Module 2

XML Basics

Part 2: XML Schema

Limitations of DTDs

- DTDs describe only the "grammar" of the XML file, not the detailed structure and/or types
- This grammatical description has some obvious shortcomings:
  - we cannot express that a "length" element must contain a non-negative number *(constraints on the type of the value of an element or attribute)*
  - **The "unit"** element should only be allowed when "**amount"** is present *(co-occurrence constraints)*
  - the "**comment"** element should be allowed to appear anywhere *(schema flexibility)*
  - There is no subtyping / inheritance *(reuse of definitions)*
  - There are no composite keys *(referential integrity)*

Overview XML Schema

- Schemas provide a complex type system, similar to strongly-typed object-oriented approaches or the UDT system of SQL 2003 (object-relational types)
- ComplexTypes and SimpleTypes
  - ComplexType correspond to Records/Objects
  - "string" is an example of a SimpleType
- Built-in and user-defined Types
  - ComplexTypes are always user-defined
  - Built-in types cover "usual" types + XML-specific ones
- Elements have complexTypes or simpleTypes; Attributes have simpleTypes

Overview XML Schema (II)

- Visibility/Scope of type and element definitions
  - Global Types vs local Types
  - Global element definition vs local element definitions
- Named vs anonymous types
- Fine-grained control of type properties: "facets"

- Type of Root element of a document is global
- (almost) downward compatible with DTDs
- Schemas are XML Documents (Syntax)
- Namespaces etc. are part of XML Schemas

"Path to Schema" - Agenda

- Schema by Example (syntax, common cases)
- Validation
- Overview on builtin types/simple types
- Defining complex content
- Key constraints
- Namespaces
- Additional Aspects (not relevant for exam)

Example Schema

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:element name="book" type="BookType"/>
     <xsd:complexType name="BookType">
      <xsd:sequence>
           <xsd:element name="title" type="xsd:string"/>
           <xsd:element name="author"
                          minOccurs="1" maxOccurs="unbounded"/>
            <xsd:complexType>
                          <xsd:sequence> ... <xsd:sequence>
             </xsd:complexType>
          </xsd:element>
          <xsd:element name="publisher" type="xsd:anyType"/>
      </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

Example Schema

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

- Schema in a separate XML Document
- Vocabulary of Schema defined in special Namespace. Prefixes "xs"/"xsd" commonly used
- There is a Schema for Schemas (don‘t worry!)
- „schema" Element is always the Root

Example Schema

```
<xsd:element name="book" type="BookType"/>
```

- "element" Element in order to declare elements
- "name" defines the name of the element.
- "type" defines the type of the element
- Declarations under "schema" are ***global***
- Global element declarations are potential roots
- Example: "book" is the only global element, root element of a valid document must be a "book".
- The type of a "book" is BookType (defined next).

Example Schema

```
<xsd:complexType name="BookType">
      <xsd:sequence>
            ...
      </xsd:sequence>
</xsd:complexType>
```

- User-defined complex type
- Defines a sequence of sub-elements
- Attribute "name" specifies name of Type
- This type definition is **global.**
  Type can be used in any other definition.

Example Schema

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

- Local element declaration within a complex type
- („title" cannot be root element of documents)
- „name" and „type" as before
- „xsd:string" is built-in type of XML Schema

Example Schema

```
<xsd:element name="author"
                    minOccurs="1" maxOccurs="unbounded"/>
```

- Local element declaration
- "minOccurs", "maxOccurs" specify cardinality of "author" Elements in "BookType".
- Default: minOccurs=1, maxOccurs=1

Example Schema

```
<xsd:complexType>
        <xsd:sequence>
                <xsd:element name="first" type="xsd:string"/>
                <xsd:element name="last" type="xsd:string"/>
        <xsd:sequence>
   </xsd:complexType>
```

- Local, anonymous type definition
- May only be used inside the scope of the definition of BookType.
- The same syntax as for BookType.

