XML Technologies

Doc. RNDr. Irena Holubova, Ph.D.

holubova@ksi.mff.cuni.cz

Web pages:
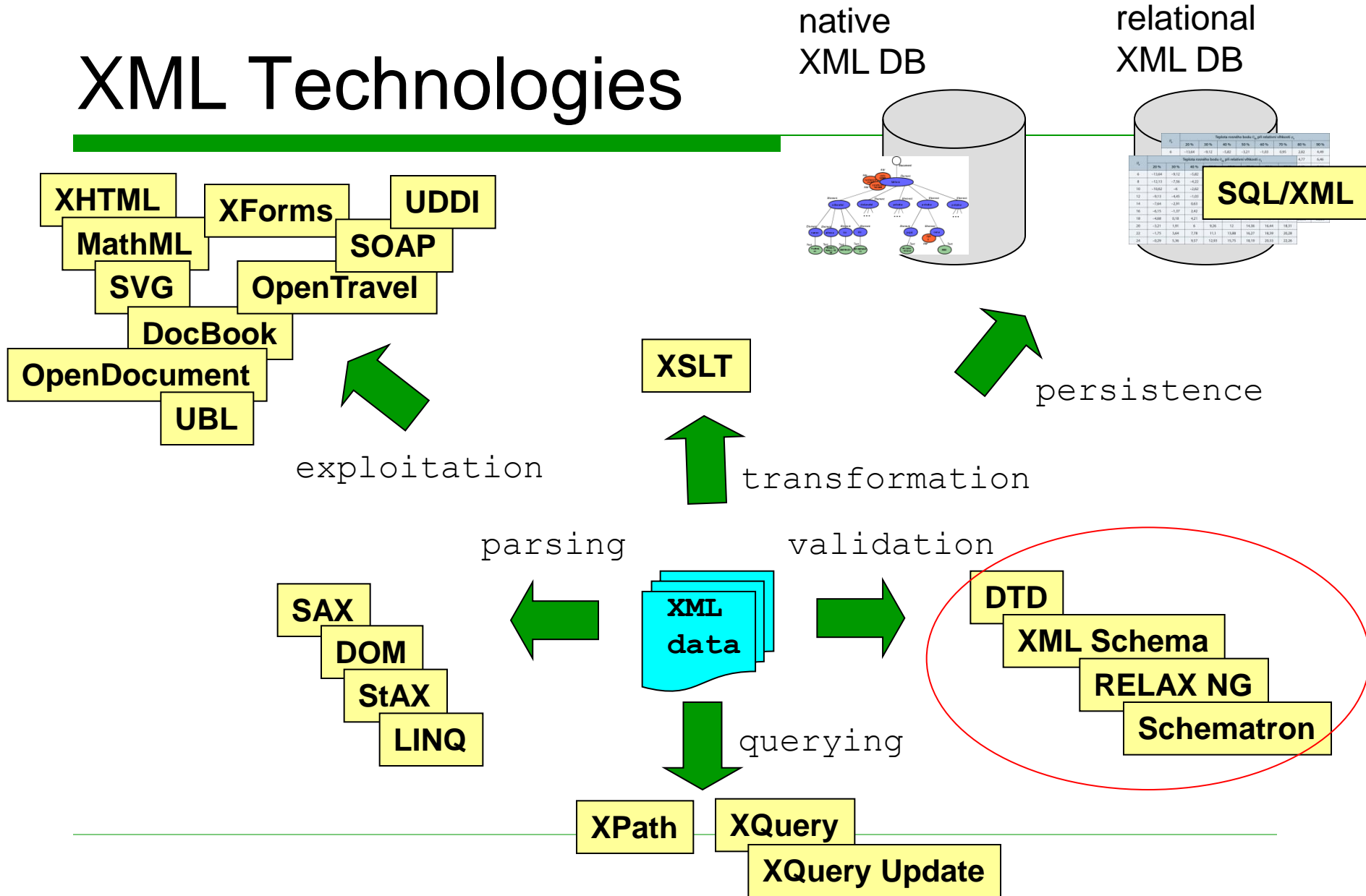MFF: http://www.ksi.mff.cuni.cz/~holubova/NPRG036/
FEL: http://www.ksi.mff.cuni.cz/~holubova/A7B36XML/

