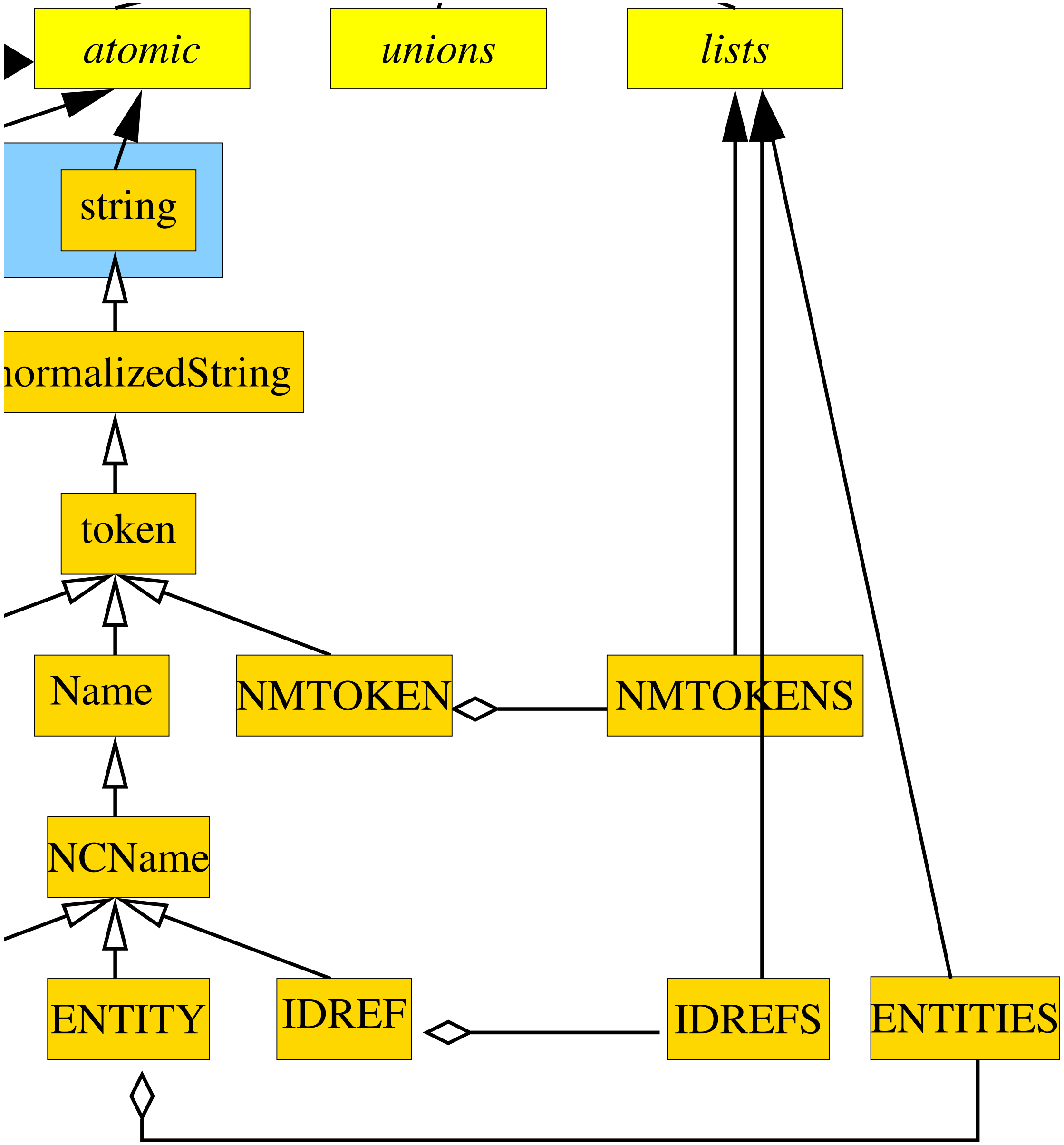


Figure 31: XML Schema built-in list types.

