XML Meta-documents – DTDs, Schema

CS 431 – February 14, 2007

Carl Lagoze – Cornell University

A little context

Semantic Web

Traditional Library
Central control
Uniform expertise

Traditional Web
Distributed, interlinked
Viewable Documents

RDF
Semantic
Relationships

OWL
Concept
Building

Xpath
Data
Decomposition

XSLT
Data
Transformation

Schema
Type Definition

Namespaces
Concept
Integration

DTD
Tag Sets

XML
Markup Syntax

URIs
Name Convention

HTTP
Access Method

Namespaces

- How the web does work
  - Individually created documents linked by ambiguous references
- How the web should work
  - Global database of knowledge
- Key to doing that is to permit distributed knowledge creation and lazy integration
- Problems
  - Vocabulary collisions
  - Joins
- Namespaces
  - Build on URI notion
  - Make it possible to uniquely qualify intra-document name collisions

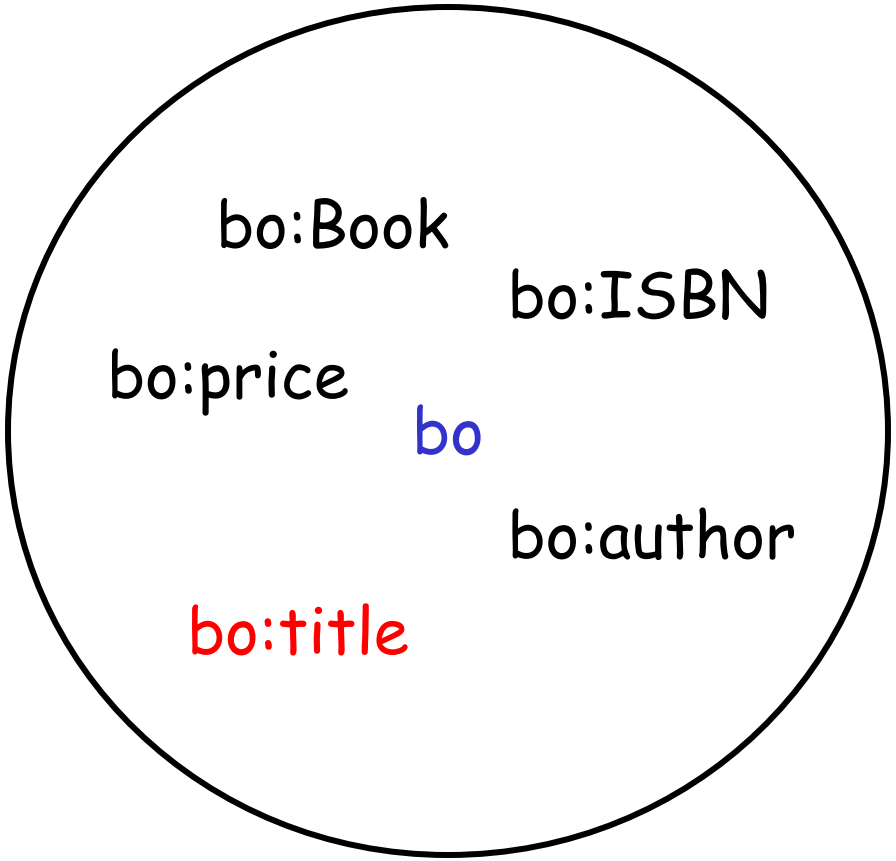
```
<?xml version="1.0" encoding="UTF-8"?>
<Book>
  <ISBN>0743204794</ISBN>
  <author>Kevin Davies</author>
  <title>Cracking the Genome</title>
  <price>20.00</price>
</Book>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<html>
<head>
  <title>My home page</title>
</head>
  <body>
<p>My hobby</p><p>My books</p>
</body>
</html>
```

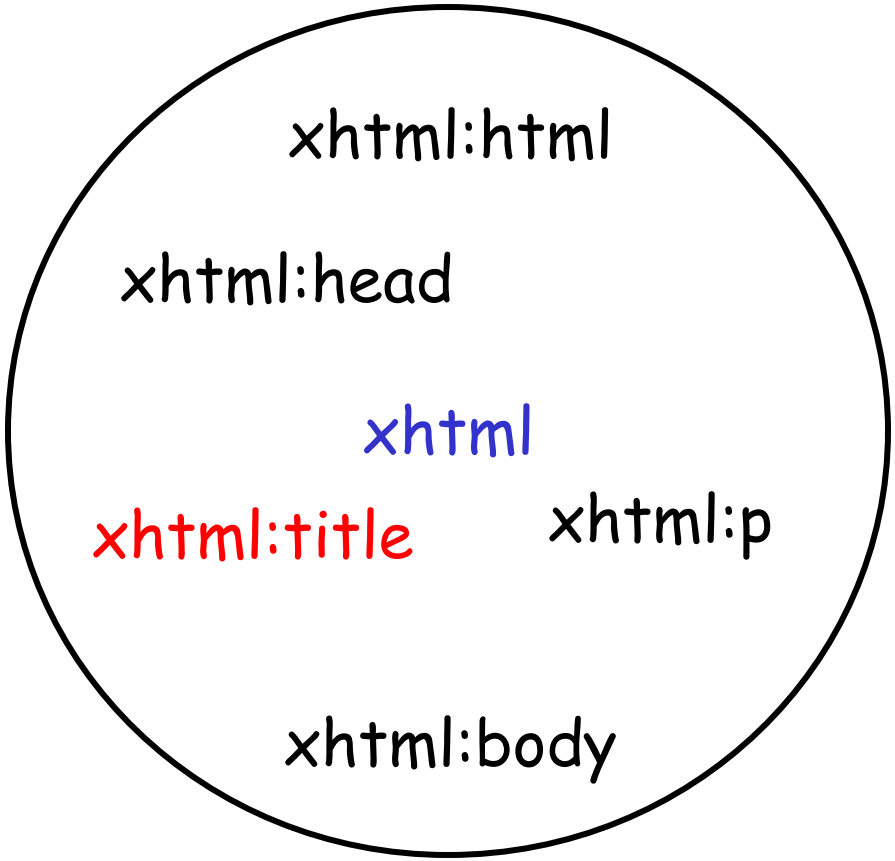
```
<?xml version="1.0" encoding="UTF-8"?>
<html>
<head>
  <title>My home page</title>
</head>
  <body>
<p>My hobby</p>
<p>My books
 <Book>
   <ISBN>0743204794</ISBN>
   <author>Kevin Davies</author>
   <title>Cracking the Genome</title>
   <price>20.00</price>
 </Book>
</p>
</body>
</html>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xhtml:html>
<xhtml:head>
  <xhtml:title>My home page</xhtml:title>
</xhtml:head>
  <xhtml:body>
<xhtml:p>My hobby</xhtml:p>
<xhtml:p>My books
 <bo:Book>
   <bo:ISBN>0743204794</bo:ISBN>
   <bo:author>Kevin Davies</bo:author>
   <bo:title>Cracking the Genome</bo:title>
   <bo:price>20.00</bo:price>
 </bo:Book>
</xhtml:p>
</xhtml:body>
</xhtml:html>
```