Outline

- ☐ Introduction to XML format, overview of XML technologies
- ☐ DTD
- ☐ XML data models
- ☐ Interfaces for XML data
- ☐ XPath
- ☐ XSLT
- ☐ XQuery, XQuery Update
- ☐ XML schema languages
- ☐ SQL/XML
- ☐ An overview of standard XML formats
- ☐ XML data persistence

XML Technologies

native XML DB

relational XML DB

SQL/XML

XHTML
XForms
UDDI
MathML
SOAP
SVG
OpenTravel
DocBook
OpenDocument
UBL

XSLT

persistence

exploitation

transformation

parsing

validation

SAX
DOM
StAX
LINQ

XML data

DTD
XML Schema
RELAX NG
Schematron

querying

XPath
XQuery
XQuery Update

Best Practices

Best Practices

- ☐ How to define XML schemas for various use cases
  - ■ *Are we going to use the schema <u>locally or share</u> it with others?*
  - ■ *Will the schema <u>evolve</u>?*
  - ■ *Are we going to preserve <u>multiple versions</u> of schemas?*
- ☐ There are many recommendations
  - ■ Fact: The W3C specification does not recommend anything

Basic Recommendations

☐ Use the XML features fully:

☐ Use your <u>own</u> elements and attributes

```xml
<?xml version="1.0"?>
<element name="order">
  <attribute name="number" value="ORD001" />
  <element name="employee">
   ...
  </element>
 ...
</element>
```

Basic Recommendations

☐ Use the XML features fully:
- ■ Maximize readability of XML documents using <u>reasonable</u> element and attribute <u>names</u>
- ■ Even though the names are longer
  - ☐ We can use (XML-specific) compression

```xml
<?xml version="1.0"?>
<o n="ORD001">
 <e><nm>Martin Necasky</nm></e>
 <il>
  <i><m>5</m><c>5</c></i>
 </il>
 ...
</o>
```

Basic Recommendations

☐ Use the XML features fully:

   ■ Do not use <u>dot notation instead</u> of XML tree hierarchy

```
<?xml version="1.0"?>
<order number="ORD001">
 <customer.name.first>Martin</customer.name.first>
 <customer.name.surname>Necasky</customer.name.surname>
 <customer.address.street>Malostranské nám.</customer.address.street>
 <customer.address.number>25</customer.address.number>
 ...
</order>
```

Basic Recommendations

☐ Use the XML features fully:

  ◾ Do not use <u>references instead of tree hierarchy</u>

    ☐ They usually have less efficient processing

    ☐ (Of course, in some cases it might make sense)

 ◾ XML data model != relational data model

```xml
<?xml version="1.0"?>
<selling>
 <order number="ORD001"><employee ref="Z002"/>...</order>
 <order number="ORD002"><employee ref="Z001"/>...</order>
 <employee id="Z001">...</employee>
 <employee id="Z001">...</employee>
</selling>
```

Elements vs. Attributes

- ☐ When to use elements and when attributes?
- ☐ There is no general rule, but:
  - ■ An attribute can be specified <u>only once</u> for each element
  - ■ An attribute can not have a <u>complex structure</u>
- ☐ If we assume new versions of schema, where complex structure or repetition is expected, element is a better choice
  - ■ If we use attributes, we must:
    - ☐ Transform all existing documents
    - ☐ Change all XPath paths
    - ☐ …
  - ■ Repetition can be solved using a multivalue attribute, but its further parsing is more difficult