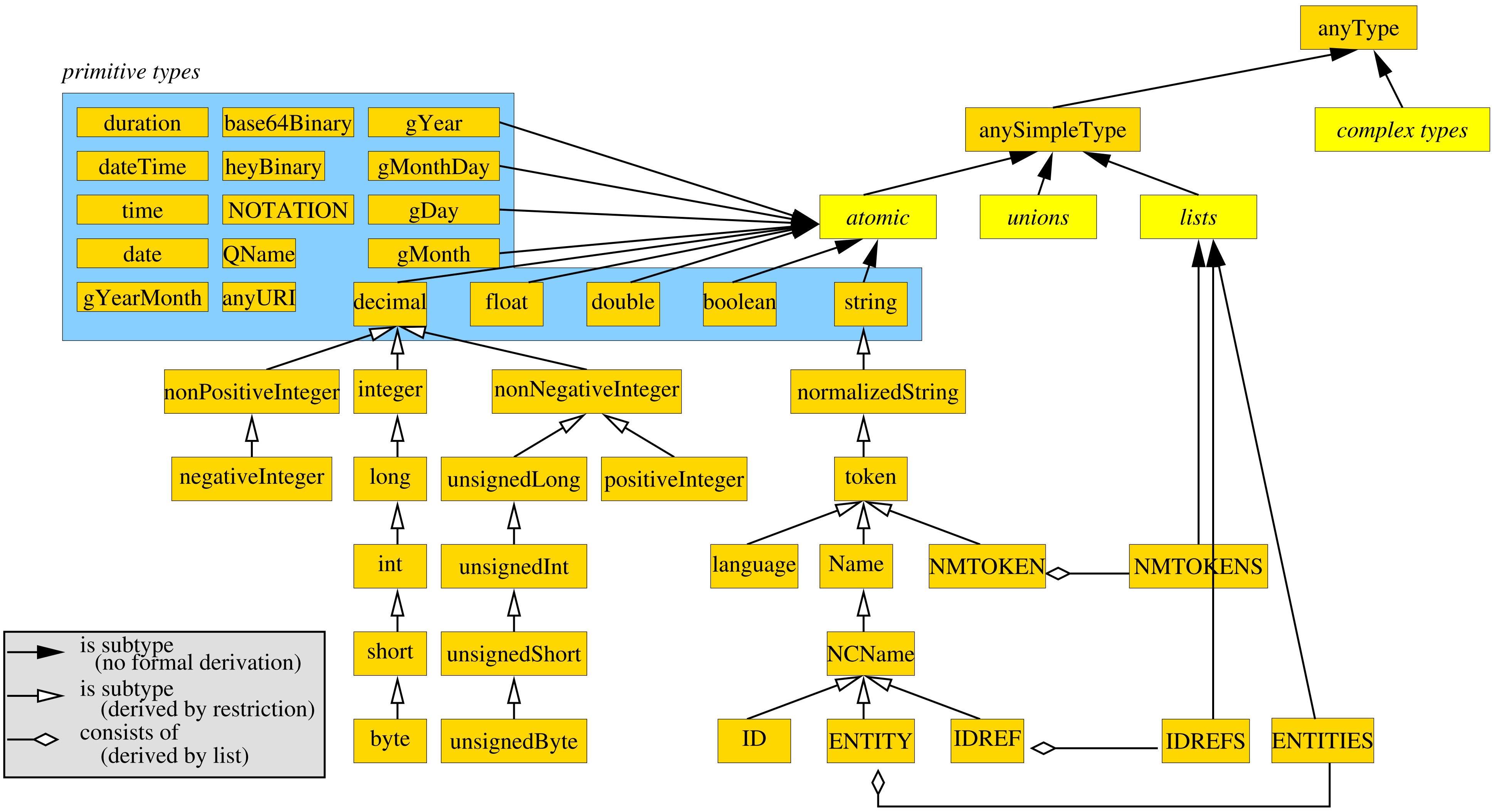


Figure 32: XML Schema built-in datatypes.

Complex Type Definition

```
<complexType
 name = ⟨NCName⟩
 mixed = ⟨boolean⟩ : false
 >
 Content: ⟨simpleContent⟩ | ⟨complexContent⟩
   | ( ( ⟨all⟩ | ⟨choice⟩ | ⟨sequence⟩ | ⟨group⟩ )?
      ( ⟨attribute⟩ | ⟨attributeGroup⟩ )* ⟨anyAttribute⟩? )
</complexType>
```

Complex type: content type plus attributes.

Simple content: content type is a simple type (i.e., no nested elements), but the element still may have attributes.

There are no built-in complex types (except **anyType**).

Complex Type Definition / Simple Content / Extension

<simpleContent>
    Content: ⟨*restriction*⟩ | ⟨*extension*⟩
</simpleContent>

<extension base = ⟨*QName*⟩>
  Content: (⟨*attribute*⟩ | ⟨*attributeGroup*⟩)* ⟨*anyAttribute*⟩?
</extension>

The extension element adds attributes to complex datatype with simple content type.

The base type must be

- a simple type or
- a complex type with simple content type.

```
<?xml version="1.0"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="price">
    <xs:simpleContent>
      <xs:extension base="xs:decimal">
        <xs:attribute name="currency" type="xs:string"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:element name="prices"/>
  <xs:element name="price" type="price"/>
</xs:schema>
```

Figure 33: XML schema with complex type with simple content.

```
<?xml version="1.1"?>
<prices xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="prices.xsd">
  <price currency="EUR">1200</price>
  <price currency="USD">67.99</price>
</prices>
```

Figure 34: Document instance of the schema above.

Complex Type Definition / Simple Content / Restriction

```
<restriction base = ⟨QName⟩>
 Content: ⟨simpleType⟩?
        ( ⟨length⟩ | ⟨minLength⟩ | ⟨maxLength⟩
        | ⟨totalDigits⟩ | ⟨fractionDigits⟩
        | ⟨pattern⟩ | ⟨enumeration⟩ | ⟨whiteSpace⟩
        | ⟨minInclusive⟩ | ⟨maxInclusive⟩ | ⟨minExclusive⟩ | ⟨maxExclusive⟩ )*
        ( ⟨attribute⟩ | ⟨attributeGroup⟩ )* ⟨anyAttribute⟩?
</restriction>
```

The restriction element

- adds constraining facets to the simple content type (the same as the restriction element for simpleTypes)

- restricts attributes by
  - narrowing their type,
  - omitting optional attributes, or
  - making optional attributes required.

  (all attributes of the new type have to be declared explicitly).

```
<xs:complexType name="price">
  <xs:simpleContent>
    <xs:extension base="xs:decimal">
      <xs:attribute name="currency" type="xs:string"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="international-price">
  <xs:simpleContent>
    <xs:restriction base="price">
      <xs:attribute name="currency" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="EUR"/>
            <xs:enumeration value="USD"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:restriction>
  </xs:simpleContent>
</xs:complexType>
```

Figure 35: XML schema with complex type with simple content (excerpt).

Complex Type Definition / Complex Content

```
<complexContent mixed = ⟨boolean⟩ >
 Content: ⟨restriction⟩ | ⟨extension⟩
</complexContent>

<restriction base = ⟨QName⟩ >
 Content: ( ⟨all⟩ | ⟨choice⟩ | ⟨sequence⟩ | ⟨group⟩ )?
         ( ⟨attribute⟩ | ⟨attributeGroup⟩ )* ⟨anyAttribute⟩?
</restriction>

<extension base = ⟨QName⟩ >
 Content: ( ⟨all⟩ | ⟨choice⟩ | ⟨sequence⟩ | ⟨group⟩ )?
         ( ⟨attribute⟩ | ⟨attributeGroup⟩ )* ⟨anyAttribute⟩?
</extension>
```

```
<?xml version="1.0"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:complexType name="personName">
  <xs:sequence>
   <xs:element name="title" minOccurs="0"/>
   <xs:element name="forename" minOccurs="0" maxOccurs="unbounded"/>
   <xs:element name="surname"/>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="extendedName">
  <xs:complexContent>
   <xs:extension base="personName">
    <xs:sequence>
     <xs:element name="generation" minOccurs="0"/>
    </xs:sequence>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
 <xs:element name="addressee" type="extendedName"/>
 <xs:element name="names"/>
</xs:schema>
```

Figure 36: XML schema with complex type with complex content / extension.

```
<?xml version="1.1"?>
<names xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="names.xsd">
  <addressee>
    <forename>Albert</forename>
    <forename>Arnold</forename>
    <surname>Gore</surname>
    <generation>Jr</generation>
  </addressee>
</names>
```

Figure 37: Sample document.

```
<xs:complexType name="simpleName">
  <xs:complexContent>
    <xs:restriction base="personName">
      <xs:sequence>
        <xs:element name="forename" minOccurs="1" maxOccurs="1"/>
        <xs:element name="surname"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

<xs:element name="who" type="simpleName"/>
```

Figure 38: XML schema with complex type with complex content / restriction (excerpt).

```
<who><forename>Bill</forename><surname>Clinton</surname></who>
```

Figure 39: Sample document (excerpt).

