Semantic Web - XML

Prof. Dr. Steffen Staab
Dipl.-Inf. Med. Bernhard Tausch
(One) Layer Model of the Semantic Web
Extensible Markup Language

Purpose here: storing and exchanging knowledge
Non-purpose here: structuring documents
XML

- eXtensible Markup Language
- Origin: structured Text (HTML4.0 ∈ XML ⊂ SGML)
- Web-Standard (W3C) for data exchange:
  - Input- and output data of applications may be described via XML
  - Industry just need to agree on standardized description (cf. Oasis.org)
- Complementary to HTML:
  - HTML describes presentation of content
  - XML describes structure of content
- Data – Point of view: XML as data model for semi-structured data
XML model

Depiction of objects as directed graphs

Nodes are labelled

Object identifier is attributed to the node

Text attributed to the node
XML Syntax (1) – XML element

- XML element
  - Description of objects via corresponding tags like `<author>` and `</author>`
  - Content of an element: Text and/or further elements (subelements)
  - Elements may be nested arbitrarily deep
  - Empty elements: `<year></year>` abbreviated: `<year/>`

```
<author>
  <firstname>Serge</firstname>
  <lastname>Abiteboul</lastname>
  <email>sab@abc.com</email>
  email address may be wrong!
</author>
```
XML syntax (2) – XML attributes

- XML attribute:
  - Name-value-pair
  - Associated to element
  - Alternative for describing data

```xml
<author firstname="Serge" lastname="Abiteboul" email="sab@abc.com"/>
```

Still another syntactic option to describe the same data:

```xml
<author firstname="Serge" lastname="Abiteboul" email="sab@abc.com"/>
```
XML model

<Bib id="o1">
  <paper id="o12">
    <title> Foundations of Databases </title>
    <author>
      <firstname> Serge </firstname>
      <lastname> Abiteboul </lastname>
    </author>
    <year> 1997 </year>
    <publisher> Addison Wesley </publisher>
  </paper>
  ...
</Bib>
XML vs HTML

- **HTML**: fixed tags and semantics (presentation of text)
- **XML**: variable tag set for describing application specific syntax (meta grammar)
- **XML \( \subseteq \) SGML
XML document

- **XML document:**
  - Text document with XML descriptions
  - Data base point of view: database

- **Well-formed XML document:**
  - All elements correctly nested within corresponding start and end tags
  - Document contains one root element
  - Well-formed documents may still contain unstructured text (mind the XML symbols!)

- **Valid XML document:**
  - Well-formed XML document that conforms to associated schema
  - Schema is used to validate XML document
  - Reasonable for data exchange or web portal
Structuring XML
Schemata in XM

- **DTD – Document Type Definitions:**
  - Simple grammar for an XML document
    - Declaration of elements, attributes, u.a.
    - Restricts arbitrary nesting of elements and attributes
  - Part of XML standard
  - Inherited from SGML

- **XML schema:**
  - Complex data definition language:
    - Many standardized base types: float, double, decimal, boolean, date, ...
    - Types and typed object references
    - Type hierarchies / Inheritance
    - Consistency constraints
  - Complements XML standard
  - Downwards compatible to DTD
XML-Schemata I: DTD

- DTD defines (kind of) context free grammar for XML document
- Arbitrary elements and attributes are restricted to a selected subset and selected structures

```xml
<!--DOCTYPE bib [
  <!ELEMENT bib (paper*)>
  <!ELEMENT paper (author+, year, publisher?)>
  <!ATTLIST paper id ID #REQUIRED>
  <!ELEMENT author (firstname*, lastname)>
  <!ATTLIST author age CDATA #IMPLIED>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT year (#PCDATA)>
  <!ELEMENT publisher (#PCDATA)>
...]
]

<bib>
  <paper id="o12">
    <title>Foundations of Databases</title>
    <author>
      <firstname>Serge</firstname>
      <lastname>Abiteboul</lastname>
    </author>
    <year>1997</year>
    <publisher>Addison Wesley</publisher>
  </paper>
...;
</bib>
```
DTD – Declaration of elements

- Describes restrictions of element contents

- Syntax:
  ```xml
  <!ELEMENT Name (Definition)>
  ```

- Single atomic type: `#PCDATA` (Parsed Character DATA)