Elements vs. Attributes

```xml
<?xml version="1.0"?>
<order number="ORD001">
 <employee name="Martin Necasky" email="martinnec@gmail.com" />
</order>
```
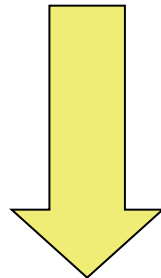
```xml
<?xml version="1.0"?>
<order number="ORD001">
 <employee>
  <name><first>Martin</first><surname>Necasky</surname></name>
  <email>martinnec@gmail.com</email>
  <email>necasky@ksi.mff.cuni.cz</email>
 </employee>
</order>
```
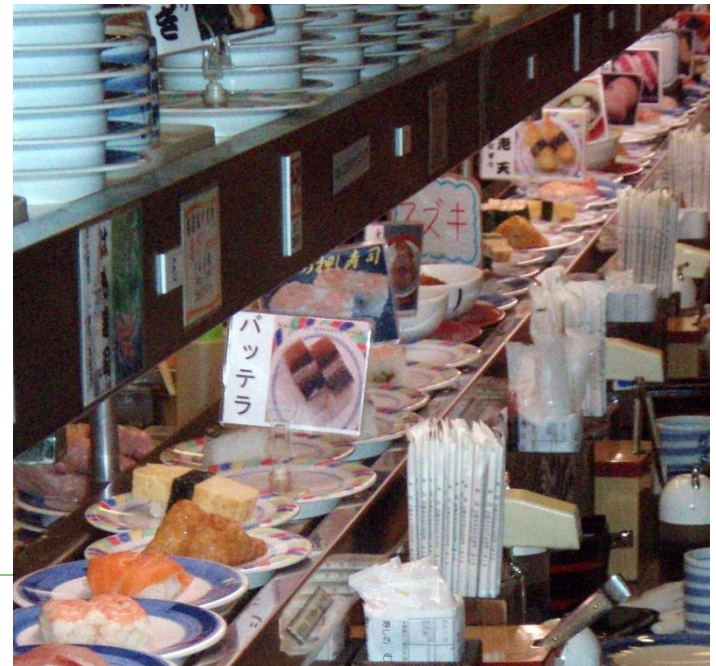
Elements vs. Attributes

☐ If we want to prefer value A to value B, we use for A element and for B attribute

  ◼ B says something about A, it contains metadata

```xml
<?xml version="1.0"?>
<menu>
 <item>
  <portion unit="it">6</portion>
  <name lang="jp">Maguro Maki</name>
  <name lang="cz">Maki Tuňák</name>
 </item>
 <item>
  <portion unit="g">200</portion>
  <name lang="jp">Sake Nigiri</name>
  <name lang="cz">Nigiri Losos</name>
 </item>
</menu>
```

Elements vs. Attributes

- The decision may also depend on the API we use for data access

- SAX: all attribute values are provided in a single event `startElement`

  - May be useful, when we do not want to wait for reading of subelements

  - The resulting code will be simpler and more lucid

Elements vs. Attributes

- ☐ Anything that can be an attribute, can be also an element

- ☐ In general: Use attributes so that the work with the data is suitable for the particular application

Namespaces

- ☐ Use them!
  - ■ When you do not use them, you limit others in usage of your schemas
- ☐ A namespace is identified with a URI
  - ■ It is good when you can put there something related
    - ☐ Documentation of the schema, examples, references to related schemas, …
    - ☐ The schema itself also, but it is not compulsory
  - ■ Note: URI does not mean that it is an address of the schema! It is an identifier!
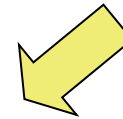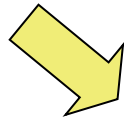
Namespaces

- Usually in a single system you will design multiple related schemas and re-use existing
- In this context there are three key approaches for work with namespaces
  - Heterogeneous
  - Homogeneous
  - Chameleon
- Example:
  - order.xsd uses
    - customer.xsd
    - product.xsd

Heterogeneous Design of XSDs

☐ Each schema has its <u>own target namespace</u>
 (attribute `targetNamespace` of element `schema`)

```
<xsd:schema
 targetNamespace=
  "http://www.customer.org">
 ...
</xsd:schema>
```
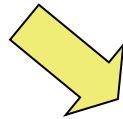
```
<xsd:schema
 targetNamespace=
  "http://www.product.org">
 ...
</xsd:schema>
```

```
<xsd:schema
 targetNamespace="http://www.order.org">
 <xsd:import namespace="http://www.customer.org"
             schemaLocation="customer.xsd"/>
 <xsd:import namespace="http://www.product.org"
             schemaLocation="product.xsd"/>
 ...
</xsd:schema>
```

Homogeneous Design of XSDs

☐ <u>One namespace</u> for all schemas

```
<xsd:schema
 targetNamespace=
  "http://www.order.org">
 ...
</xsd:schema>
```

```
<xsd:schema
 targetNamespace=
  "http://www.order.org">
 ...
</xsd:schema>
```

```
<xsd:schema
 targetNamespace="http://www.order.org">
 <xsd:include schemaLocation="customer.xsd"/>
 <xsd:include schemaLocation="product.xsd"/>
 ...
</xsd:schema>
```