II. XML / 4. XML Schema

1. **Constraining Document Structure**

2. **Datatypes**

3. **Integrity Constraints**

4. **Schema Modularization**

5. **Namespaces in XML Schema**

6. **RELAX NG**

7. **First Applications**

Defining Keys (1/3)

<unique name = ⟨*NCName*⟩>

Content: ⟨*selector*⟩ ⟨*field*⟩+

</unique>

unique requires the values of a key to be unique.

<key name = ⟨*NCName*⟩>

Content: ⟨*selector*⟩ ⟨*field*⟩+

</key>

key furthermore requires each selected element to have a key.

<selector xpath = ⟨*SimpleXPath*⟩/>

selector specifies a set of elements (relative to the element it is defined in) for which a key is defined.

<field xpath = ⟨*SimpleXPath*⟩/>

field specifies a set of elements or attributes (relative to a selected element) which's values make the key.

Defining Keys (2/3)

Simple XPath expressions for **xpath** attribute of elements **selector** and **field**, respectively:

⟨*SimpleXPath*⟩ := ⟨*Path*⟩ (`|` ⟨*Path*⟩)*

⟨*Path.selector*⟩ := (`.//`)? (⟨*Step*⟩ `/`)* ⟨*Step*⟩

⟨*Path.field*⟩ := (`.//`)? (⟨*Step*⟩ `/`)* (⟨*Step*⟩ | `@` ⟨*NameTest*⟩)

⟨*Step*⟩ := `.` | ⟨*NameTest*⟩

⟨*NameTest*⟩ := ⟨*QName*⟩ | `*` | ⟨*NCName*⟩ `:` `*`

Defining Keys (3/3)

⟨*SimpleXPath*⟩ selects a set of elements or attributes relative to the context element:

".": the context element, i.e.,

- for **selector** the parent element of the **key**, **unique**, or **keyref** element,
- for **field** the selected element (i.e., the elements in the **selector** node set),

"/elem": all **children elements** with name "elem" of the elements of the previous step,

"/*": all children elements of the elements of the previous step,

"/ns:*": all children elements with namespace "ns" of the elements of the previous step,

"/@att": all attributes with name "att" of the elements of the previous step,

".//elem", ".//*", ".//ns:*", ".//@att": all **descendent elements** with name "elem" of the context element, . . . , all attributes with name "att" of descendant elements of the context element.

"|" takes unions of its operand node sets.

Referencing Keys

```
<keyref
  name = ⟨NCName⟩
  refer = ⟨QName⟩
  >
  Content: ⟨selector⟩ ⟨field⟩+
</keyref>
```

keyref references a key.
The name of the key referenced is given with **refer**.
selector defines the elements that contain the key reference.
field defines the elements or attributes which's values make the key reference.

```
<?xml version="1.1" encoding="UTF-8" ?>
<books xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="books-isbn.xsd">
  <book isbn="3-89864-222-4" cites="0-596-00420-6">
    <author>Rainer Eckstein</author><author>Silke Eckstein</author>
    <title>XML und Datenmodellierung</title><year>2004</year></book>
  <book isbn="0-596-00420-6">
    <author>Erik T. Ray</author><title>Learning XML</title><year>2003</year></bo
</books>
```

Figure 40: A document containing keys and key references.

```
<?xml version="1.0"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="books">
  <xs:complexType>
   <xs:sequence maxOccurs="unbounded">
    <xs:element ref="book"/>
   </xs:sequence>
  </xs:complexType>
  <xs:key name="isbnkey">
   <xs:selector xpath="book"/>
   <xs:field xpath="@isbn"/>
  </xs:key>
  <xs:keyref name="citesref" refer="isbnkey">
   <xs:selector xpath="book"/>
   <xs:field xpath="@cites"/>
  </xs:keyref>
 </xs:element>
```

Figure 41: XML schema defining identity constraints.

```
 <xs:element name="book">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="author" minOccu
    <xs:element name="title" type="xs:st
    <xs:element name="year" type="xs:g
   </xs:sequence>
   <xs:attribute name="isbn" type="xs:str
   <xs:attribute name="cites" type="xs:st
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Keys in XML Schema vs. ID/IDREF in DTDs [XML Schema 3.11.1]

- Functioning as a part of an identity-constraint is in addition to, not instead of, having a type.
- Not just attribute values, but also
  - element content (if it is of simple type) and
  - combinations of values and content

  can be declared to be unique.
- Identity-constraints are specified to hold within the scope of particular elements.
- (Combinations of) attribute values and/or element content can be declared to be keys, that is, not only unique, but always present and non-nillable;
- The comparison between keyref fields and key or unique fields is by value equality, not by string equality.

But one cannot define in XML schema a simple list type of key references (as e.g., IDREFS attributes in DTDs).

II. XML / 4. XML Schema

1. **Constraining Document Structure**

2. **Datatypes**

3. **Integrity Constraints**

4. **Schema Modularization**

5. **Namespaces in XML Schema**

6. **RELAX NG**

7. **First Applications**

Attribute Groups

a) Definition of an attribute group:

<attributeGroup name = ⟨*NCName*⟩>
 Content: (⟨*attribute*⟩ | ⟨*attributeGroup*⟩)* ⟨*anyAttribute*⟩?
</attributeGroup>

b) Reference of an attribute group (allowed whereever attributes are allowed):

<attributeGroup ref = ⟨*QName*⟩/>

Model Group Definitions

a) Definition of a model group:

<group name = ⟨*NCName*⟩>
 Content: ⟨*all*⟩ | ⟨*choice*⟩ | ⟨*sequence*⟩
</group>

b) Reference of a model group (allowed whereever **sequence**, etc. are allowed):

<group
 ref = ⟨*QName*⟩
 maxOccurs = (⟨*nonNegativeInteger*⟩ | unbounded) : 1
 minOccurs = ⟨*nonNegativeInteger*⟩ : 1
 />

Attribute and model groups are a simple macro facility
(as parameter entities in DTDs).

In most cases a cleaner design can be achieved by

- factoring out a complex type or
- using extensions or restrictions of complex types.

```
<?xml version="1.1"?>
<hardware xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="computer.xsd">
  <computer name="R2D2" price="2500" cpu="P IV 3.0 GHz"/>
  <computer name="C3PO" price="7200" cpu="Crusoe"/>
  <monitor name="TFT1320" price="499" size="18"/>
</hardware>
```

Figure 42: Example document with elements with common attributes.