- `(a,b,c)`: List of subelements

- `(a|b|c)`: Alternatives

- Cardinalities:
  - `*`: arbitrary often (incl 0)
  - `+`: once or more often
  - `?`: at most once (optional)
  - `(no indication)`: exactly once

- `EMPTY`: Enforce empty element

```xml
<!DOCTYPE bib [
  <!ELEMENT bib (paper*)>  
  <!ELEMENT paper (author+, year, publisher?)>  
  <!ATTLIST paper id ID #REQUIRED>  
  <!ELEMENT author (firstname*, lastname)>  
  <!ATTLIST author age CDATA #IMPLIED>  
  <!ELEMENT firstname (#PCDATA)>  
  <!ELEMENT lastname (#PCDATA)>  
  <!ELEMENT year (#PCDATA)>  
  <!ELEMENT publisher (#PCDATA)>  
  ...  
]>
```

**DTD**
Introduction and definition of root element bib

```xml
<!DOCTYPE bib [  
  <!ELEMENT bib (paper*)>  
  <!ELEMENT paper (author+, year, publisher?)>  
  <!ATTLIST paper id ID #REQUIRED>  
  <!ELEMENT author (firstname*, lastname)>  
  <!ATTLIST author age CDATA #IMPLIED>  
  <!ELEMENT firstname (#PCDATA)>  
  <!ELEMENT lastname (#PCDATA)>  
  <!ELEMENT year (#PCDATA)>  
  <!ELEMENT publisher (#PCDATA)>  
  ...  
  ]>
```

• Describes restrictions of element contents
• Syntax:
  `<!ELEMENT Name (Definition)>`
• Single atomic type: `#PCDATA`
  (Parsed Character DATA)
• `(a,b,c)`: List of subelements
• `(a|b|c)`: Alternatives
• Cardinalities:
  – `*` arbitrary often (incl 0)
  – `+` once or more often
  – `?` at most once (optional)
  – `(no indication)` exactly once
• **EMPTY**: Enforce empty element
DTD – Declaration of elements (3)

- Describes restrictions of element contents
- Syntax:
  `<!ELEMENT Name (Definition)>`
- Single atomic type: `#PCDATA` (Parsed Character DATA)
- `(a,b,c)`: List of subelements
- `(a|b|c)`: Alternatives
- Cardinalities:
  - `*`: arbitrary often (incl 0)
  - `+`: once or more often
  - `?`: at most once (optional)
  - `(no indication)`: exactly once
- `EMPTY`: Enforce empty element

`bib` may contain arbitrarily many elements of type `paper`

```
<!DOCTYPE bib [
  <!ELEMENT bib (paper*)>
  <!ELEMENT paper (author+, year, publisher?)>
  <!ATTLIST paper id ID #REQUIRED>
  <!ELEMENT author (firstname*, lastname)>
  <!ATTLIST author age CDATA #IMPLIED>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT year (#PCDATA)>
  <!ELEMENT publisher (#PCDATA)>
  ...]
```