Chameleon Design of XSDs

- ☐ Chameleons <u>do not have target namespace</u>
  - ■ No attribute `targetNamespace`
- ☐ Chameleons become a part of namespace into which they are included

```
<xsd:schema>
 ...
</xsd:schema>
```

```
<xsd:schema>
 ...
</xsd:schema>
```

```
<xsd:schema
 targetNamespace="http://www.order.org">
 <xsd:include schemaLocation="customer.xsd"/>
 <xsd:include schemaLocation="product.xsd"/>
 ...
</xsd:schema>
```
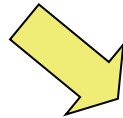
Namespaces – Recommendations

☐ Homogeneous approach:
- ■ For XML schemas which are conceptually linked
  - ☐ i.e. describe the same problem domain and have a common administrator
- ■ There are no conflicts among element / attribute names
  - ☐ Collision: Two globally defined elements / attributes with the same name but different type
- ■ There are no conflicts among types
  - ☐ Collision: Two globally defined data types with the same name but different content model

Namespaces – Recommendations

☐ Heterogeneous approach:

- ■ If the XML schemas describe domains of different problems

- ■ If the XML schemas have distinct administrators

- ■ If there may occur a collision of elements / attributes / types

Namespaces – Recommendations

□ 　🦎　:

- ■ If the XML schema defines general types which do not have a special semantics or which can be used regardless the problem domain

  - □ e.g. data types for an address, name, e-mail, …
  - □ e.g. general data types like array / list of items, string of a particular length, …

XML Schema Versioning

☐ Your XML schemas will <u>evolve</u> over time

- Users have new requirements

- The world is changing

- …

☐ If we use XML data format, we usually have to manage multiple versions of each schema

☐ The users must be provided with the version they are interested in and want to use

- And, of course, the XML documents valid against the selected version

XML Schema Versioning

□ 1ˢᵗ option: XML Schema attribute `version` of element `schema`

  ■ Problem: XML documents do not know this information

    □ This version cannot be directly checked

```
<xsd:schema version="BFLM.PSFVZ">
 <element name="order"
          type="Order"/>
 ...
</xsd:schema>
```

```
<?xml version="1.0"?>
<order ... >

</order>
```

This is NOT the version

XML Schema Versioning

☐ 2nd option: own versioning attribute declared for the root element

■ XML documents must have correct version of XML schema in the root element

☐ If not, validation reveals it

■ Problem: XML documents must change also in case when it is not necessary (the version must change)

```
<xsd:schema>
 <xsd:element name="order"
              type="Order"/>
 <xsd:complexType name="Order">
  ...
  <xsd:attribute name="version"
               type="string"
               fixed="AEIO.U"/>
 </xsd:complexType>
 ...
</xsd:schema>
```

```
<?xml version="1.0"?>
<order version="AEIO.U" >

</order>
```

XML Schema Versioning

□ 3rd option: including the number of version into the value of attribute `targetNamespace`

   ■ XML documents can use the namespace (with particular version) suitable for particular situation

```
<xsd:schema
 targetNamespace=
  ".../orders/1.0">
 ...
</xsd:schema>
```

```
<?xml version="1.0"?>
<order
 xsi:schemaLocation=".../orders/1.0
                       Orders.xsd">

</order>
```

XML Schema Versioning

- Use versions for your XML schemas
- Force the XML documents to contain version number of the schema
  - Somehow
- Provide the users with all versions of XML schemas
- Specify versioning notation so that the users know which changes influence validity of XML documents
  - E.g. *X.Y*, where:
    - New *Y* means that XML documents valid against *X.Y-1* ARE valid against *X.Y*
    - New *X* means that XML documents valid against *X-1.Y* DO NOT HAVE TO BE valid against *X.0*

XML Schema Versioning

- ❑ Another problem:
  - ■ The provider of XML data has several versions of XML schemas (*1, …, n*)
  - ■ The consumer wants XML data valid against version *i*
  - ■ The provider does not want to store the XML data in all possible versions
    - ❑ Inefficient (space requirements, management, …)
- ❑ Solution:
  - ■ Internal representation such that:
    - ❑ The data are stored only once
    - ❑ We can export the data into the version required by user
      - ■ Using XSLT | XQuery | SQL/XML