```
<xs:attributeGroup name="attHardware">
 <xs:attribute name="name" type="xs:string"/>
 <xs:attribute name="price" type="xs:decimal"/>
</xs:attributeGroup>
<xs:complexType name="computer">
 <xs:attributeGroup ref="attHardware"/>
 <xs:attribute name="cpu" type="xs:string"/>
</xs:complexType>
<xs:complexType name="monitor">
 <xs:attributeGroup ref="attHardware"/>
 <xs:attribute name="size" type="xs:decimal"/>
</xs:complexType>

<xs:element name="hardware">
 <xs:complexType><xs:choice maxOccurs="unbounded">
  <xs:element name="computer" type="computer"/>
  <xs:element name="monitor" type="monitor"/>
 </xs:choice></xs:complexType>
</xs:element>
```

Figure 43: Schema with attribute group (excerpt).

```
<xs:complexType name="hardware">
  <xs:attribute name="name" type="xs:string"/>
  <xs:attribute name="price" type="xs:decimal"/>
</xs:complexType>
<xs:complexType name="computer">
  <xs:complexContent>
    <xs:extension base="hardware">
      <xs:attribute name="cpu" type="xs:string"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="monitor">
  <xs:complexContent>
    <xs:extension base="hardware">
      <xs:attribute name="size" type="xs:decimal"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 44: Schema with element extension (excerpt).

```
<xs:complexType name="hardware">
  <xs:attribute name="name" type="xs:string"/>
  <xs:attribute name="price" type="xs:decimal"/>
</xs:complexType>
<xs:complexType name="computer">
  <xs:sequence>
    <xs:element name="basic" type="hardware"/>
  </xs:sequence>
  <xs:attribute name="cpu" type="xs:string"/>
</xs:complexType>
<xs:complexType name="monitor">
  <xs:sequence>
    <xs:element name="basic" type="hardware"/>
  </xs:sequence>
  <xs:attribute name="size" type="xs:decimal"/>
</xs:complexType>
```

Figure 45: Schema with factored-out element (excerpt).

II. XML / 4. XML Schema

1. **Constraining Document Structure**

2. **Datatypes**

3. **Integrity Constraints**

4. **Schema Modularization**

5. **Namespaces in XML Schema**

6. **RELAX NG**

7. **First Applications**

Linking Schemas to Documents (namespaces)

To link a schema to a document (that does use namespaces)

1. the elements/attributes have to be associated with a namespace and
2. the location of the schema can be specified by the attribute

   schemaLocation

   from the schema instance namespace

   http://www.w3.org/2001/XMLSchema-instance

The value of **schemaLocation** is a sequence of pairs consisting of

- a namespace URI and
- an URL to a resource containing the schema.

Linking Schemas to Documents (namespaces) / example

```
<?xml version="1.1"?>
<persons xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://www.cgnm.de/xml/persons.xsd"
         xsi:schemaLocation="http://www.cgnm.de/xml/persons.xsd persons-ns.xsd">
  <person><sn>Doe</sn><fn>John</fn></person>
  <person><fn>Alice</fn><sn>Meier</sn></person>
  <person><fn>Bob</fn><sn>Miller</sn></person>
</persons>
```

Figure 46: Linking a schema to a document (using namespaces).

Qualified Names in XML Schema

XML schema documents use qualified names ($\langle$*QName*$\rangle$s, i.e., names with namespaces) in two different places:

1. for references (e.g., **ref**, **type** attributes),
2. for the schema elements themselves

These names must be associated with a namespace by the ususal XML namespace mechanism, i.e.,

- either using an explicit prefix (e.g., **xs:element**, **xs:string**)
- or declaring a default namespace (with **xmlns="..."**).

Target namespace

Names of declarations and definitions (i.e., **name** attributes) are unqualified names (⟨*NCNames*⟩).

- Names of global declarations and definitions are placed in the **target namespace**.
- Names of local declarations are placed in the
  - target namespace, if **elementFormDefault** or **attributeFormDefault** is **qualified**.
  - empty namespace, otherwise.

```
<schema
 version = ⟨token⟩
 targetNamespace = ⟨anyURI⟩
 elementFormDefault = (qualified | unqualified) : unqualified
 attributeFormDefault = (qualified | unqualified) : unqualified
 >
 Content: ...
</schema>
```

Recommendation: always use **elementFormDefault="qualified"**.

```
<?xml version="1.1"?>
<a xmlns="http://www.cgnm.de/xml/nstest.xsd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.cgnm.de/xml/nstest.xsd nstest.xsd">
 <b>test</b>
</a>
```

Figure 47: XML document using namespaces.

```
<?xml version="1.0"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.cgnm.de/xml/nstest.xsd">
 <xs:element name="a">
  <xs:complexType mixed="true">
   <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="b" type="xs:string"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Figure 48: XML schema with unqualified local names.

```
<?xml version="1.1"?>
<a xmlns="http://www.cgnm.de/xml/nstest.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.cgnm.de/xml/nstest.xsd nstest.xsd">
 <b xmlns="">test</b>
</a>
```

Figure 49: XML document using namespaces for top-level elements only.

```
<?xml version="1.0"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://www.cgnm.de/xml/nstest.xsd">
 <xs:element name="a">
  <xs:complexType mixed="true">
   <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="b" type="xs:string"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Figure 50: XML schema with unqualified local names.

```
<?xml version="1.1"?>
<a xmlns="http://www.cgnm.de/xml/nstest.xsd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.cgnm.de/xml/nstest.xsd nstest.xsd">
 <b>test</b>
</a>
```

Figure 51: XML document using namespaces.

```
<?xml version="1.0"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.cgnm.de/xml/nstest.xsd"
           elementFormDefault="qualified">
  <xs:element name="a">
    <xs:complexType mixed="true">
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:element name="b" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 52: XML schema with qualified local names.

```
<?xml version="1.1"?>
<a xmlns="http://www.cgnm.de/xml/nstest.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.cgnm.de/xml/nstest.xsd nstest.xsd">
 <b>test</b>
</a>
```

Figure 53: XML document using namespaces.

```
<?xml version="1.0"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://www.cgnm.de/xml/nstest.xsd">
 <xs:element name="b" type="xs:string"/>
 <xs:element name="a">
  <xs:complexType mixed="true">
   <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element ref="b"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Figure 54: XML schema with global elements.

```
<?xml version="1.1"?>
<a xmlns="http://www.cgnm.de/xml/nstest.xsd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.cgnm.de/xml/nstest.xsd nstest.xsd">
 <b>test</b>
</a>
```

Figure 55: XML document using namespaces.

```
<?xml version="1.0"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://www.cgnm.de/xml/nstest.xsd"
           targetNamespace="http://www.cgnm.de/xml/nstest.xsd">
  <xs:element name="b" type="xs:string"/>
  <xs:element name="a">
    <xs:complexType mixed="true">
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="b"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 56: XML schema with global elements and default namespace.

Recommendation wrt. namespaces:

- always use namespaces in instance documents.
- always use namespaces in schema documents.
  - use prefix for schema namespace.
  - specify explicit target namespace.
  - use **elementFormDefault="qualified"** (i.e., namespaces for local elements).
  - use target namespace as default namespace.