**DTD**
DTD – Declaration of elements (4)

- Describes restrictions of element contents
- Syntax:
  `<!ELEMENT Name (Definition)>`
- Single atomic type: `#PCDATA` (Parsed Character DATA)
- `(a,b,c)`: List of subelements
- `(a|b|c)`: Alternatives
- Cardinalities:
  - `*` arbitrary often (incl 0)
  - `+` once or more often
  - `?` at most once (optional)
  - `(no indication)`: exactly once
- `EMPTY` : Enforce empty element

```xml
<!DOCTYPE bib [
  <!ELEMENT bib (paper*)>
  <!ELEMENT paper (author+, year, publisher?)>
  <!ATTLIST paper id ID #REQUIRED>
  <!ELEMENT author (firstname*, lastname)>
  <!ATTLIST author age CDATA #IMPLIED>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT year (#PCDATA)>
  <!ELEMENT publisher (#PCDATA)>
  ...
]>```

- `paper` consists of at least one `author`
- exactly one `year` and
- an optional `publisher`
- in exactly this order!
DTD – Declaration of elements (5)

• Describes restrictions of element contents
• Syntax:
  <!ELEMENT Name (Definition)>  
• Single atomic type: #PCDATA
  (Parsed Character DATA)
• (a,b,c): List of subelements
• (a|b|c): Alternatives
• Cardinalities:
  – * arbitrary often (incl 0)
  – + once or more often
  – ? at most once (optional)
  – (no indication): exactly once
• EMPTY : Enforce empty element

firstname is of type string

<!DOCTYPE bib [  
  <!ELEMENT bib (paper*)>  
  <!ELEMENT paper (author+, year, publisher?)>  
  <!ATTLIST paper id ID #REQUIRED>  
  <!ELEMENT author (firstname*, lastname)>  
  <!ATTLIST author age CDATA #IMPLIED>  
  <!ELEMENT firstname (#PCDATA)>  
  <!ELEMENT lastname (#PCDATA)>  
  <!ELEMENT year (#PCDATA)>  
  <!ELEMENT publisher (#PCDATA)>  
  ...]>

DTD
DTD – Declaration of attributes

- Name-stringvalue-pair
- Associated with element
- **Syntax:**
  ```xml
  <!ATTLIST Element
  Attributname1 Type1 Add1
  Attributname2 ...>
  ```
- **Type:**
  - **CDATA** String
  - **ID** OID
  - **IDREF** References
  - **IDREFS** Set of references
- **Add:**
  - **REQUIRED** mandatory
  - **IMPLIED** optional
  - (Initial value)

```xml
<!DOCTYPE bib [  
<!ELEMENT bib (paper*)>  
<!ELEMENT paper (author+, year, publisher?)>  
<!ATTLIST paper id ID #REQUIRED>  
<!ELEMENT author (firstname*, lastname)>  
<!ATTLIST author age CDATA #IMPLIED>  
<!ELEMENT firstname (#PCDATA)>  
<!ELEMENT lastname (#PCDATA)>  
<!ELEMENT year (#PCDATA)>  
<!ELEMENT publisher (#PCDATA)>  
...  
<!ELEMENT publisher (#PCDATA)>  
... ]>
```

```xml
<!DOCTYPE bib [  
<!ELEMENT bib (paper*)>  
<!ELEMENT paper (author+, year, publisher?)>  
<!ATTLIST paper id ID #REQUIRED>  
<!ELEMENT author (firstname*, lastname)>  
<!ATTLIST author age CDATA #IMPLIED>  
<!ELEMENT firstname (#PCDATA)>  
<!ELEMENT lastname (#PCDATA)>  
<!ELEMENT year (#PCDATA)>  
<!ELEMENT publisher (#PCDATA)>  
...  
<!ELEMENT publisher (#PCDATA)>  
... ]>
```
DTD – Declaration of attributes (2)

• Name-string-value-pair
• Associated with element
• Syntax:

```xml
<!DOCTYPE bib [
<!ELEMENT bib (paper*)>
<!ELEMENT paper (author+, year, publisher?)>
<!ATTLIST paper id ID #REQUIRED>
<!ELEMENT author (firstname*, lastname)>
<!ATTLIST author age CDATA #IMPLIED>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
...
]
```

• Type:
  – CDATA String
  – ID OID
  – IDREF References
  – IDREFS Set of references
• Add:
  – REQUIRED mandatory
  – IMPLIED optional
  – (Initial value)

paper comes with attribute id, an OID, which must be assigned a unique value.

```xml
<!DOCTYPE bib [
  <!ELEMENT bib (paper*)>
  <!ELEMENT paper (author+, year, publisher?)>
  <!ATTLIST paper id ID #REQUIRED>
  <!ELEMENT author (firstname*, lastname)>
  <!ATTLIST author age CDATA #IMPLIED>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT year (#PCDATA)>
  <!ELEMENT publisher (#PCDATA)>
  ...
]>
```
DTD – Declaration of attributes (3)

- Name-stringvalue-pair
- Associated with element
- Syntax:
  ```xml
  <!ATTLIST Element
  Attributname1 Type1 Add1
  Attributname2 ...>
  ```
- Type:
  - CDATA String
  - ID OID
  - IDREF References
  - IDREFS Set of references
- Add:
  - REQUIRED mandatory
  - IMPLIED optional
  - (Initial value)

An author may have an attribute age, which may be used to assign him some age value as a string