XML Schema Versioning
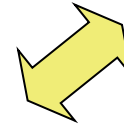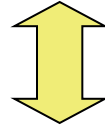
```
Customer 1:
 - number
 - name
 - email
```

```
Customer 2:
 - code
 - name
    - first
    - surname
 - email+ | phone
 - discount
```

```
Customer 3:
 - code
 - name
    - first
    - surname
 - email*
 - phone
```

```
Customer:
 - number
 - first
 - surname
 - email*
 - phone?
 - discount?
```

Extensible Content Types

- ☐ A basic strategy is derivation of new data types from existing ones using `extension`
- ☐ But, what if:
  - ■ Authors want <u>more freedom</u> in extension of data types?
    - ☐ But they cannot modify the schema
  - ■ We have <u>insufficient information</u> for definition of the type?
  - ■ The <u>variability</u> of the content is so extensive that we cannot cover it in the schema?
  - ■ The structure of data <u>changes so fast</u> that we cannot change it on time?
    - ☐ e.g. the domain of mobile phones

Extensible Content Types

□ If `extension` is insufficient, use `any`!

```
<complexType name="Publication">
 <sequence>
  <element name="name"/>
  <element name="price"/>
  <any namespace="##any" minOccurs="0"
       maxOccurs="unbounded"/>
  <element name="publisher"/>
 </sequence>
</complexType>
```

```
<?xml version="1.0"?>
<publication
  xmlns="http://www.publication.org"
  xmlns:rc="http://www.review.org">
 <name>
  King's speech
 </name>
 <price>LOW</price>
 <rc:review>
  ...
 </rc:review>
 <publisher>The best one</publisher>
</publication>
```

But…

Extensible Content Types

☐ Type `Publication` is nondeterministic!
  - ■ When the parser validates **publisher**, it does not know where it is defined

```
<complexType name="Publication">
 <sequence>
  <element name="name"/>
  <element name="price"/>
  <any namespace="##any" minOccurs="0"
       maxOccurs="unbounded"/>
  <element name="publisher"/>
 </sequence>
</complexType>
```

```
<?xml version="1.0"?>
<publication
   xmlns="http://www.publication.org"
   xmlns="http://www.review.org">
 <name>
  King's speech
 </name>
 <price>LOW</price>
 <rc:review>

  ...
 </rc:review>
 <publisher>The best one</publisher>
</publication>
```

Extensible Content Types

□ Better:

```
<complexType name="Publication">
 <sequence>
  <element name="name"/>
  <element name="price"/>
  <element name="other">
   <complexType>
    <sequence><any .../></sequence>
   </complexType>
  </element>
  <element name="publisher"/>
 </sequence>
</complexType>
```

```
<?xml version="1.0"?>
<publication
   xmlns="http://www.publication.org"
   xmlns="http://www.review.org">
 <name>
  King's speech
 </name>
 <price>LOW</price>
 <other>
  <rc:review>
   ...
  </rc:review>
 </other>
 <publisher>The best one</publisher>
</publication>
```

Conclusion

- ☐ The provided set of cases was not complete, but a set of motivating examples
  - ■ There are numerous other best practices, recommendations etc.
    - ☐ More or less reasonable
- ☐ General advice: If you design a set of XML schemas, first think about the particular application and consequences

RELAX NG

RELAX NG

- Results from two older languages:
  - TREX (Tree Regular Expressions for XML)
    - James Clark
    - http://www.thaiopensource.com/trex/
  - RELAX (Regular Language Description for XML)
    - Murata Makoto
    - http://www.xml.gr.jp/relax/
- ISO standard: ISO/IEC 19757-2:2002
- Based on the idea of patterns
  - RELAX NG schema = a pattern of XML document
  - Note: XML Schema is considered to be based on types

General features

- ☐ Simple and easy-to-learn
- ☐ Has two types of syntaxes: XML a compact (non-XML)
  - ■ Mutually convertible
  - ■ XML Schema does not have a compact version
- ☐ Supports namespaces
  - ■ DTD does not
- ☐ Has unlimited support for unordered sequences
  - ■ XML Schema does not
- ☐ Has unlimited support for mixed content
  - ■ DTD does not
- ☐ It enables to use a wide set of simple data types
  - ■ e.g. from XML Schema