Example Schema

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

- Local element declaratation
- Every book has exactly one "publisher" minOccurs, maxOccurs by default 1
- "anyType" is built-in Type
- "anyType" allows any content
- "anyType" is default type. Equivalent definition:

```
<xsd:element name="publisher" />
```

Example Schema

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://w3.org/2001/XMLSchema">
   <xsd:element name="book" type="BookType"/>
     <xsd:complexType name="BookType">
      <xsd:sequence>
           <xsd:element name="title" type="xsd:string"/>
             <xsd:element name="author">
                          minOccurs="1" maxOccurs="unbounded"/>
             <xsd:complexType>
                          <xsd:sequence> ... <xsd:sequence>
             </xsd:complexType>
          </xsd:element>
          <xsd:element name="publisher" type="xsd:anyType"/>
      </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

Valid Document?

```
<?xml version="1.0">
<book>
   <title>Die Wilde Wutz</title>
   <author><first>D.</first>
                  <last>K.</last></author>
   <publisher> Addison Wesley,
                        <state>CA</state>, USA
  </publisher>
</book>
```

Validate Document

```
<?xml version="1.0">
<book>
   <title>Die Wilde Wutz</title>
   <author><first>D.</first>
                  <last>K.</last></author>
   <publisher> Addison Wesley,
                       <state>CA</state>, USA
  </publisher>
</book>
```

Root is **book**

Validate Document

```
<?xml version="1.0">
<book>
   <title>Die Wilde Wutz</title>
   <author><first>D.</first>
                  <last>K.</last></author>
   <publisher> Addison Wesley,
                       <state>CA</state>, USA
  </publisher>
</book>
```

Exactly one title of Type **string**

Validate Document

```
<?xml version="1.0">
<book>
   <title>Die Wilde Wutz</title>
   <author><first>D.</first>
               <last>K.</last></author>
   <publisher> Addison Wesley,
                    <state>CA</state>, USA
  </publisher>
</book>
```

Subelements in right order

At least one **author** of Type **PersonType**

One **publisher** with arbitrary content.

Schema Validation

- Conformance Test
  - Result: "true" or "false"
  - Varying degree of strictness: strict, lax, skip
- Infoset Contribution (see Module 3)
  - Annotate Types
  - Set Default Values
  - Result: new instance of the data model
- Tools: Xerces (Apache)
- Theory: Graph Simulation Algorithms
- Validation is a-posteri; explicit - not implicit!

"Path to Schema" - Agenda

- Schema by Example (syntax, common cases)
- Validation
- Overview on builtin types/simple types
- Defining complex content
- Key constraints
- Namespaces
- Additional Aspects (not relevant for exam)

Pre-defined SimpleTypes

- Numeric Values

  Integer, Short, Decimal, Float, Double, HexBinary, ...

- Date, Timestamps, Periods

  Duration, DateTime, Time, Date, gMonth, ...

- Strings

  String, NMTOKEN, NMTOKENS, NormalizedString

- Others

  Qname, AnyURI, ID, IDREFS, Language, Entity, ...

- In summary, 44 pre-defined simple types

  Question: How many does SQL have?

Derived SimpleTypes

- Restrict domain

```
<xsd:simpleType name="MyInteger">
   <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="10000"/>
      <xsd:maxInclusive value="99999"/>
   </xsd:restriction>
 </xsd:simpleType>
```

- minInclusive, maxInclusive are "Facets"

Derived SimpleTypes

- Restriction by Pattern Matching
- Currencies have three capital letters

```
<xsd:simpleType name="Currency">
   <xsd:restriction base="xsd:string" >
      <xsd:pattern value="[A-Z]{3}"/>
   </xsd:restriction>
</xsd:simpleType>
```

Derived SimpleTypes

- Restriction by Enumeration

```
<xsd:simpleType name="Currency">
   <xsd:restriction base="xsd:string" >
      <xsd:enumeration value="CHF"/>
      <xsd:enumeration value="EUR"/>
      <xsd:enumeration value="GBP"/>
      <xsd:enumeration value="USD"/>
   </xsd:restriction>
</xsd:simpleType>
```

Derived SimpleTypes

- There are 15 different kinds of Facets
  - e.g., minExclusive, totalDigits, ...
- Most built-in types are derived from other built-in types by restriction
  - e.g., Integer is derived from Decimal
  - there are only 19 base types (out of 44)
- Ref: Appendix B of XML Schema Primer

Union Types

- Corresponds to the "|" in DTDs
  - (Variant Records in Pascal or Union in C)
- Valid instances are valid to any of the types

```
<xsd:simpleType name = "Potpurri" >
   <xsd:union memberTypes = "xsd:string intList"/>
</xsd:simpleType>
```

