Information Systems 2

2. Modelling Information Systems II: XML

Lars Schmidt-Thieme

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

What is XML?

XML is . . .

- . . . the extensible markup language.
- . . . an industry standard for document and data interchange languages.
- . . . facilitating the separation of content from presentation.
- . . . (from a perspective of HTML) allowing the definition of own tags.
- . . . (from a perspective of SGML) a subset of SGML.
- . . . a W3C recommendation since 1998.

XML Technologies

technology	goal
XML	defines how to encode structured documents and data
XML Schema	defines how to describe a vocabulary and the structure (the schema) of a class of XML documents (there are alternatives such as RelaxNG)
XQuery	defines a query language to retrieve some specific piece of a XML document or some aggregation etc. (there are alternatives such as XSLT)

A First Example

```
1 <?xml version="1.1"?>
2 <contacts>
3   <contact>
4     <name>Anna Müller</name>
5     <address>Schuhstaße 3, 31139 Hildesheim</address>
6     <phone area="05121">123456</phone>
7     <email>mueller@example.com</email>
8   </contact>
9   <contact>
10    <name>Bert Meier</name>
11    <address>Hauptstraße 11, 30300 Hannover</address>
12    <phone area="050">12480</phone>
13    <email>meier@beispiel.de</email>
14  </contact>
15 </contacts>
```

Figure 1: A simple example XML document.

1. XML Syntax

2. XML Schema

3. XPath

4. XQuery

W3C development process

W3C specifications are called **Recommendations**.

Stages of W3C recommendations:

stage	completion date XML 1.0	XML 1.1
Working Draft	1996/11/14	2001/12/13
	1997/11/17	
Last Call Working Draft		2002/04/25
Candidate Recommendation		2002/10/15
Proposed Recommendation	1997/12/08	2003/11/05
Recommendation	1998/02/10	2004/04/15
Working Draft	2000/08/14	
Recommendation (2nd edition)	2000/10/06	2006/08/16
Proposed Edited Recommendation	2003/10/30	
Recommendation (3rd edition)	2004/02/04	
Recommendation (4th edition)	2006/08/16	

Every XML document consists of a **prolog** and a single element, called **root element**.

⟨*document*⟩ := ⟨*prolog*⟩ ⟨*element*⟩ (⟨*Comment*⟩ | ⟨*PI*⟩ | ⟨*S*⟩)*

⟨*prolog*⟩ := `<?xml` ⟨*S*⟩ `version` = `"1.1"`
(⟨*S*⟩ `encoding` = ⟨*encoding*⟩)?
(⟨*S*⟩ `standalone` = (`"yes"` | `"no"`))?
⟨*S*⟩? `?>`
(⟨*Comment*⟩ | ⟨*PI*⟩ | ⟨*S*⟩)*
(⟨*DoctypeDecl*⟩ (⟨*Comment*⟩ | ⟨*PI*⟩ | ⟨*S*⟩)*)?

In all productions

- matching `"` can be replaced by `'`.
- `=` may be surrounded by spaces (i.e., match ⟨*S*⟩? `=` ⟨*S*⟩?).

⟨*S*⟩ := (#x20 | #x9 | #xD | #xA)+

A minimal XML document

```
1 <?xml version="1.1"?>
2 <page/>
```

Figure 2: A minimal XML document with root element "page".

In XML 1.1 the version attribute is mandatory.

If the version attribute is missing, version 1.0 is assumed.

Elements and Attributes

⟨*element*⟩ := ⟨*emptyElementTag*⟩
| ⟨*STag*⟩ ⟨*content*⟩ ⟨*ETag*⟩

⟨*emptyElementTag*⟩ := < ⟨*Name*⟩ (⟨*S*⟩ ⟨*Name*⟩ = " ⟨*AttValue*⟩ ")* ⟨*S*⟩? />

⟨*STag*⟩ := < ⟨*Name*⟩ (⟨*S*⟩ ⟨*Name*⟩ = " ⟨*AttValue*⟩ ")* ⟨*S*⟩? >

⟨*ETag*⟩ := </ ⟨*Name*⟩ ⟨*S*⟩? >

⟨*Name*⟩s

- start with a unicode letter or _
  (: is also allowed, but used for namespaces).
- may contain unicode letters, uncode digits, -, ., or ·.

A wellformed document requires,

- that start and end tag of each element match,
- that for each tag the same attribute never occurs twice.

Not-wellformed Documents (1/2)

```
1 <?xml version="1.1"?>
2 <book>
3   <author><fn>Rainer</fn><sn>Eckstein</sn></author>
4   <author><fn>Silke</fn><sn>Eckstein</sn></author>
5   <title>XML und Datenmodellierung</title>
6   <year>2004</year>
7 </book>
8 <book>
9   <author><fn>Erik T.</fn><sn>Ray</sn></author>
10  <title>Learning XML</title>
11  <year edition="2">2003</year>
12 </book>
```

Figure 3:

Not-wellformed Documents (2/2)

```
1 <?xml version="1.1"?>
2 <book>
3   <author><fn>Erik T.</fn><sn>Ray</author></sn>
4   <title>Learning XML</title>
5   <year edition="2">2003</year>
6 </book>
```

Figure 4:

```
1 <?xml version="1.1"?>
2 <book author="Rainer Eckstein" author="Silke Eckstein">
3   <title>XML und Datenmodellierung</title>
4   <year>2004</year>
5 </book>
```

Figure 5:

Element content

The contents of an element can be made up from 6 different things:

1. other elements,
2. Character data,
3. References,
4. CDATA sections,
5. Processing instructions, and
6. comments.

$$\langle content \rangle := \langle CharData \rangle? \; (\; (\; \langle element \rangle \mid \langle Reference \rangle \mid \langle CDSect \rangle \mid \langle PI \rangle \mid \langle Comment \rangle \;) \; \langle CharData \rangle? \;)^*$$

Character data

⟨*CharData*⟩ may contain any characters except

`<`, `&`, or the sequence `>]]`

Attribute values may not contain

- `"`, if delimited by `"`,
- `'`, if delimited by `'`,

These characters can be expressed by references.