```
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns="http://www.cgnm.de/xml/nstest.xsd"
        targetNamespace="http://www.cgnm.de/xml/nstest.xsd"
        elementFormDefault="qualified">
```

Figure 57: Recommended usage of namespaces in XML schema.

Including and Importing

```
<include
 schemaLocation = ⟨anyURI⟩
 />
```

include builds a single-namespace schema.

```
<import
 schemaLocation = ⟨anyURI⟩
 namespace = ⟨anyURI⟩
 />
```

import builds a multi-namespace schema.

An included schema must have

- either the same target namespace as the including schema
- or no target namespace
  (in which case it is converted to the target namespace of the including schema).

The imported namespace must be

- different from the target namespace of the importing schema,
- identical to the target namespace of the imported schema.

In neither case can an explicitly stated target namespace be redefined.

```
<?xml version="1.0"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://www.cgnm.de/xml/a.xsd"
           targetNamespace="http://www.cgnm.de/xml/a.xsd"
           elementFormDefault="qualified">
  <xs:include schemaLocation="b.xsd"/>
  <xs:element name="a">
    <xs:complexType mixed="true">
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="b"/>
      </xs:sequence></xs:complexType></xs:element>
</xs:schema>
```

Figure 58: XML schema `a.xsd` with an include.

```
<?xml version="1.0"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://www.cgnm.de/xml/a.xsd"
           targetNamespace="http://www.cgnm.de/xml/a.xsd"
           elementFormDefault="qualified">
  <xs:element name="b" type="xs:string"/>
</xs:schema>
```

Figure 59: XML schema `b.xsd` with namespace "`a.xsd`".

```
<?xml version="1.1"?>
<a xmlns="http://www.cgnm.de/xml/a.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.cgnm.de/xml/a.xsd a-ns.xsd">
 <b>test</b>
</a>
```

Figure 60: An instance document of XML schema `a.xsd`.

```
<?xml version="1.0"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://www.cgnm.de/xml/a.xsd"
           targetNamespace="http://www.cgnm.de/xml/a.xsd"
           elementFormDefault="qualified">
  <xs:include schemaLocation="b.xsd"/>
  <xs:element name="a">
    <xs:complexType mixed="true">
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="b"/>
      </xs:sequence></xs:complexType></xs:element>
</xs:schema>
```

Figure 61: XML schema `a.xsd` with an include.

```
<?xml version="1.0"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="b" type="xs:string"/>
</xs:schema>
```

Figure 62: XML schema `b-nons.xsd` w./o. namespace ("chamaeleon").

```
<?xml version="1.0"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://www.cgnm.de/xml/a.xsd"
           xmlns:b="http://www.cgnm.de/xml/b.xsd"
           targetNamespace="http://www.cgnm.de/xml/a.xsd"
           elementFormDefault="qualified">
 <xs:import schemaLocation="b-b.xsd" namespace="http://www.cgnm.de/xml/b.xsd
 <xs:element name="a">
  <xs:complexType mixed="true">
   <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element ref="b:b"/>
   </xs:sequence></xs:complexType></xs:element></xs:schema>
```

Figure 63: XML schema `a-imp.xsd` with an import.

```
<?xml version="1.0"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://www.cgnm.de/xml/b.xsd"
           targetNamespace="http://www.cgnm.de/xml/b.xsd"
           elementFormDefault="qualified">
 <xs:element name="b" type="xs:string"/>
</xs:schema>
```

Figure 64: XML schema `b-b.xsd` with own namespace.

```
<?xml version="1.1"?>
<a xmlns="http://www.cgnm.de/xml/a.xsd"
  xmlns:b="http://www.cgnm.de/xml/b.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.cgnm.de/xml/a.xsd a-imp.xsd
                http://www.cgnm.de/xml/b.xsd b-b.xsd">
 <b:b>test</b:b>
</a>
```

Figure 65: An instance document of XML schema `a-imp.xsd`.

Integrating Schemata

```
<?xml version="1.1"?>
<article xmlns="http://www.cgnm.de/xml/article-book.xsd"
     xmlns:bk="http://www.cgnm.de/xml/books.xsd"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.cgnm.de/xml/article-book.xsd article-book.xs
                 http://www.cgnm.de/xml/books.xsd books.xsd">
 <title>What others say</title>
 A short overview of basic and most important XML technologies
 is given in
 <bk:book>
  <bk:author><bk:fn>Erik T.</bk:fn><bk:sn>Ray</bk:sn></bk:author>
  <bk:title>Learning XML</bk:title>
  <bk:year edition="2">2003</bk:year>
 </bk:book>
 Also useful is ...
</article>
```

Figure 66: Article document with books part.

```
<?xml version="1.0"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.cgnm.de/xml/article-book.xsd"
           xmlns="http://www.cgnm.de/xml/article-book.xsd"
           elementFormDefault="qualified">
 <xs:element name="article">
  <xs:complexType mixed="true">
   <xs:choice minOccurs="0" maxOccurs="unbounded">
    <xs:element ref="strong"/>
    <xs:element ref="em"/>
    <xs:element name="title">
     <xs:complexType mixed="true">
      <xs:choice minOccurs="0" maxOccurs="unbounded">
       <xs:element ref="strong"/>
       <xs:element ref="em"/>
      </xs:choice></xs:complexType>
    </xs:element>
    <xs:any namespace="http://www.cgnm.de/xml/books.xsd"/>
   </xs:choice>
  </xs:complexType>
```

Figure 67: Modified article schema allowing elements from book schema (excerpt).

II. XML / 4. XML Schema

1. Constraining Document Structure

2. Datatypes

3. Integrity Constraints

4. Schema Modularization

5. Namespaces in XML Schema

6. RELAX NG

7. First Applications

RELAX NG

RELAX NG is another schema language (besides DTDs and XML Schema):

- OASIS Committee Specification (2001/12/03).
- Based on older schema languages
  - RELAX (Regular Language description for XML) and
  - TREX (Tree Regular Expressions for XML).
- Has a XML syntax and a compact non-XML syntax.
- Tools:
  - trang – convert RELAXNG → XML Schema as well as between XML and compact syntax of RELAX NG.
  - jing – validate document against RELAX NG schema.
  - Sun RELAX NG Converter – convert XML Schema → RELAX NG.
  - nxml-mode – emacs-mode for instant validation against RELAX NG.
- More info at http://www.relaxng.org.

RELAX NG Example