- Valid Instanzes

"fünfzig" "1 3 17" "wunderbar"   "15"

- Supported Facets
  - pattern, enumeration

List Types

- SimpleType for Lists
- Built-in List Types: IDREFS, NMTOKENS
- User-defined List Types

```
<xsd:simpleType name = "intList" >
   <xsd:list itemType = "xsd:integer" />
</xsd:simpleType>
```

- Items in instances are separed by whitespace

```
"5  -10  7        -20"
```

- Facets for Restrictions:
  - length, minLength, maxLength, enumeration

Facets of List Types

```
<xsd:simpleType name = "Participants" >
    <xsd:list itemType = "xsd:string" />
<xsd:simpleType>

<xsd:simpleType name = "Medalists" >
    <xsd:restriction base = "Participants" >
         <xsd:length value = "3" />
    </xsd:restriction>
</xsd:simpleType>
```

Attribute Declaration

- Attributes may only have a SimpleType
- SimpleTypes are, e.g., "string" (more later)
- Attribute declarations can be global
  - **Reuse declarations with ref**
- Compatible to Attribute lists in DTDs
  - Default values possible
  - Required and optional attributes
  - Fixed attributes
  - (In addition, there are "prohibited" attributes)

Attribute Declaration Example

```
<xsd:complexType name="BookType">
  <xsd:sequence> ... </xsd:sequence>
  <xsd:attribute name="isbn" type="xsd:string"
                       use="required" />
  <xsd:attribute name="price" type="xsd:decimal"
                       use="optional" />
  <xsd:attribute name="curr"   type="xsd:string"
                       fixed="EUR" />
  <xsd:attribute name="index" type="xsd:idrefs"
                       default="" />
</xsd:complexType>
```

Complex Type Definitions

- Empty content <a b="3"/>
- Simple content <a b="3">foo</a>
- Complex content <a b="3"><b>12</b></a>
  - Sequence
  - Choice
  - All
  - Element Groups
- Mixed content <a b="3">f<b>o</b>o</a>
  - *mixed* attribute for complex content
- Complex event constructors may be nested (e.g. sequence inside choice), but result must deterministic

Empty Content (using complexType without a definition)

```
<xsd:element name="price">
  <xsd:complexType>
    <xsd:attribute name="curr" type="xsd:string"/>
    <xsd:attribute name="val" type="xsd:decimal"/>
  </xsd:complexType>
</xsd:element>
```

- Valid Instance:

```
<price curr="USD" val="69.95" />
```

- Alternative: see next slide

Simple Elements + Attributes

```
<xsd:element name="price">
   <xsd:complexType>
      <xsd:simpleContent>
         <xsd:extension base= "xsd:decimal" >
            <xsd:attribute name="curr" type="xsd:string"/>
         </xsd:extension>
      </xsd:simpleContent>
   </xsd:complexType>   </xsd:element>
```

- Valid Instance:

  `<price curr="USD" >69.95</price>`

- Empty Content: String with zero length

Choice: "Union" in ComplexTypes

- A book has either an "author" or an "editor"

```
<xsd:complexType name = "Book" > <xsd:sequence>
   <xsd:choice>
      <xsd:element name = "author" type = "Person"
                              maxOccurs = "unbounded" />
      <xsd:element name = "editor" type = "Person" />
   </xsd:choice>
</xsd:sequence> </xsd:complexType>
```

Element Groups: Co-Occurence

If the book has an "editor", then the book also has a "sponsor":

```
<xsd:complexType name = "Book" > <xsd:sequence>
  <xsd:choice>
    <xsd:element name = "Author" type = "Person" .../>
    <xsd:group ref = "EditorSponsor" />
  </xsd:choice> </xsd:sequence> </xsd:complexType>

<xsd:group name = "EditorSponsor" > <xsd:sequence>
  <xsd:element name ="Editor" type="Person" />
  <xsd:element name = "Sponsor" type = "Org" />
</xsd:sequence> </xsd:group>
```