Character data

```
1 <?xml version="1.1"?>
2 <abstract>
3   x^2 = y has no real solution for y < 0.
4   But there are solutions for y = 0 & for y > 0.
5 </abstract>
```

Figure 6: Forbidden characters in character data.

```
1 <?xml version="1.1"?>
2 <abstract>
3   x^2 = y has no real solution for y &lt; 0.
4   But there are solutions for y = 0 &amp; for y &gt; 0.
5 </abstract>
```

Figure 7: Using references in character data.

References

⟨*Reference*⟩ := ⟨*EntityRef*⟩ | ⟨*CharRef*⟩

⟨*CharRef*⟩ := `&#` [0-9]+ `;`
| `&#x` [0-9a-fA-F]+ `;`

⟨*EntityRef*⟩ := `&` ⟨*Name*⟩ `;`

There are five predefined entity references:

`<`	`>`	`&`	`&apos;`	`&quot;`
<	>	&	’	"

All other entities known from HTML (as `&auml;`) are **not** predefined in XML.

Custom entities can be defined in the document type declaration.

Attribute values

```
1 <?xml version="1.1"?>
2 <book abstract="Discusses meaning of "wellformed"">
3   <author>John Doe</author>
4   <title>About wellformedness</title>
5 </book>
```

Figure 8: Literal usage of attribute delimiter.

```
1 <?xml version="1.1"?>
2 <book abstract=’Discusses meaning of "wellformed"’>
3   <author>John Doe</author>
4   <title>About wellformedness</title>
5 </book>
```

Figure 9: Using different attribute delimiters.

```
1 <?xml version="1.1"?>
2 <book abstract="Discusses meaning of &quot;wellformed&quot;">
3   <author>John Doe</author>
4   <title>About wellformedness</title>
5 </book>
```

Figure 10: Using references in attribute values.

CDATA sections

CDATA sections allow the literal usage of all characters (except the sequence `]]>`).

⟨*CDSect*⟩ := `<![CDATA[` ⟨*CData*⟩ `]]>`

CDATA sections are typically used for longer text containing `<` or `&`.

CDATA sections are flat, i.e., there is no possibility to structure them with elements (as `<` or `&` are interpreted literally).

Character data and CDATA sections

```
1 <?xml version="1.1"?>
2 <abstract>
3   x^2 = y has no real solution for y &#3c; 0.
4   But there are solutions for y = 0 &#26; for y &#3e; 0.
5 </abstract>
```

Figure 11: Using numeric character references.

```
1 <?xml version="1.1"?>
2 <abstract><![CDATA[
3   x^2 = y has no real solution for y < 0.
4   But there are solutions for y = 0 & for y > 0.
5 ]]></abstract>
```

Figure 12: Using a CDATA-section.

Comments & Processing Instructions

⟨*Comment*⟩ := <!-- ⟨*Char*⟩* -->

⟨*PI*⟩ := <? ⟨*Name*⟩ (⟨*S*⟩ ⟨*Char*⟩*)? ?>

Comments are not allowed to contain the character sequence --.

Processing instructions (PIs) allow documents to contain instructions for applications.

```
1 <?xml version="1.1"?>
2 <!-- list is not complete yet ! -->
3 <books>
4   <!-- yet to be ordered -->
5   <book>
6     <author><fn>Rainer</fn><sn>Eckstein</sn></author>
7     <author><fn>Silke</fn><sn>Eckstein</sn></author>
8     <title>XML und Datenmodellierung</title>
9     <year><!-- look up year of publication --></year>
10  </book>
11 </books>
12
```

Figure 13: Comments in the prolog and in the contents of elements.

XML Namespaces

For "mixing vocabularies" XML Namespaces have been designed. They provide mechanisms for

- marking elements and attributes with namespaces and
- validating documents with elements and attributes from different namespaces (mostly in conjunction with XML Schema)

version: Namespaces in XML 1.1 (W3C Recommendation, 2004/02/04)

A namespace is identified by an (absolute) IRI reference.

Expanded name: pair of

- namespace IRI (**namespace name**) and
- **local name**.

XML Namespaces / Declaration of Namespace Prefixes

Namespace attribute to declare namespace prefixes:

⟨*NamespaceAtt*⟩ := (`xmlns` | `xmlns :` ⟨*NCName*⟩) `=` `"` ⟨*IRI*⟩ `"`

⟨*NCName*⟩ = non-colonized name (i.e., without ":"s).

Scope: element it is attribute of.

Without prefix defines **default namespace**.

Implicitly declared prefixes:

- xml: `http://www.w3.org/XML/1998/namespace`
- xmlns: `http://www.w3.org/2000/xmlns/`

XML Namespaces / Namespace Usage

Qualified name (⟨*QName*⟩): name subject to namespace interpretation (maybe prefixed, maybe unprefixed).

⟨*QName*⟩ := NCName | (⟨*NamespacePrefix*⟩ `:` ⟨*NCName*⟩)

A prefix associates the name of an element or attribute with a namespace.

Default namespace applies

- to the element it is attribute of (if it is unprefixed) and
- to all nested elements (unless they are prefixed or the default namespace is overwritten).
- but not to unprefixed attributes.

XML Namespaces / Example

```
1 <?xml version="1.1"?>
2 <article xmlns="http://www.cgnm.de/xml/article.dtd"
3         xmlns:bk="http://www.cgnm.de/xml/books.dtd">
4   <title>What others say</title>
5   A short overview of basic and most important XML technologies
6   is given in
7   <bk:book>
8     <bk:author><bk:fn>Erik T.</bk:fn><bk:sn>Ray</bk:sn></bk:author>
9     <bk:title>Learning XML</bk:title>
10    <bk:year edition="2">2003</bk:year>
11  </bk:book>
12  Also useful is ...
13 </article>
```

Figure 14: Namespaces are used to differentiate elements from different DTDs (default namespace and prefix).