XML vs. Compact Syntax

```
<element xmlns="http://relaxng.org/ns/structure/1.0"
         name="employee">
 <attribute name="id">
   <text/>
 </attribute>
 <element name="name">
   <text/>
 </element>
 <element name="surname">
   <text/>
 </element>
 <element name="salary">
   <text/>
 </element>
</element>
```

```
element employee {
  attribute id { text },
  element name { text },
  element surname { text },
  element salary { text } }
```

```
<employee id="101">
 <name>Irena</name>
 <surname>Holubova</surname>
 <salary>1000000</salary>
</employee>
```

Basic Patterns

☐ Arbitrary text: `<text/>`   `text`

☐ Attribute:

```
<attribute name="note">
 <text/>
</attribute>

<attribute name="note"/>
```

`attribute note { text }`

☐ Element with text content:

```
<element name="name">
 <text/>
</element>
```

`element name { text }`

Basic Patterns

☐ Empty element:

```
<element name="hr">
 <empty/>
</element>
```

```
element hr { empty }
```

☐ Element with an attribute (and text content):

```
<element name="person">
 <attribute name="id"/>
 <text/>
</element>

<element name="person">
 <text/>
 <attribute name="id"/>
</element>
```

```
element person {
  attribute id { text },
  text }
element person {
  text,
  attribute id { text } }
```

The order is not important

Basic Patterns

☐ Element with subelements (and an attribute):

```
<element name="person">
 <element name="name">
  <text/>
 </element>
 <element name="surname">
  <text/>
 </element>
 <element name="salary">
  <text/>
 </element>
 <attribute name="id"/>
</element>
```

```
element person {
  element name { text },
  element surname { text },
  element salary { text },
  attribute id { text } }
```

Basic Patterns

☐ Optional pattern:

```
<element name="person">
  <element name="name">
    <text/>
  </element>
  <optional>
    <element name="fax">
      <text/>
    </element>
  </optional>
  <optional>
    <attribute name="note"/>
  </optional>
</element>
```

```
element person {
  element name { text },
  element fax { text }?,
  attribute note { text }? }
```

…

Basic Patterns

```
<element name="person">
 <element name="name">
  <text/>
 </element>
 <element name="surname">
  <text/>
 </element>
 <optional>
  <element name="fax">
   <text/>
  </element>
  <attribute name="note"/>
 </optional>
</element>
```

```
element person {
 element name { text },
 element surname { text },
 ( element fax { text },
  attribute note { text } )? }
```

!

Neither in DTD nor in XML
Schema 1.0 can be
expressed

- version 1.1: assert

Basic Patterns

☐ Repeating patterns:

```
<element name="person">
 <element name="name">
  <text/>
 </element>
 <oneOrMore>
  <element name="phone">
   <text/>
  </element>
 </oneOrMore>
 <attribute name="id"/>
</element>
```

```
element person {
 element name { text },
 element phone { text }+,
 attribute id { text } }
```

zeroOrMore ⟺ *

Precise number of
 occurrences as in DTD

■ Non-trivial

Advanced Patterns

☐ Ordered sequences:

```
<element name="person">
  <element name="name">
    <text/>
  </element>
  <element name="surname">
    <text/>
  </element>
  <element name="email">
    <text/>
  </element>
</element>
```

```
<element name="person">
  <group>
    <element name="name">
      <text/>
    </element>
    <element name="surname">
      <text/>
    </element>
    <element name="email">
      <text/>
    </element>
  </group>
</element>
```

Advanced Patterns

☐ Choice:

```
<element name="person">
 <element name="name">
  <text/>
 </element>
 <choice>
  <element name="email">
   <text/>
  </element>
  <element name="phone">
   <text/>
  </element>
 </choice>
</element>
```

```
element person {
 element name { text },
 ( element email { text } |
   element phone { text } ) }
```

Advanced Patterns

☐ Unordered sequences:

```
<element name="person">
 <element name="name">
  <text/>
 </element>
 <interleave>
  <element name="email">
   <text/>
  </element>
  <element name="phone">
   <text/>
  </element>
 </interleave>
</element>
```

```
element person {
 element name { text },
 ( element email { text } &
   element phone { text } ) }
```

Contrary to XML Schema no restrictions

Advanced Patterns

☐ Mixed content:

```
<element name="para">
 <interleave>
  <zeroOrMore>
   <element name="bold">
    <text/>
   </element>
  </zeroOrMore>
  <zeroOrMore>
   <element name="italic">
    <text/>
   </element>
  </zeroOrMore>
  <text/>
 </interleave>
</element>
```

```
element para {
  element pojem { bold }* &
  element odkaz { italic }* &
  text }
```