Optional Element Groups

- All or nothing; unordered content
- PubInfo has "name", "year", "city"
  or nothing at all

```
<xsd:complexType name = "PubInfo" > <xsd:sequence>
    <xsd:all>
      <xsd:element name = "name" type = "xsd:string"/>
      <xsd:element name = "year" type = "xsd:string" />
      <xsd:element name = "city" type = "xsd:string" />
    </xsd:all>  <!-- Attributdeklarationen -->
   </xsd:sequence> </xsd:complexType>
```

- No other element declarations allowed!!!
- maxOccurs must be 1

Attribute Groups

```
<xsd:attributeGroup name = "PriceInfo" >
   <xsd:attribute name = "curr" type = "xsd:string" />
   <xsd:attribute name = "val" type = "xsd:decimal" />
</xsd:attributeGroup>

<xsd:complexType name = "Book" >
   ...
   <xsd:attributeGroup ref = "PriceInfo" />
</xsd:complexType>
```

Derived Complex Types

- Two concepts of subtyping / inheritance
- Subtyping via Extension
  - Add Elements
  - Similar to inheritance in OO
- Subtyping via Restriction
  - e.g., constrain domains of types used
  - substituitability is preserved
- Further "features"
  - Constrain Sub-typing (~final)
  - Abstract Types

Subtyping via Extension

A "book" is a "publication"

```
<xsd:complexType name = "Publication"> <xsd:sequence>
    <xsd:element name = "title" type = "xsd:string" />
    <xsd:element name = "year" type = "xsd:integer" />
</xsd:sequence> </xsd:complexType>

<xsd:complexType name = "Book"> <xsd:complexContent>
  <xsd:extension base = "Publication" > <xsd:sequence>
      <xsd:element name = "author" type = "Person" />
  </xsd:sequence> </xsd:extension>
</xsd:complexContent> </xsd:complexType>
```

Subtyping by Extension

- A "bib" contains "Publications"

```
<xsd:element name = "bib" > <xsd:sequence>
  <xsd:element name = "pub" type = "Publication"
    maxOccurs = "unbounded"/>
</xsd:sequence> </xsd:element>
```

- "pub" Elements may be books!
- Instanzes have "xsi:type" Attribute

```
<bib> <pub xsi:type = "Book">
   <title>Wilde Wutz</title><year>1984</year>
   <author>D.A.K.</author> </pub>
</bib>
```

Subtyping via Restriction

The following restrictions are allowed:

- Instances of subtypes have default values
- Instances of subtypes are fixed (i.e., constant)
- Instances of subtypes have stronger types (e.g., string vs. anyType)
- Instances of subtypes have mandatory fields which optional in supertype
- Supertype.minOccurs <= Subtype.minOccurs
  Supertype.maxOccurs >= Subtype.maxOccurs

Subtyping via Restriction

```
<complexType name = "superType"> <sequence>
  <element name = "a" type = "string" minOccurs = "0" />
  <element name = "b" type = "anyType" />
  <element name = "c" type = "decimal" />
</sequence> <complexType>

<complexType name = "subType"> <complexContent>
  <restriction base = "superType"> <sequence>
     <element name = "a" type = "string" minOccurs = "0"
                                        maxOccurs = "0" />
     <element name = "b" type = "string" />
     <element name = "c" type = "decimal" />
  </sequence>  </restriction>
</complexContent> </complexType>
```

Substitution Groups

- Elements, which substitute global elem.
- E.g., "editor" is a "person"

```
<element name = "person" type = "string" />
<complexType name = "Book" > <sequence>
   <element ref = "person" /> ...
</sequence> </complexType>

<element name = "author" type = "string"
            substitutionGroup = "person" />
<element name = "editor" type = "string"
            substitutionGroup = "person" />
```

Abstract Elements and Types

- No instances exist
- Only instances of subtypes of substitions exist
- person in Book must be an author or editor

```
<element name = "person" type = "string"
                abstract = "true" />
<complexType name = "Book" > <sequence>
   <element ref = "person" /> ...
</sequence> </complexType>
...
```

Constrain Subtyping

- Corresponds to "**final" in Java**
- XML Schema is more clever!(?)
  - Constrain the kind of subtyping (extension, restriction, all)
  - Constrain the facets used

```
<simpleType name = "ZipCode" >
   <restriction base = "string">
     <length value = "5" fixed = "true" />
   </restriction> <simpleType>
<complexType name = "Book" final = "restriction" >
... </complexType>
```

- You may subtype ZipCode.
  But all subtypes have length 5.