1. XML Syntax

2. XML Schema

3. XPath

4. XQuery

XML Schema

There are several standards to define schemata for XML documents:

- **Document Type Definitions** (**DTD**s):
  - old standard, usable for general SGML
  - very modest expressivity
  - specific grammar
- **XML Schema**:
  - standard specific for XML
  - rich expressivity
  - XML grammar
- **RelaxNG** and other alternative standards: more or less XML Schema compatible.

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 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:

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

Figure 15: 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.

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

Figure 16: Linking a schema to a document.

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

xerces -v -s persons-empty.xml

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.

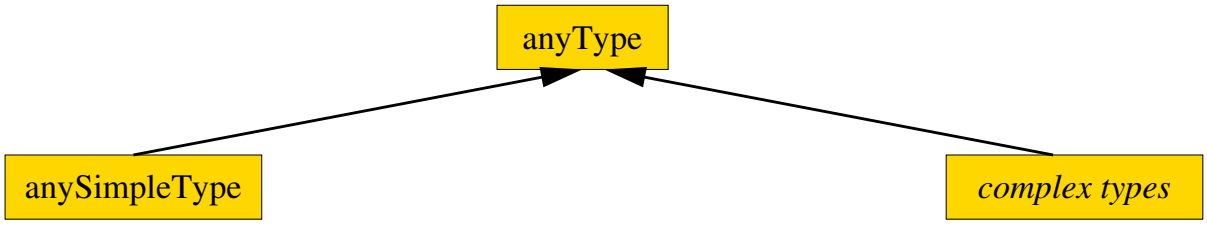

Figure 17: XML Schema Type hierarchy (top-level).

Simple types:

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

Complex types: rich description of

- attributes and
- element contents.

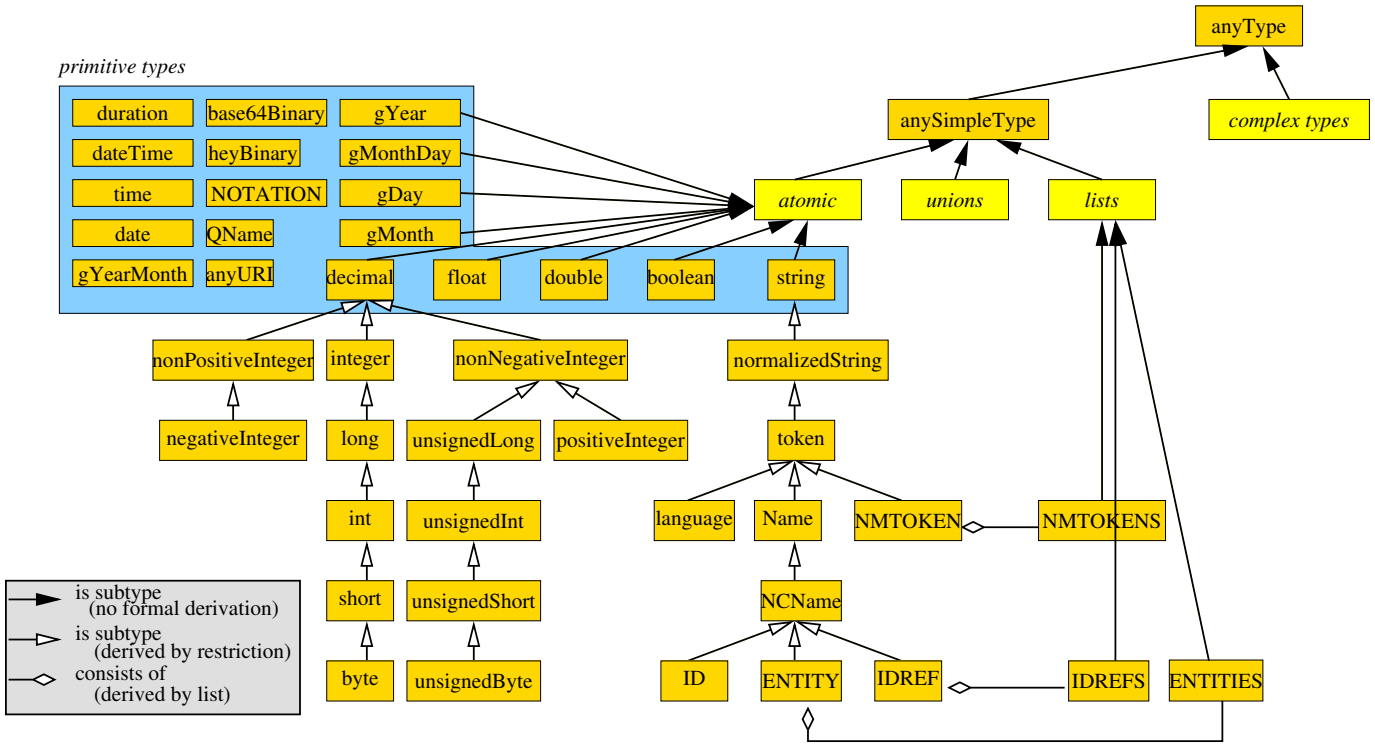

Figure 18: XML Schema built-in datatypes.

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

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

Figure 19: Example document.

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

Figure 20: Minimal schema `persons-minimal.xsd` s.t. the example document is valid w.r.t. that schema.