<text/> corresponds to arbitrary number of text nodes

■ Does not need + or *

Advanced Patterns

```
<element name="para">
 <mixed>
  <zeroOrMore>
   <element name="bold">
    <text/>
   </element>
  </zeroOrMore>
  <zeroOrMore>
   <element name="italic">
    <text/>
   </element>
  </zeroOrMore>
 </mixed>
</element>
```

⟺

```
<element name="para">
 <mixed>
  <group>
   <zeroOrMore>
    <element name="bold">
     <text/>
    </element>
   </zeroOrMore>
   <zeroOrMore>
    <element name="italic">
     <text/>
    </element>
   </zeroOrMore>
  </group>
 </mixed>
</element>
```

```
element para {
 mixed {
  element bold { text }* &
  element italic { text }* } }
```

group ⇒ interleave:

Data Types

- ☐ Built-in: string and token
- ☐ Typically we use XML Schema data types

```
<element name="price">
 <data type="decimal"
       datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"/>
</element>
```

```
<element name="coordinates"
         datatypeLibrary="...">
 <element name="x">
  <data type="double"/>
 </element>
 <element name="y">
  <data type="double"/>
 </element>
</element>
```

```
datatypes xs = "…"
element price { xs:decimal }
```

```
element price { xsd:decimal }
```

xsd = default for XML Schema

Data Types

☐ Parameters of data types:

```
<element name="salary">
 <data type="decimal">
   <param name="minInclusive">100000</param>
   <param name="maxExclusive">1000000</param>
 </data>
</element>
```

```
element salary {
  xsd:decimal {
    minInclusive = "100000"
    maxExclusive = "1000000" } },
```

☐ Enumerations:

```
<attribute name="ports">
 <choice>
   <value>1001</value>
   <value>1002</value>
   <value>1003</value>
 </choice>
</attribute>
```

```
attribute ports {
   "1001" | "1002" | "1003" }
```

Data Types

☐ Negative enumeration:

```
<element name="color">
 <data type="string">
  <except>
   <choice>
     <value>black</value>
     <value>white</value>
   </choice>
  </except>
 </data>
</element>
```

```
element color {
  string – ( "black" | "white" ) }
```

Data Types

☐ Multivalue type:

```
<element name="vehicle">
 <list>
  <oneOrMore>
   <data type="token"/>
  </oneOrMore>
 </list>
</element>
```

```
element vehicle {
  list { token+ }
```

```
<element name="vehicle">
 <list>
  <oneOrMore>
   <choice>
    <value>rickshaw</value>
    <value>camel</value>
    <value>elephant</value>
   </choice>
  </oneOrMore>
 </list>
</element>
```

```
element vehicle {
  list { ("rickshaw" | "camel" | "elephant" )+ } }
```

```
<vehicle>rickshaw camel</vehicle>
```

Data Types

```
<attribute name="dimensions">
 <list>
   <data type="decimal"/>
   <data type="decimal"/>
   <data type="decimal"/>
   <choice>
     <value>cm</value>
     <value>mm</value>
   </choice>
 </list>
</attribute>
```

```
attribute dimensions {
  list { xsd:decimal,
         xsd:decimal,
         xsd:decimal,
         ("cm" | "mm" ) } }
```

```
dimensions="40 38.5 90 cm"
```

Conclusion

- ☐ Other schema components:
  - ■ Naming of schema parts + repeatable usage
  - ■ Usage of namespaces
  - ■ Documentation and comments
- ☐ Other sources of information:
  - ■ RELAX NG Tutorial:
    http://www.relaxng.org/tutorial-20011203.html
  - ■ RELAX NG Compact Syntax Tutorial:
    http://www.relaxng.org/compact-tutorial-20030326.html
  - ■ http://www.relaxng.org/

Schematron

Schematron

- ISO standard: ISO/IEC 19757-3:2006
- Based on the idea of <u>patterns</u>
  - Similar meaning as in XSLT
  - Does not define a grammar
    - DTD, XML Schema, RELAX NG are <u>grammar-based</u>
- Defines a set of rules which must be followed by a valid XML document
  - Expressed using XPath
  - For validation we can use an XSLT processor