```xml
<!DOCTYPE bib [
  <!ELEMENT bib (paper*)>
  <!ELEMENT paper (author+, year, publisher?)>
  <!ATTLIST paper id ID #REQUIRED>
  <!ELEMENT author (firstname*, lastname)>
  <!ATTLIST author age CDATA #IMPLIED>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT year (#PCDATA)>
  <!ELEMENT publisher (#PCDATA)>
  ...]
```
DTD – OIDs and references

- DTDs allow for declaration of OIDs, References and reference set as attribute
- Example:

```
<family>
  <person id="jane" mother="mary" father="john">
    <name> Jane Doe </name>
  </person>
  <person id="john" children="jane jack">
    <name> John Doe </name>
  </person>
  <person id="mary" children="jane jack">
    <name> Mary Smith </name>
  </person>
  <person id="jack" mother="mary" father="john">
    <name> Jack Smith </name>
  </person>
</family>
```

```
<!DOCTYPE family [ 
  <!ELEMENT family (person*)> 
  <!ELEMENT person (name)> 
  <!ELEMENT name (#PCDATA)> 
  <!ATTLIST person 
    id ID #REQUIRED 
    mother IDREF #IMPLIED 
    father IDREF #IMPLIED 
    children IDREFS #IMPLIED > ]>
```

DTD
Summary of DTD properties

• Remember: DTDs define context free grammars
  – Recursive definitions are possible

• DTDs are weak for defining schemas:
  – Imposing order where not necessary/wanted:
    <!ELEMENT person ( name, phone ) >
    • Workaround:
      <!ELEMENT person ( (name, phone ) | ( phone, name ) ) >
  – Specifying arbitrary orderings may lead to underspecification:
    <!ELEMENT person ( ( name | phone | email )* ) >
  – References cannot be typed
  – All element names are global (XML supports namespaces, DTDs don’t)
XML-Schemata II: XML-Schema

- Proper schema mechanism with many extensions beyond DTDs
- Uses XML syntax for schema definition

```xml
<schema>
  <element name="bib">
    <complexType>
      <element name="paper" minOccurs="0" maxOccurs="unbounded">
        <complexType>
          <attribute name="id" type="ID" use="required"/>
          <sequence>
            <element name="author" type="authorType" maxOccurs="unbounded"/>
            <element name="year" type="string"/>
            <element name="publisher" type="string" minOccurs="0"/>
          </sequence>
        </complexType>
      </element>
    </complexType>
  </element>
</schema>
```