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

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

Figure 21: Example document (valid).

```
1 <?xml version="1.0"?>
2 <xs:schema version="1.0"
3   xmlns:xs="http://www.w3.org/2001/XMLSchema">
4   <xs:element name="persons">
5     <xs:complexType mixed="true"/>
6   </xs:element>
7 </xs:schema>
```

Figure 22: Schema with nested type.

```
1 <?xml version="1.0"?>
2 <xs:schema version="1.0"
3   xmlns:xs="http://www.w3.org/2001/XMLSchema">
4   <xs:element name="persons"
5           type="personsType"/>
6
7   <xs:complexType name="personsType"
8           mixed="true"/>
9 </xs:schema>
```

Figure 23: 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 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 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 26: 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 27: 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

```
1 <?xml version="1.1"?>
2 <article xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3         xsi:noNamespaceSchemaLocation="article.xsd">
4   <title>What <em>others</em> say</title>
5   A <strong>short overview</strong> of basic and
6   most important XML technologies is given in ...
7
8   <em>Also</em> useful is ...
9 </article>
```

Figure 28: 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 29: Schema with choice model group.

Model Groups / Other

Beneath **sequence** and **choice**, there are some further model groups:

- **Collections** (`all`):
  – each member must occur in arbitrary order
- **Element wildcards** (`any`):
  – any element from a specified schema may occur

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
  />


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

Figure 30: A Sample Document.

```
6       <xs:element name="book">
7         <xs:complexType>
8           <xs:sequence>
9             <xs:element name="author" minOccurs="1" maxOccurs="unbounded" type
10            <xs:element name="title" type="xs:string"/>
11          </xs:sequence>
12          <xs:attribute name="year" type="xs:gYear"/>
13          <xs:attribute name="isbn" type="xs:string"/>
14        </xs:complexType>
15      </xs:element>
```

Figure 31: Schema with attributes (excerpt).

Integrity Constraints / 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) who's values make the key.

Integrity Constraints / 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*⟩ : *

Integrity Constraints / 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.

Integrity Constraints / 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 who's values make the key reference.