Structure of Schema

Root element <schema xmlns="http://purl.oclc.org/dsdl/schematron"> contains:

- ☐ <title> – name of schema (optional)
- ☐ <ns prefix="…" uri="…" /> – declarations of namespace prefixes (arbitrary amount)
- ☐ <pattern> – (at least one) pattern containing:
  - ■ <rule context="…"> – (at least one) rule applied in the context and containing subelements:
    - ☐ <assert test="…"> – if XPath expression in attribute test <u>does not</u> return true, the result of validation is the content of assert
    - ☐ <report test="…"> – if XPath expression in attribute test <u>does</u> return true, the result of validation is the content of report

Example

```xml
<?xml version="1.0" encoding="utf-8"?>
<schema xmlns="http://www.ascc.net/xml/schematron">
 <pattern name="The list of employees is not empty">
  <rule context="employees">
   <assert test="employee">There must be at least one employee in
    the list</assert>
   <report test="sum(employee/salary) &gt; 500000">The sum of salaries
    can not be greater than 500.000</report>
  </rule>
 </pattern>
 <pattern name="Rules for an employee">
  <rule context="employee">
   <assert test="name">An employee must have a name.</assert>
   <assert test="surname"> An employee must have a surname.</assert>
   <assert test="email"> An employee must have an e-mail.</assert>
   <assert test="@id"> An employee must have an id.</assert>
   …
```

Example

```
    …
    <report test="name[2]|surname[2]">An employee can not have multiple
names.</report>
   </rule>
 </pattern>
 <pattern name="Duplicity of IDs">
   <rule context="employee">
    <report test="count(../employee[@id = current()/@id]) &gt; 1">
      Duplicity in ID <value-of select="@id"/> of element
      <name/>.</report>
   </rule>
 </pattern>
</schema>
```

☐ <name> – name of current element

☐ <value-of select="…"> – result of XPath expression in select

Validation

- ☐ Option A: Special SW / library for Schematron
- ☐ Option B: Arbitrary XSLT processor
    - ■ There exists an XSLT script which transforms Schematron schemas to XSLT scripts
    - ■ The resulting XSLT script is applied on XML documents
    - ■ The result is content of assert a report elements

Combination with XML Schema

```xml
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns:sch="http://www.ascc.net/xml/schematron">
 <xs:element name="employees">
  <xs:annotation>
   <xs:appinfo>
     <sch:pattern name="Do we have enough for salaries?">
      <sch:rule context="employees">
       <sch:report test="sum(employee/salary) &gt; 50000">The sum of
salaries can not be greater than 50.000.</sch:report>
      </sch:rule>
     </sch:pattern>
   </xs:appinfo>
  </xs:annotation>
  <xs:complexType>
    <!-- … definition of element content in XSD … -->
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Combination with XML Schema

- Validation:
    - Option A: Special SW / library for Schematron
    - Option B: Using XSLT we extract a Schematron schema and validate it

Combination with RELAX NG

```xml
<?xml version="1.0" encoding="utf-8"?>
<element xmlns="http://relaxng.org/ns/structure/1.0"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"
         xmlns:sch="http://www.ascc.net/xml/schematron"
         name="employees">
 <sch:pattern name="Do we have enough for salaries?">
  <sch:rule context="employees">
   <sch:report test="sum(employee/salary) &gt; 50000">The sum of
salaries can not be greater than 50.000.</sch:report>
  </sch:rule>
 </sch:pattern>
 <oneOrMore>
  <!-- … definition of element content in XSD … -->
 </oneOrMore>
</element>
```

Combination with RELAX NG

☐ There exists also a compact non-XML version:

```
namespace sch = "http://www.ascc.net/xml/schematron"
[ sch:pattern [ name = " Do we have enough for salaries? "
   sch:rule [ context = "employees"
    sch:report [ test = "sum(employee/salary) > 50000"
                 "The sum of salaries can not be greater than 50.000." ] ] ] ]
```

☐ Validation:

- ■ Option A: Special SW / library for Schematron
- ■ Option B: Using XSLT we extract a Schematron schema and validate it

Conclusion

- A different approach to schema validation
  - Does not define a grammar
  - Usage of XPath + combination with other languages
    - The language itself has limitations – the grammar definition is still more user friendly
- Inspiration for XML Schema assert element
- Other information:
  - http://www.schematron.com/