```
# books.rnc
start = books
books = element books { book* }
book = element book { author+ & title & year }
author = element author { fn, sn }
fn = element fn { text }
sn = element sn { text }
title = element title { text }
year = element year { text }
```

Figure 68: RELAX NG schema for books: the interleave operator & allows specification of content models as authors, title, and year in any order.

II. XML / 4. XML Schema

1. **Constraining Document Structure**

2. **Datatypes**

3. **Integrity Constraints**

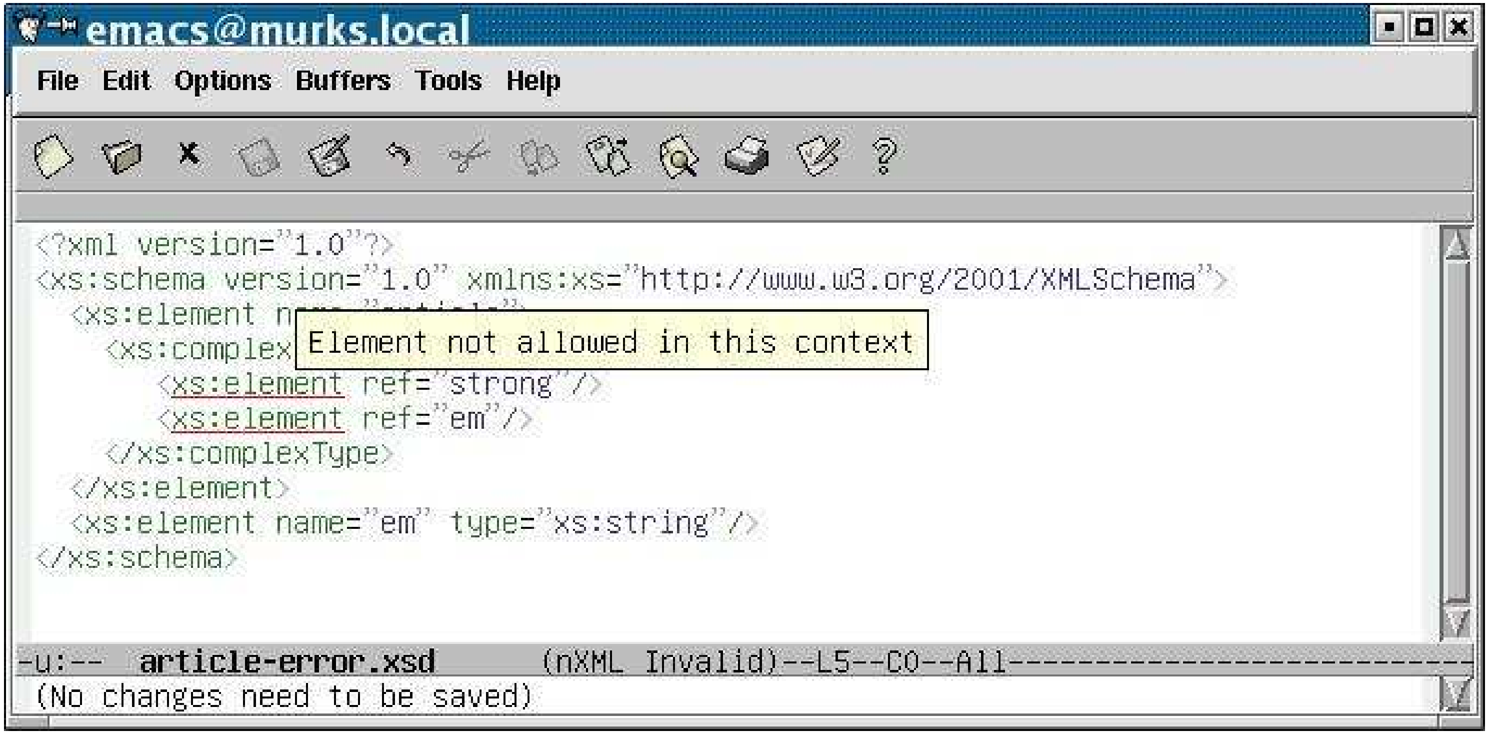
4. **Schema Modularization**

5. **Namespaces in XML Schema**

6. **RELAX NG**

7. **First Applications**

Schema-aware Editor / Document-centric



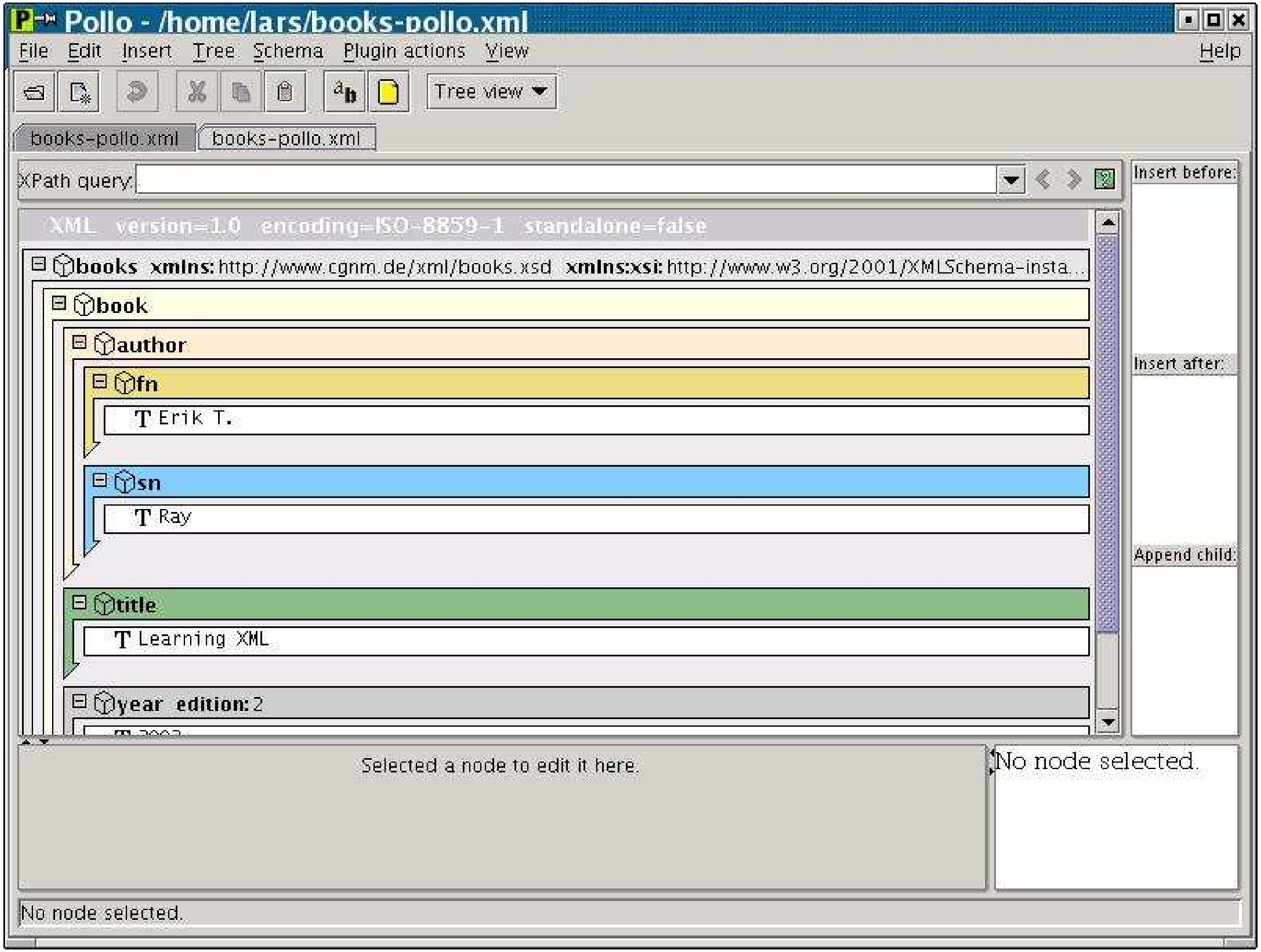
```
<?xml version="1.0"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element n
    <xs:complex
       <xs:element ref="strong"/>
       <xs:element ref="em"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="em" type="xs:string"/>
</xs:schema>
```

Element not allowed in this context