```xml
<!DOCTYPE bib [ 
  <!ELEMENT bib (paper*)>
  <!ELEMENT paper (author+, year, publisher?)>
  <!ATTLIST paper id ID #REQUIRED>
  <!ELEMENT author (firstname*, lastname)>
  <!ATTLIST author age CDATA #IMPLIED>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT year (#PCDATA)>
  <!ELEMENT publisher (#PCDATA)>
  ... ]>
```
XML-Schema: Elements

• Syntax: `<element name="Name"/>
• Optional attributes:
  – Type
    • type = "Type" atomic, simple or complex typename
  – Cardinalities (default [1,1]):
    • minOccurs = "x" x ∈ \{ 0, 1, n \}
    • maxOccurs = "y" y ∈ \{ 1, n, unbounded \}
  – Value assignments (mutually exclusive!):
    • default = "v" modifiable pre-assignment
    • fixed = "u" unmodifiable pre-assignment
• Examples:
  – `<element name="bib"/>
  – `<element name="paper" minOccurs="0" maxOccurs="unbounded"/>
  – `<element name="publisher" type="string" minOccurs="0"/>
XML-Schema: Attribute

• Syntax: <attribute name="Name"/>

• Optional attributes:
  – Type:
    • type = "Type"
  – Existence:
    • use = "optional"  Cardinality [0,1]
    • use = "required"  Cardinality [1,1]
  – Pre-assignment:
    • use = "default"  value = "v"  modifiable pre-assignment
    • use = "fixed"    value = "u"    unmodifiable pre-assignment

• Examples:
  – <attribute name="id" type="ID" use="required"/>
  – <attribute name="age" type="string" use="optional"/>
  – <attribute name="language" type="string" use="default" value="de"/>
XML schema: Types

• In XML schema we distinguish atomic, simple and complex types

• Atomic types:
  – Built-in basic types like int or string

• Simple types:
  – Have neither subelements nor attributes
  – Typically derived from atomic types

• Complex types:
  – May have elements and attributes

• Additionally, we may distinguish:
  – Pure type definitions describe re-usable type structures
  – Document definitions describe which elements may appear in a document
XML schema: Atomic types

• XML schema supports a large set of built-in basic types (>40):
  – Numerical: byte, short, int, long, float, double, decimal, binary, …
  – Temporal data: time, date, month, year, timeDuration, timePeriod, …
  – Others: string, boolean, uriReference, ID, …

• Examples:
  <element name="year" type="year"/>
  <element name="pages" type="positiveInteger"/>
  <attribute name="age" type="unsignedShort"/>
XML schema: Simple Types

• Existing types may be used to derive new types:
  – Type definition:
    
    ```xml
    <simpleType name="humanAge" base="unsignedShort">
      <maxInclusive value="200"/>
    </simpleType>
    ```
  – Document definition:
    
    ```xml
    <attribute name="age" type="humanAge"/>
    ```

• Such simple types may not contain nested elements!

• Similarly we may define lists:
  – Type definition:
    
    ```xml
    <simpleType name="authorType" base="string" derivedBy="list"/>
    ```
    (Name of authors as a list of strings separated by blanks)
  – Document definition:
    
    ```xml
    <element name="author" type="authorType"/>
    ```
XML schema: Complex types

• Complex types may have subelements and attributes

• Example:
  – Type definition:
    ```xml
    <complexType name="authorType">
      <sequence>
        <element name="firstname" type="string" minOccurs="0"
          maxOccurs="unbounded"/>
        <element name="lastname" type="string"/>
      </sequence>
      <attribute name="age" type="string" use="optional"/>
    </complexType>
    ```

• Collection descriptors:
  – `<sequence> ... </sequence>` Fixed order (a,b)
  – `<all> ... </all>` Arbitrary order (a,b or b,a)
  – `<choice> ... </choice>` Choice (either a or b)
XML schema: complex types

```xml
<complexType name="authorType">
    <sequence>
        <element name="firstname" type="string"
            minOccurs="0"
            maxOccurs="unbounded"/>
        <element name="lastname" type="string"/>
    </sequence>
    <attribute name="age" type="string" use="optional"/>
</complexType>

... basis for further examples!
```
Type hierarchies

- Relationship between two types

- Type definition by
  - Extension
  - Restriction
  of an existing type definition

- All types in XML schema are either
  - Atomic (e.g. string) or
  - Extensions of existing types or
  - Restrictions of existing types

- All types form a type hierarchy
  - Tree with root of type string
  - No multiple inheritance

- Types are downward compatible
  - Substitution principle of type instances
  - Elements of a particular type also accept data that conforms to extensions or restrictions of a given type
Type hierarchies: Extensions of types

• Types may be extended constructively by elements or attributes to form new types

• Example:
  
  ```xml
  <complexType name="extendedAuthorType">
    <extension base="authorType">
      <sequence>
        <element name="email" type="string" minOccurs="0" maxOccurs="1"/>
      </sequence>
      <attribute name="homepage" type="string" use="optional"/>
    </extension>
  </complexType>
  ```

  Extends previously defined type *authorType* by
  
  – An optional element *email*
  – An optional attribute *homepage*
Type hierarchies: Extensions of types (2)

- Extensions are appended to the existing definition:
  
  ```xml
  <complexType name="extendedAuthorType">
    <sequence>
      <element name="firstname" type="string" minOccurs="0" maxOccurs="unbounded"/>
      <element name="lastname" type="string"/>
      <element name="email" type="string" minOccurs="0" maxOccurs="1"/>
    </sequence>
    <attribute name="age" type="string" use="optional"/>
    <attribute name="homepage" type="string" use="optional"/>
  </complexType>
  ```