Constrain Substituability

```
<complexType name = "Book" block = "all" >

... </complexType>
```

- It is possible to define subtypes of "Book"
- So, it is possible to reuse structe of "Book"
- But instances of subtypes of "Book" are NOT books themselves.
- (Now, things get really strange!)

Global vs. Local Declarations

- Instances of global element declarations are potential root elements of documents
- Global declarations can be referenced

```
<xsd:schema xmlns:xsd="...">
   <xsd:element name="book" type="BookType"/>
   <xsd:element name="comment"
                              type="xsd:string"/>
   <xsd:ComplexType name="BookType">
   ...   <xsd:element ref="comment" minOccurs="0"/>...
```

- Constraints
  - "ref" not allowed in global declarations
  - No "minOccurs", "maxOccurs" in global Decl.

"Path to Schema" - Agenda

- Schema by Example (syntax, common cases)
- Validation
- Overview on builtin types/simple types
- Defining complex content
- Key constraints
- Namespaces
- Additional Aspects (not relevant for exam)

Definition of keys

- Part of element declaration
- Special sub element "key"
  - Describes context in which unique values are required (*selector*)
  - Describes the key (*field*)
  - Composite Keys by using multiple "field"
- Selector und fields: XPath (next section)
- Document validation with keys
  - Evaluate "selector"- result: set of nodes
  - Evaluate "fields" on result aus: set of tuples
  - Check for duplicates in set of tuples

Syntax of Key Definition

- "isbn" is key of "books" in "bib"

```
<element name = "bib"> <complexType> <sequence>
    <element name ="book" maxOccurs = "unbounded">
      <complexType> <sequence> ... </sequence>
          <attribute name = "isbn" type = "string" />
    </complexType></element>
   </sequence></complexType>
<key name = "constraintX" >
         <selector xpath = "book" />  !! Get all books
         <field xpath = "@isbn" />      !! Get isbn
    </key>
</element>
```

References (Foreign Keys)

- Also part of element declaration
- Also "selector" and "field(s)"
  - Selector describes which part should be checked for referential integrity
  - "field" declarations compose "foreign key"
  - refer: gives the scope of the references (key constr.)
- Syntax (in our previous example):

```
<keyref name = "constraintY" refer = "constraintX" >
   <selector xpath = "book/references" />
   <field xpath = "@isbn" />
</keyref>
```

UNIQUE Constraints

- Same concept as in SQL
  - uniqueness, but no referentiability
- Syntax and concept almost the same as for keys

```
<unique name = "constraintZ">
   <selector xpath = "book" />
   <field xpath = "title" />
</unique>
```

- Part of the definition of an element

Null Values

- "not there" vs. "unknown" (i.e., null)
- "empty" vs. "unknown"
- Concept: Attribute "nil" with value "true"
- Only works for elements
- Schema definition: "NULL ALLOWED"

```
<xsd:element name = "publisher" type = "PubInfo"
                     nillable = "true" />
```

- Valid Instance with content "unknown"

```
<publisher xsi:nil = "true" />
```

- xsi: Namespace for predefined Instances
- Publisher may have other attributes, but content must be empty!

Namespaces and XML Schema

- Declare the Namespace of Elements?
- TargetNamespace for Global Elements
  - qualifies names of root elements
- elementFormDefault
  - qualifies names of local (sub-) elements
- attributeFormDefault
  - qualifies names of attributes

Namespaces in the Schema Definition

```
<xsd:schema  xmlns:xsd="http://w3.org/2001/XMLSchema"
               xmlns:bo="http://www.Book.com"
               targetNamespace="http://www.Book.com">
   <xsd:element name="book" type="bo:BookType"/>
    <xsd:complexType name="BookType" >
    ... </xsd:complexType>
</xsd:schema>
```

- "book" und "BookType" are part of targetNamespace

Namespaces in Schema Definition (2)

```
<schema  xmlns = "http://w3.org/2001/XMLSchema"
                xmlns:bo="http://www.Book.com"
                targetNamespace="http://www.Book.com" >
    <element name="book" type = "bo:BookType" />
    <complexType name="BookType" >
    ... </complexType>
</schema>
```

Make the namespace for schema the default namespace

Namespaces in Schema Definition (3)

```
<xsd:schema  xmlns:xsd="http://w3.org/2001/XMLSchema"
             xmlns ="http://www.Book.com"
             targetNamespace="http://www.Book.com" >
  <xsd:element name="book" type = "BookType" />
  <xsd:complexType xsd:name="BookType" >
  ... </xsd:complexType>
</xsd:schema>
```

- Target "www.Book.com" as Default Namespace

Instances of www.Book.com

```
<bo:book xmlns:bo = "http://www.Book.com" >
   ...
</bo:book>
```

- Valid according to all three schemas!