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

Figure 32: 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>
  <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>
```

Figure 33: XML schema defining identity constraints.

1. XML Syntax

2. XML Schema

3. XPath

4. XQuery

XPath Specification

XML Path Language is an expression language for XSLT & XQuery consisting of

1. XQuery 1.0 and XPath 2.0 Data Model (Rec-2007/01/23),
2. XML Path Language (XPath) 2.0 (Rec-2007/01/23),
3. XQuery 1.0 and XPath 2.0 Functions and Operators (Rec-2007/01/23)

as well as further documents (Formal Semantics, Requirements, Use Cases, etc.).

XPath 2.0 is a superset of XPath 1.0 (REC-1999/11/16) that improves by

- using (node) sequences instead of node sets,
- exploiting type information available through XML Schema,
- adding some powerful language constructs (e.g., if- and for-expressions).

XPath 2.0 is implemented, e.g., in Saxon (but not yet in Xalan).

Axis Steps / Node Tests

⟨*PathExpr*⟩ := (/ ⟨*RelativePathExpr*⟩?) | ⟨*RelativePathExpr*⟩

⟨*RelativePathExpr*⟩ := ⟨*StepExpr*⟩ (/ ⟨*StepExpr*⟩)*

⟨*StepExpr*⟩ := ⟨*Axis*⟩ :: ⟨*NodeTest*⟩ ⟨*Predicates*⟩ /* axis step */
| ⟨*PrimaryExpr*⟩ ⟨*Predicates*⟩ /* filter step */

⟨*Axis*⟩ := `self`
| `child` | `descendant` | `descendant-or-self`
| `following-sibling` | `following`
| `parent` | `ancestor` | `ancestor-or-self`