  ```xml
  <complexType name="authorType">
    <sequence>
      <element name="firstname" type="string" minOccurs="0" maxOccurs="unbounded"/>
      <element name="lastname" type="string"/>
    </sequence>
    <attribute name="age" type="string" use="optional"/>
  </complexType>
  ```
Type hierarchies: Restrictions of types

• Types are restricted by adding constraints that enforce a smaller value range

• Examples for restrictions:
  – New type-, default- or fixed-attributes
  – Tightening of cardinality constraints minOccurs, maxOccurs

• Substitutability
  – Set of instances of restricted subtype is always a subset of the set of instances of the supertype

• Restrictions of complex types
  – Structure is retained: elements or attributes may not be discarded

• Restrictions of simple types
  – Simple types may be restricted (but not extended)
Type hierarchies: restrictions of types (2)

• Example (complex type):

```xml
<complexType name="restrictedAuthorType">
  <restriction base="authorType">
    <sequence>
      <element name="firstname" type="string" minOccurs="0"
               maxOccurs="2"/>
      <element name="lastname" type="string"/>
    </sequence>
    <attribute name="age" type="string" use="required"/>
  </restriction>
</complexType>
```

• Compare to its supertype:

  number of **firstname**s restricted to 2
  attribute **age** is now required
Polymorphic integrity constraints

- Elements and attributes may have integrity constraints attached

- **Uniqueness (candidate key):**
  <unique name="uniqueAuthorName">
    <field xpath="bib/paper/author/@firstname"/>
    <field xpath="bib/paper/author/@lastname"/>
  </unique>

- The combinations of firstname and lastname are always unique in this XML repository
  - **Field** is used to identify elements or attributes
  - **Xpath** is used to indicate where (i.e. following which path) a particular field is found
  - **Xpath:** XML standard for path expressions (simple kind of query language, also basis for more comprehensive standards like XLink, XSL, Xquery)
Polymorphic integrity constraints (2)

- **Key constraint:**
  \[
  \begin{align*}
  &\text{\texttt{<key name="paperKey">}} \\
  &\quad \quad \text{\texttt{<field xpath="bib/paper/@id"/>}} \\
  &\text{\texttt{</key>}}
  \end{align*}
  \]
  
  Attribute id in paper functions as key
  - Keys are unique and may be used for referring unambiguously

- **Foreign key constraint:**
  \[
  \begin{align*}
  &\text{\texttt{<keyref name="paperForeignKey" refer="paperKey">}} \\
  &\quad \quad \text{\texttt{<field xpath="bib/paper/@references"/>}} \\
  &\text{\texttt{</keyref>}}
  \end{align*}
  \]
  
  references is a list of papers referred from a given paper
  - refer points to the name attribute of a key constraint (not directly to an element/attribute)
  - Values in references must always be found among the keys of papers
Example schema

<!-- XMLSchema for literature database -->
<schema>

<!– global root element bib -->
<element name="bib">
    <complexType>
        <element name="paper" minOccurs="0" maxOccurs="unbounded">
            <complexType>
                <attribute name="id" type="ID" use="required"/>
                <sequence>
                    <element name="author" type="authorType" maxOccurs="unbounded"/>
                    <element name="year" type="string"/>
                    <element name="publisher" type="string" minOccurs="0"/>
                    <element name="references" type="listOfPapers" minOccurs="0"/>
                </sequence>
            </complexType>
        </element>
    </complexType>
</element>

<!– List of references to papers (cf bib/paper/@references) -->
<simpleType name="listOfPapers" base="ID" derivedBy="list"/>
Example schema (2)

```xml
<!– Type definitions -->
<complexType name="authorType">
    <sequence>
        <element name="firstname" type="string" minOccurs="0" maxOccurs="unbounded"/>
        <element name="lastname" type="string"/>
    </sequence>
    <attribute name="age" type="humanAge" use="optional"/>
</complexType>

<simpleType name="humanAge" base="unsignedShort">
    <maxInclusive value="200"/>
</simpleType>

<complexType name="extendedAuthorType">
    <extension base="authorType">
        <sequence>
            <element name="email" type="string" minOccurs="0" maxOccurs="1"/>
        </sequence>
        <attribute name="homepage" type="simpleURLType" use="optional"/>
    </extension>
</complexType>