Unqualified "Locals"

- Local Declarations are not qualified

```
<bo:book  xmlns:bo = "http://www.Book.com"
          price = "69.95" curr = "EUR" >
   <title>Die wilde Wutz</title> ...
</bo:book>
```

- Valid Instance: globally qualifed, locally not
- Even works within Schema

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

- Full flexibility to control use of namespaces

Qualified Sub-elements

```
<schema  xmlns = "http://w3.org/2001/XMLSchema"
                xmlns:bo="http://www.Book.com"
                targetNamespace="http://www.Book.com" >
                elementFormDefault="qualified"
    <element name="book" type = "bo:BookType" />
    <complexType name="BookType" > <sequence>
       <element name = "title" type = "string" />
       <element name = "author" />  <sequence>
          <element name = "vname" type = "string" />
          <element name = "nname" type = "string" />
       </sequence> </sequence> </complexType>
</schema>
```

Valid Instances

```
<bo:book  xmlns:bo = "http://www.Book.com"
   <bo:title>Die wilde Wutz</bo:title>
    <bo:author><bo:vname>D.</bo:vname>
   <bo:nname>K.</bo:nname></bo:author>
</bo:book>
```

```
<book  xmlns = "http://www.Book.com"
    <title>Die wilde Wutz</title>
     <author><vname>D.</vname>
                  <nname>K.</nname></author>
</book>
```

Qualified Attributes

- Enforce Qualified Attributes

  attributeFormDefault = "qualified" in Element definition

- Enforce that certain attributes must be qualified

  <attribute name = "..." type = "..." form = "qualified" />

  (Analogous, enforce that Sub-elements must be qualified)

Schema Location in Instance

- Declare within an XML document, where to find the schema that should validate that document
- Declare "**target Namespace**"
- Declare **URI** of Schema

```
<book  xmlns = "http://www.Book.com"
   xmlns:xsi = "http://w3.org/XMLSchema-instance"
   xsi:schemaLocation = "http://www.Book.com
                   http://www.book.com/Book.xsd"
   ...
</book>
```

- This is not enforced!
  Validation using other Schemas is legal.

Composition of Schemas

- Construct libraries of schemas
- Include a Schema
  - Parent and child have the same Target Namespace
  - Only Parent used for Validation
- Redefine: Include + Modify
  - Again, parent and child have the same Target Namespace
- Include individual types from a schema

```
<element ref = "lib:impType" />
```

Summary

- XML Schema is very powerful
  - simple Types and complex Types
  - many pre-defined types
  - many ways to derive and create new types
  - adopts database concepts (key, foreign keys)
  - full control and flexibility
  - fully aligned with namespaces and other XML standards
- XML Schema is too powerful?
  - too complicated, confusing?
  - difficult to implement
  - people use only a fraction anyway
- XML Schema is very different to what you know!
  - the devil is in the detail

XML vs. OO

- Encapsulation
  - OO hides data
  - XML makes data explicit
- Type Hierarchy
  - OO defines superset / subset relationship
  - XML shares structure; *set* rel. make no sense
- Data + Behavior
  - OO packages them together
  - XML separates data from its interpretation

XML vs. Relational

- Structural Differences
  - Tree vs. Table
  - Heterogeneous vs. Homegeneous
  - Optional vs. Strict typing
  - Unnormalized vs. Normalized data
- Some commonalities
  - Logical and physical data independance
  - Declarative semantics
  - Generic data model

XML vs RDF

- Differences
  - Graph vs Tree
  - Order vs Unordered
  - Metadata as constraint vs Metadata as extension
- Communalities
  - Separation of Data and Metadata
  - Formal Semantics
  - Declarative Processing