```
-u:--  article-error.xsd       (nXML Invalid)--L5--C0--All-------------------------
(No changes need to be saved)
```

Figure 69: Emacs with nxml-mode.

Schema-aware Editor / Tree View



XML version=1.0 encoding=ISO-8859-1 standalone=false

books xmlns: http://www.cgnm.de/xml/books.xsd xmlns:xsi: http://www.w3.org/2001/XMLSchema-insta...

book

author

fn

T Erik T.

sn

T Ray

title

T Learning XML

year edition: 2

T 2003

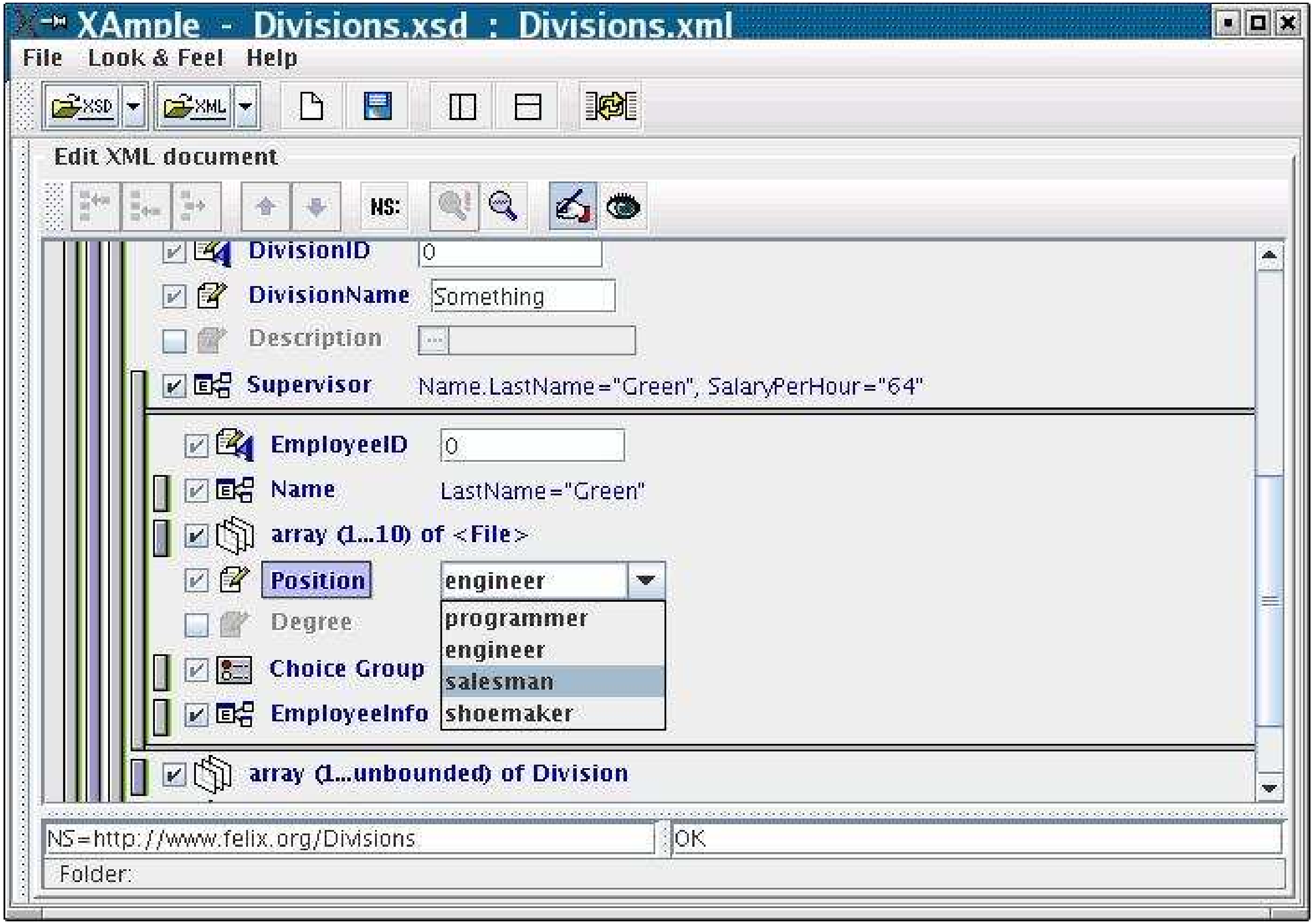
Selected a node to edit it here.

No node selected.

No node selected.

Figure 70: Pollo XML editor with tree view.

Schema-aware Editor / Schema-specific GUI



XAmple - Divisions.xsd : Divisions.xml

Edit XML document

☑ DivisionID 0
☑ DivisionName Something
□ Description
☑ Supervisor Name.LastName="Green", SalaryPerHour="64"
☑ EmployeeID 0
☑ Name LastName="Green"
☑ array (1...10) of <File>
☑ Position engineer
□ Degree
☑ Choice Group
☑ EmployeeInfo

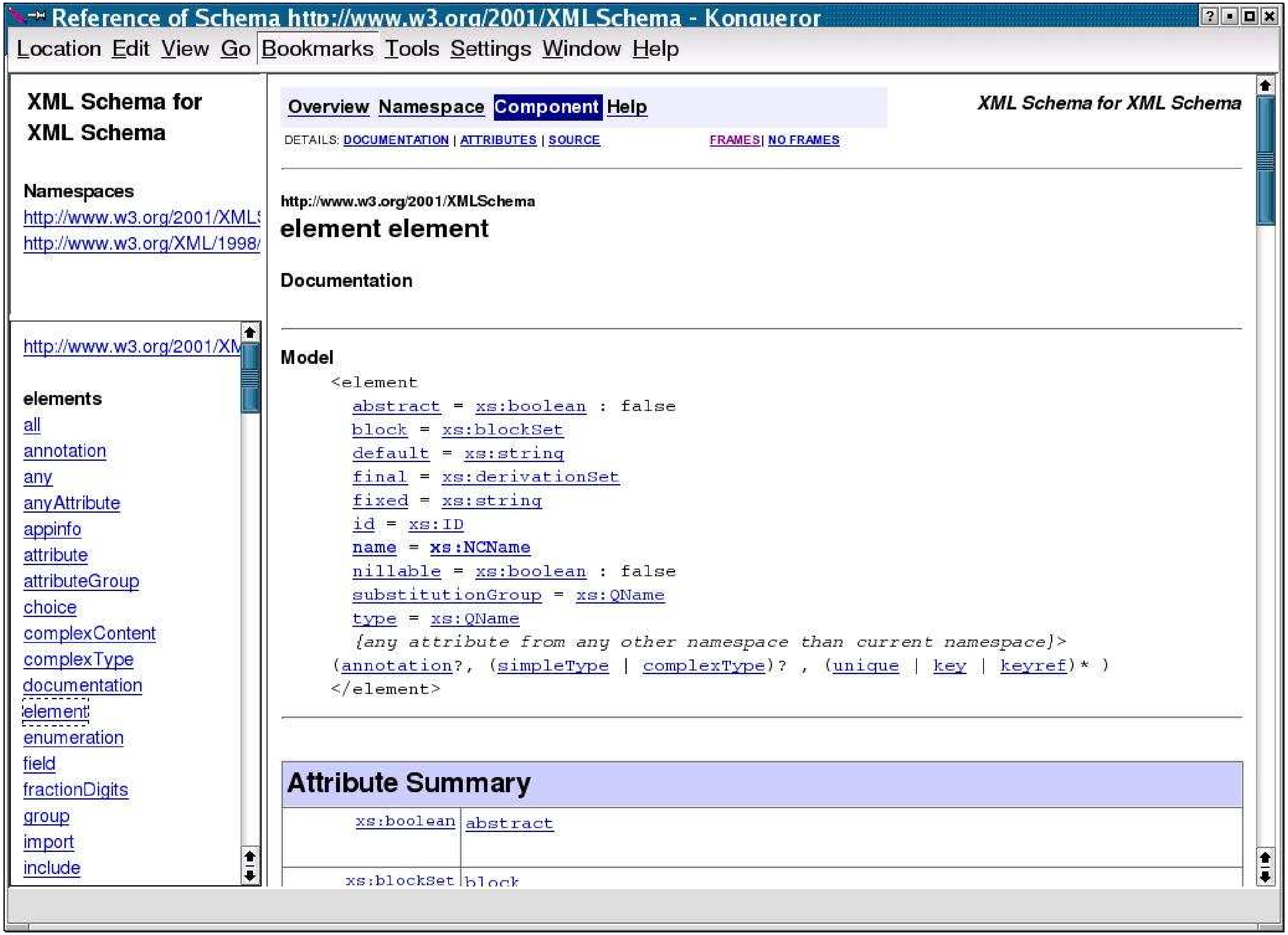
programmer
engineer
salesman
shoemaker

☑ array (1...unbounded) of Division

NS=http://www.felix.org/Divisions
OK
Folder:

Figure 71: XAmple schema-specific GUI creator.

XML Schema Documentation Generator


Reference of Schema http://www.w3.org/2001/XMLSchema - Konqueror
Location Edit View Go Bookmarks Tools Settings Window Help
Overview Namespace Component Help
DETAILS: DOCUMENTATION | ATTRIBUTES | SOURCE
FRAMES | NO FRAMES

XML Schema for XML Schema

Namespaces

http://www.w3.org/2001/XMLS
http://www.w3.org/XML/1998/

http://www.w3.org/2001/XM

elements
- all
- annotation