<simpleType name="simpleURLType" base="string">
    <pattern value="http://([^/?#]*)?([^?#]*)(\?([^#]*))?([^#])?"/>
</simpleType>
```
Example schema (3)

<!– Integrity constraints -->

<unique name="uniqueAuthorName">
  <field xpath="bib/paper/author/@firstname"/>
  <field xpath="bib/paper/author/@lastname"/>
</unique>

<key name="paperKey">
  <field xpath="bib/paper/@id"/>
</key>

<keyref name="paperForeignKey " refer=" paperKey ">
  <field xpath="bib/paper/@references"/>
</keyref>

</schema>
Summary of XML Schema

• XML schema gives many more possibilities to define structures of documents/structures of data

• Syntax and expressiveness of XML schema is comprehensive
  – Weakness: few possibilities for integrity constraints
  – Weakness: no logical entailment (see next slide)

• Mehr zu XML-Schema im Web:
  – http://www.w3.org/TR/xmlschema-0/ Introduction
  – http://www.w3.org/TR/xmlschema-1/ Teil I: Structures
  – http://www.w3.org/TR/xmlschema-2/ Teil II: Data types
No logical entailment in XML

```xml
<complexType name="authorType">
  <sequence>
    <element name="firstname" type="string" minOccurs="0" maxOccurs="unbounded"/>
    <element name="lastname" type="string"/>
  </sequence>
</complexType>

<complexType name="bookAuthorType">
  <extension base="authorType">
    <sequence>
      <element name="book" type="string" minOccurs="1" maxOccurs="unbounded"/>
    </sequence>
  </extension>
</complexType>

<complexType name="franticWriterType">
  <extension base="authorType">
    <sequence>
      <element name="book" type="string" minOccurs="10" maxOccurs="unbounded"/>
    </sequence>
  </extension>
</complexType>
```
No logical entailment in XML

Herbert Simon

John Doe
XML Namespaces
XML Namespaces and Programming-Language Modules

- XML namespaces are akin to namespaces, packages, and modules in programming languages

- Disambiguation of tag names from different XML applications ("spaces") through different prefixes

- A prefix is separated from the local name by a ":", obtaining prefix:name tags

- Namespaces constitute a layer on top of XML 1.0, since prefix:name is again a valid tag name and namespace bindings are ignored by some tools ("flat namespaces")
Namespace Bindings

- Prefixes are bound to namespace URIs by attaching an xmlns:prefix attribute to the prefixed element or one of its ancestors, \texttt{prefix:name}\_\texttt{1},..., \texttt{prefix:name}\_\texttt{n}

- The value of the xmlns:prefix attribute is a URI, which may or (unlike for DTDs!) may not point to a description of the namespace’s syntax

- An element can use bindings for multiple namespaces via attributes xmlns:prefix\_\texttt{1},..., xmlns:prefix\_\texttt{m}
Namespaceless Example: Address Variant

Namespaceless XML Markup:

```xml
<address>
  <name>Xaver M. Linde</name>
  <street>Wikingerufer 7</street>
  <town>10555 Berlin</town>
  <bill>12.50</bill>
  <phone>030/1234567</phone>
  <phone>030/1234568</phone>
  <fax>030/1234569</fax>
  <bill>76.20</bill>
</address>
```

*bill is ambiguous tag (name clash from two XML applications)*
Two-Namespace Example:
Snail-Mail and Telecoms Address Parts

Namespace XML Markup:

```xml
<mail:address xmlns:mail="http://www.deutschepost.de/"
               xmlns:tele="http://www.telekom.de/m"
>
  <mail:name>Xaver M. Linde</mail:name>
  <mail:street>Wikingerufer 7</mail:street>
  <mail:town>10555 Berlin</mail:town>
  <mail:bill>12.50</mail:bill>
  <tele:phone>030/1234567</tele:phone>
  <tele:phone>030/1234568</tele:phone>
  <tele:fax>030/1234569</tele:fax>
  <tele:bill>76.20</tele:bill>
</mail:address>
```

- The root element, `mail:address`, as well as the children `mail:name`, `mail:street`, `mail:town`, and `mail:bill`, use the `mail` prefix, bound to a deutschlandepost URI.
- The `tele:phone`, `tele:fax`, and `tele:bill` children use the `tele` prefix, bound to a telekom URI.

*bill disambiguation through mail and tele prefixes*