| `preceding-sibling` | `preceding`
| `attribute`

⟨*NodeTest*⟩ := ⟨*QName*⟩ | * | (⟨*NCName*⟩ : *) | (* : ⟨*NCName*⟩)
| ⟨*KindTest*⟩

⟨*Predicates*⟩ := ([⟨*Expr*⟩])*

Axis Steps / Axes

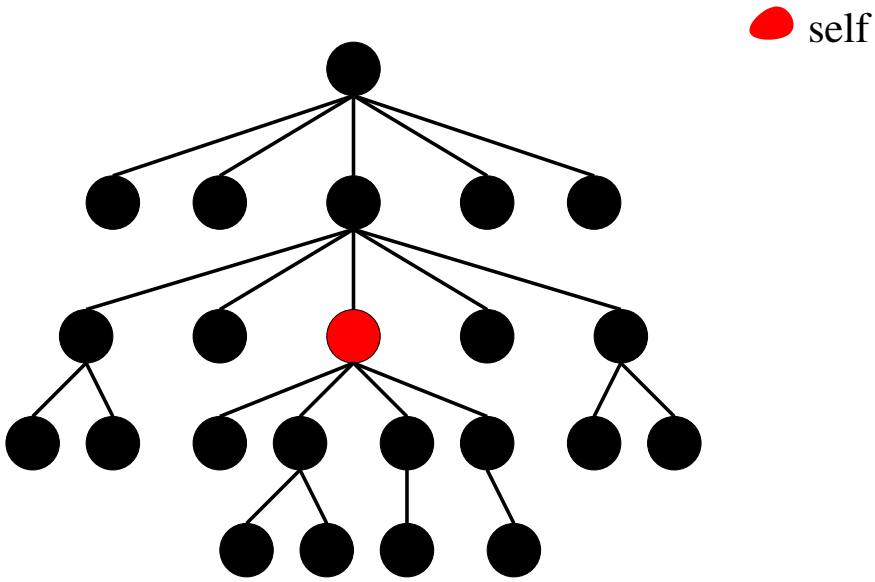

Figure 34: Self axis.

Axis Steps / Axes

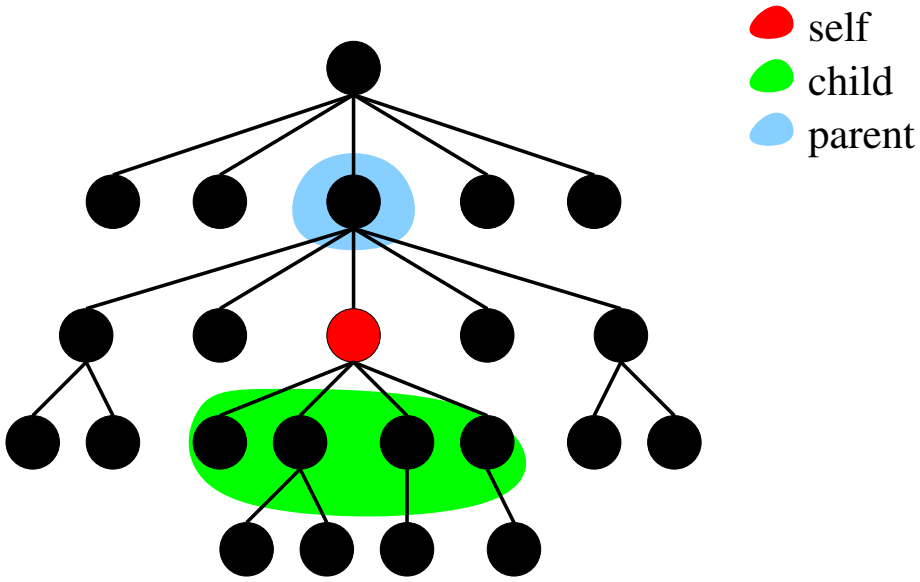

Figure 35: Child and parent axis.

Axis Steps / Axes

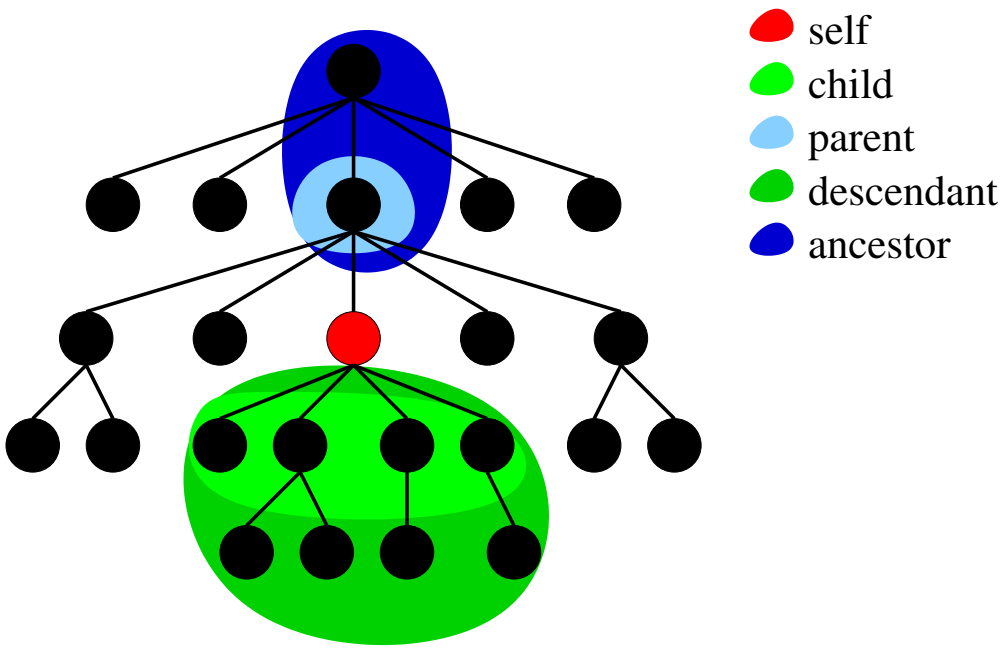

Figure 35: Descendant and ancestor axis.

Axis Steps / Axes

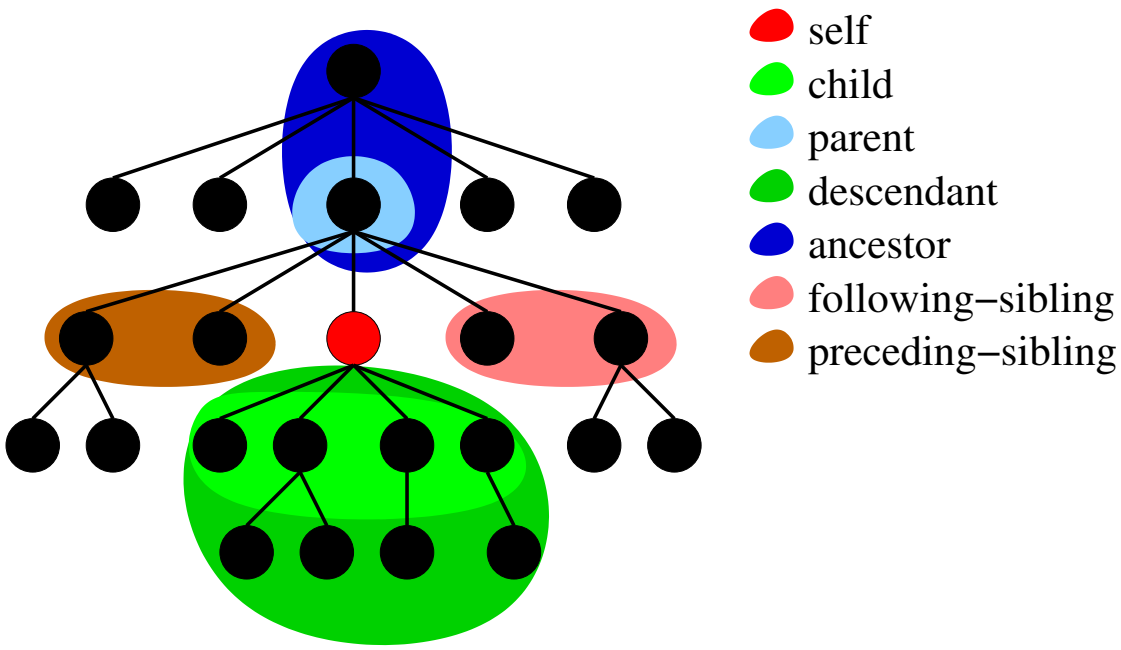

Figure 35: Following-sibling and preceding-sibling axis.

Axis Steps / Axes

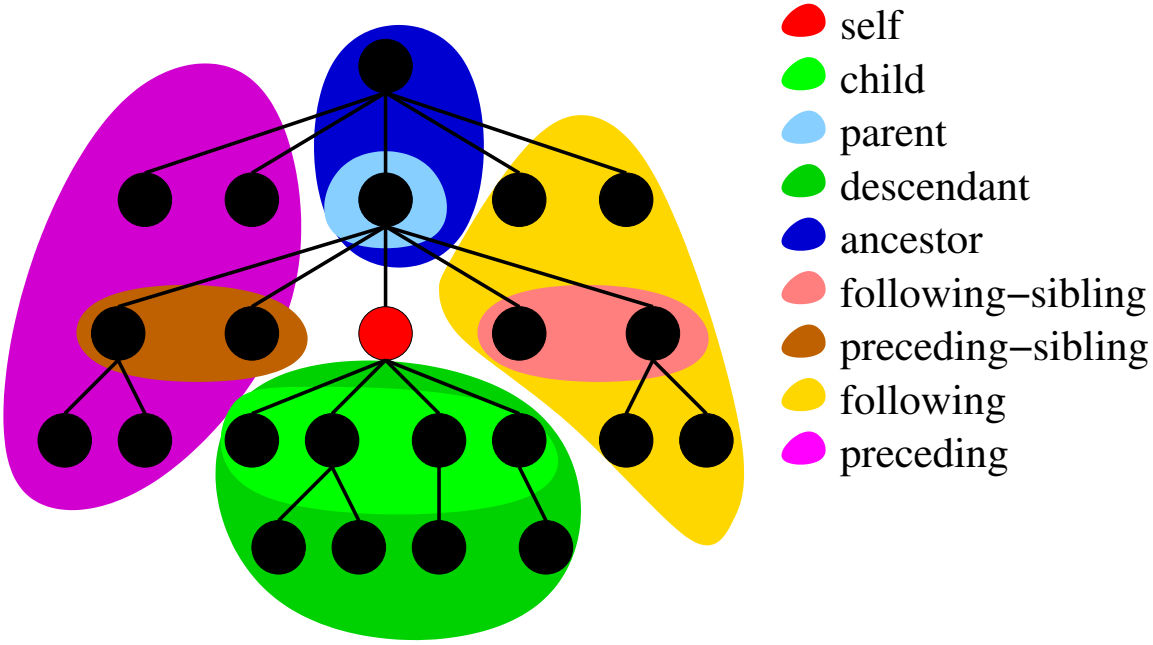

Figure 35: Following and preceding axis.

Axis Steps / Node Tests

Absolute path expressions start with the document node as **context node**, for **relative path expressions** the context node is set by the host language.

Step expressions successively shift the context node.

Axis selects a sequence of nodes relative to the context node ("scope").

Node tests allow to choose a subsequence of these nodes by tests on names or types / kinds.

Predicates allow more complex choices of subsequences of these nodes.

Sequences of nodes are always in document order.
Context positions are assigned starting from 1

- in document order for forward axes and
- in reverse document order for reverse axes.

Axis Steps / Node Tests / Example