- any
- anyAttribute
- appinfo
- attribute
- attributeGroup
- choice
- complexContent
- complexType
- documentation
- element
- enumeration
- field
- fractionDigits
- group
- import
- include

XML Schema for XML Schema

http://www.w3.org/2001/XMLSchema

element element

Documentation

Model

```
<element
  abstract = xs:boolean : false
  block = xs:blockSet
  default = xs:string
  final = xs:derivationSet
  fixed = xs:string
  id = xs:ID
  name = xs:NCName
  nillable = xs:boolean : false
  substitutionGroup = xs:QName
  type = xs:QName
  {any attribute from any other namespace than current namespace}>
(annotation?, (simpleType | complexType)? , (unique | key | keyref)* )
</element>
```

Attribute Summary

xs:boolean	abstract
xs:blockSet	block

Figure 72: xsddoc generated documentation for XML Schema element **element**.

Summary

- XML Schema is the industry standard for defining XML document schemata.
- XML Schema maps element and attribute names to types.
- XML Schema distinguishes between **simple types** (for attributes and elements) and **complex types** (only for elements).
- XML Schema provides most elementary datatypes such as ints, floats, strings etc. as built-in simple types.
- XML Schema allows to define complex types using model groups (sequence, choice, all, wildcards) and attributes.
- XML Schema allows to derive simple and complex types from other simple and complex types using extension and restriction mechanisms, esp. facets for attributes.
- XML Schema supports multi-namespace documents.

References