```
1 <?xml version="1.1"?>
2 <books>
3   <book>
4     <author>R.E.</author><author>S.E.</author>
5     <title>XML und DM</title></book>
6   <book>
7     <author>E.R.</author><title>Learning XML</title></book>
8   <book>
9     <author>N.W.</author><author>L.M.</author>
10    <title>DocBook</title></book>
11 </books>
```

Figure 35: An abreviated books document `books-short.xml`.

Axis Steps / Node Tests / Example

Query: /descendant-or-self::title

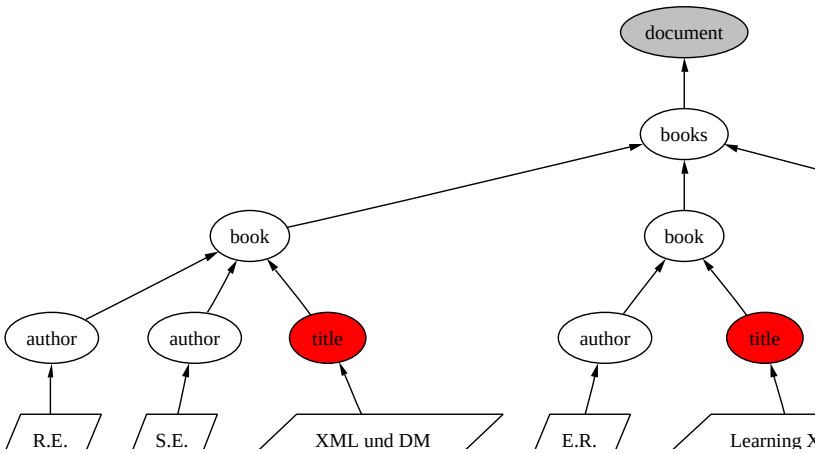

Figure 36: Result of XPath query /descendant-or-self::title.

Axis Steps / Node Tests / Example

Query: /descendant-or-self::title[contains(string(.),"XML")]

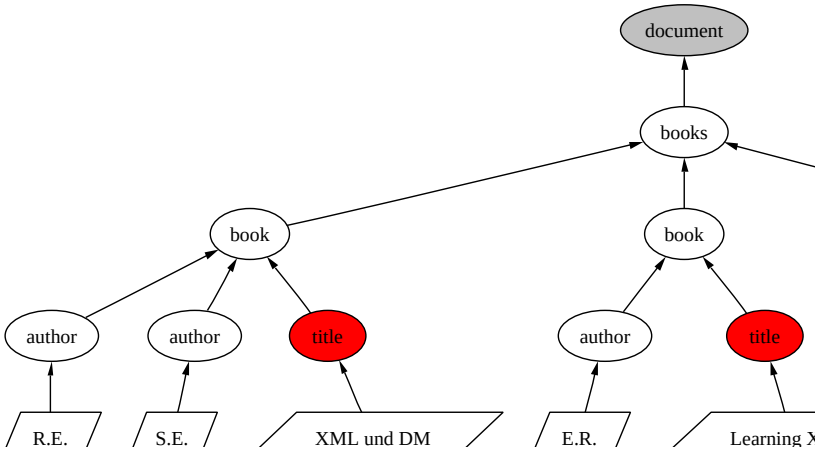

Figure 37: Result of XPath query /descendant-or-self::title[contains(string(.),"XML")].

Axis Steps / Node Tests / Example

Query: /descendant-or-self::title[contains(string(.),"XML")]/parent::node()

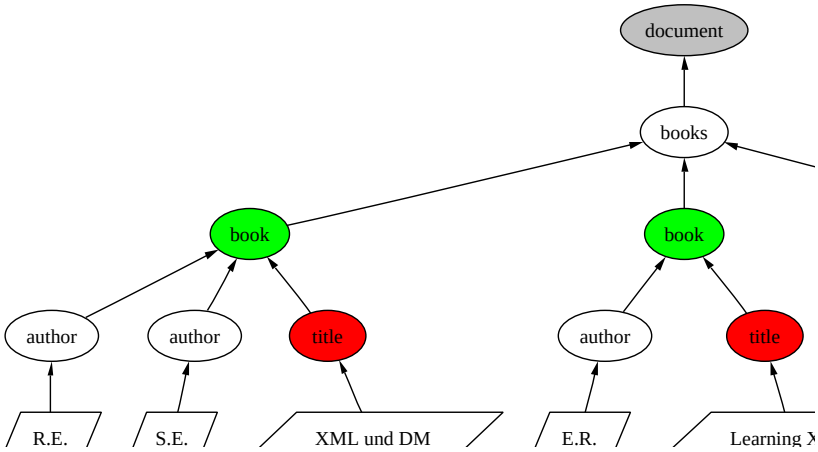

Figure 38: Result of XPath query /descendant-or-self::title[contains(string(.),"XML")]/parent::node().

Axis Steps / Node Tests / Example

Query:
/descendant-or-self::title[contains(string(.),"XML")]/parent::node()/child::author

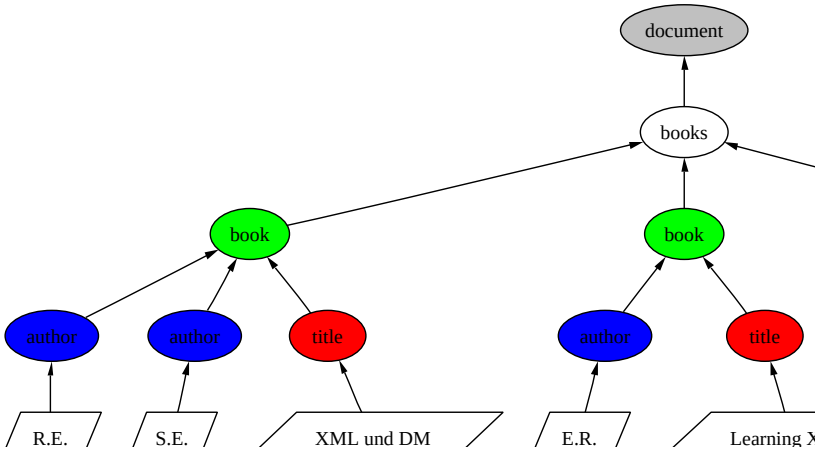

Figure 39: Result of XPath query
/descendant-or-self::title[contains(string(.),"XML")]/parent::node()/child::author.

Axis Steps / Abbreviated Syntax

abbreviation	meaning
no axis name e.g., section/para	child:: axis child::section/child::para
@ as axis name e.g., section/@no	attribute:: axis child::section/attribute::no
// e.g., section//para	/descendant-or-self::node()/ child::section/descendant-or-self::node()/child::para
.. e.g., ../section	parent::node() parent::node()/child::section
[number] e.g., section[1]	[position()=number] section[position()=1]

/descendant-or-self::title[contains(string(.),"XML")]/parent::node()/
child::author[position()=1]

can be written more compactly as

//title[contains(string(.),"XML")]/../author[1]

Performing XPath Queries by Saxon

XPath queries can be performed, e.g., by Saxon.

```
1 /descendant-or-self::title[contains(string(.),"XML")]/parent::node()/child::author
```

Figure 40: File `books.xpath` containing an XPath query.

call (with saxon.jar in classpath):

```
java net.sf.saxon.Query -s books-short.xml books.xpath
```

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <author>R.E.</author>
3 <?xml version="1.0" encoding="UTF-8"?>
4 <author>S.E.</author>
5 <?xml version="1.0" encoding="UTF-8"?>
6 <author>E.R.</author>
```

Figure 41: Result of the XPath query above.

1. XML Syntax

2. XML Schema

3. XPath

4. XQuery

XQuery Specification

XQuery is specified in

1. XQuery 1.0: An XML Query Language (Rec 2007/01/23) and
2. XML Syntax for XQuery 1.0 (XQueryX; (Rec 2007/01/23)

as well as documents about requirements, use cases, serialization, and formal semantics.

XQuery **extends** XPath 2.0, i.e., (most) any XPath expressions **are** XQuery "queries".

XQuery does not have an XML Syntax
(like XPath, but contrary to XSLT).

XQuery Modules

The XQuery processing unit is the **module**:

⟨*Module*⟩ := (`xquery version` ⟨*StringLiteral*⟩ `;`)?
(⟨*ModuleDecl*⟩ `;`)?
⟨*Prolog*⟩
⟨*Expr*⟩

Usually one module is stored in one file.

Library modules have a module declaration, but no body expression;
main modules have a body expression, but no module declaration.

All XPath expressions are XQuery expressions ⟨*Expr*⟩.

```
1 xquery version "1.0" ;
2 //title[contains(string(.),"XML")]/../author
```

Figure 42: Example XQuery consisting of an XPath expression.

FLWOR expressions / Clauses

⟨*FLWORExpr*⟩ := (⟨*ForClause*⟩ | ⟨*LetClause*⟩)+
`where` ⟨*ExprSingle*⟩)? ⟨*OrderByClause*⟩?
`return` ⟨*Expr*⟩

⟨*ForClause*⟩ := `for`
`$` ⟨*QName*⟩ (`as` ⟨*SequenceType*⟩)? (`at $` ⟨*QName*⟩)? `in` ⟨*Expr*⟩
(`,` `$` ⟨*QName*⟩ (`as` ⟨*SequenceType*⟩)? (`at $` ⟨*QName*⟩)? `in` ⟨*Expr*⟩)*

⟨*LetClause*⟩ := `let $` ⟨*QName*⟩ (`as` ⟨*SequenceType*⟩)? `:=` ⟨*Expr*⟩
(`,` `$` ⟨*QName*⟩ (`as` ⟨*SequenceType*⟩)? `:=` ⟨*Expr*⟩)*

XPath's for-expressions is a special case of an XQuery FLWOR expression.
New is:

- `let` binds additional variables,
- `where` filters tuples,
- `order by` orders tuples,
- `at $` ⟨*QName*⟩ binds an additional positional variable,
- `as` ⟨*SequenceType*⟩ types the for-/let-variable.

FLWOR expressions / `for` and `let` Clauses

```
1 xquery version "1.0" ;
2 for $s in (<one/>, <two/>, <three/>)
3   return <out>{$s}</out>
```

Figure 43: FLWOR-expression using `for`.

```
1 xquery version "1.0" ;
2 let $s := (<one/>, <two/>, <three/>)
3   return <out>{$s}</out>
```

Figure 44: FLWOR-expression using `let`.

```
1 <?xml version="1.0"?>
2 <out>
3   <one/>
4 </out>
5 <out>
6   <two/>
7 </out>
8 <out>
9   <three/>
10 </out>
```

Figure 45: Result of `for`-expression.

```
1 <?xml version="1.0"?>
2 <out>
3   <one/>
4   <two/>
5   <three/>
6 </out>
```

Figure 46: Result of `let`-expression.

FLWOR expressions / `where` Clause

```
xquery version "1.0" ;
let $inputvalues := 1 to 1000 return
  avg(for $x at $i in $inputvalues
      where $i mod 100 = 0
      return $x)
```

Figure 47: FLWOR-expression using `where`.

```
xquery version "1.0" ;
let $inputvalues := 1 to 1000 return
  avg($inputvalues[position() mod 100 = 0])
```

Figure 48: Same query using a predicate.

```
550
```

Figure 49: Result of the queries.

FLWOR expressions / `order by` Clause

⟨*OrderByClause*⟩ := (`order by` | `stable order by`) ⟨*OrderSpecList*⟩

⟨*OrderSpecList*⟩ := ⟨*OrderSpec*⟩ (`,` ⟨*OrderSpec*⟩)*

⟨*OrderSpec*⟩ := ⟨*Expr*⟩ ⟨*OrderModifier*⟩

⟨*OrderModifier*⟩ := (`ascending` | `descending`)?
(`empty greatest` | `empty least`)?

```
xquery version "1.0" ;
<html><body>
  Authors: <ol>
  { for $a in distinct-values(//author)
    let $sn := substring-after($a, ' '),
        $fn := substring-before($a, ' ')
    order by $sn, $fn
    return <li>{ concat($sn, ", ", $fn) }</li>
  }
  </ol></body></html>
```

Figure 50: FLWOR-expression with `order by` clause.

```
<html>
  <body>
    Authors:
    <ol>
      <li>Eckstein, Rainer</li>
      <li>Eckstein, Silke</li>
      <li>Muellner, Leonard</li>
      <li>T. Ray, Erik</li>
      <li>Walsh, Norman</li>
    </ol>
  </body>
</html>
```

Figure 51: Result of the query on the `books.xml` document.

Performing XQuery Queries by Saxon

XQuery queries can be performed, e.g., by Saxon.

call (with saxon8.jar in classpath):

```
java net.sf.saxon.Query -s anarticle.xml element.xq
```


Some First XML Software

- XML Processors / Parsers:
  - Apache Xerxes (http://xml.apache.org/xerces2-j/index.html).
    v2.9.1: XML 1.1; Namespaces 1.1, XML Schema 1.0.

- XQuery Processor:
  - Saxon (http://saxon.sourceforge.net; Michael H. Kay).
    v9.0.0.5: XSLT 2.0, XPath 2.0; XQuery 1.0.

Summary

- XML is an industry standard for document and data interchange languages.
- XML documents are made from nested elements with attributes and text content.
- XML documents need to be well-formed.
- XML Schema associates elements with types and thus allows to define a vocabulary and a structure for a specific class of documents.
- XPath allows to address parts of an XML document with path expressions made from axis steps and predicates.
- XQuery builds on XPath and allows complex queries to XML documents with FLOWR expressions.

References