XML – namespaces



vocabulary bo



vocabulary xhtml

But who guarantees uniqueness of prefixes?

XML – namespaces

- Give prefixes only local relevance in an instance document
- Associate local prefix with global namespace name
  - ⇒ a unique name for a namespace
  - ⇒ uniqueness is guaranteed by using a URI (preferably URN) in domain of the party creating the namespace
  - ⇒ doesn't have any meaning, i.e. doesn't have to resolve into anything

An XML namespace is a collection of names, identified by a URI reference, which are used in XML documents as element types and attribute names.

```
<?xml version="1.0" encoding="UTF-8"?>
<xhtml:html
  xmlns:xhtml="http://www.w3c.org/1999/xhtml"
  xmlns:bo="http://www.nogood.com/Book">
<xhtml:head>
  <xhtml:title>My home page</xhtml:title>
</xhtml:head>
  <xhtml:body>
<xhtml:p>My hobby</xhtml:p>
<xhtml:p>My books
 <bo:Book>
   <bo:ISBN>0743204794</bo:ISBN>
   <bo:author>Kevin Davies</bo:author>
     ................
 </bo:Book>
</xhtml:p>
</xhtml:body>
</xhtml:html>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<html
  xmlns="http://www.w3c.org/1999/xhtml"
  xmlns:bo="http://www.nogood.com/Book">
<head>
  <title>My home page</xhtml:title>
</head>
  <body>
<p>My hobby</xhtml:p>
<p>My books
 <bo:Book>
   <bo:ISBN>0743204794</bo:ISBN>
   <bo:author>Kevin Davies</bo:author>
     ……
 </bo:Book>
</p>
</body>
</html>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<html
  xmlns="http://www.w3c.org/1999/xhtml">
<head>
  <title>My home page</title>
</head>
  <body>
<p>My hobby</p>
<p>My books
 <bo:Book
xmlns:bo="http://www.nogood.com/Book">
   <bo:ISBN>0743204794</bo:ISBN>
   <bo:author>Kevin Davies</bo:author>
     ……
 </bo:Book>
</p>
</body>
</html>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<html
  xmlns="http://www.w3c.org/1999/xhtml">
<head>
  <title>My home page</title>
</head>
  <body>
<p>My hobby</p>
<p>My books
 <Book xmlns="http://www.nogood.com/Book">
   <ISBN>0743204794</bo:ISBN>
   <author>Kevin Davies</bo:author>
     ................
 </Book>
</p>
</body>
</html>
```

What do namespace URI's point to?

- There are lots of opinions on this subject!
- The "abstraction" camp
  - A namespace URI is the id for a concept
  - It shouldn't resolve to anything
  - Example – my SSN #, it doesn't point to Carl Lagoze but to the concept of Carl Lagoze with different facets can be (ab)used in numerous concepts
- The "orthodox" camp
  - It should resolve to a schema (xml schema)
- The "liberal" camp
  - It should resolve to many things
  - RDDL (http://www.rddl.org)

- Reality: Read Wikipedia about this if you want to see how ambiguous this all is: http://en.wikipedia.org/wiki/Uniform_Resource_Identifier
- Moral: Interoperability is hard once you move beyond the basics

From well-formedness to validity

- Goal of standards is interoperability
  - Allow different communities to share data
  - Requires meta-level understanding
- Levels of XML interoperability
  - Well-formedness
    - Base-level syntax
    - Properly formed tree
  - Validity
    - Structure of tree
    - Adherence to tree constraint rules

Tree constraint languages

- Document Type Definitions (DTDs)
- XML Schema
- Schematron
- RELAX NG

DTD – Document Type Definition

- Artifact of XML's roots in SGML
- Defines validity XML document
- Useful for interoperability among document instances

Constructing a DTD - Elements

- Declaration of element
- Types
  - EMPTY – no children, only attributes
  - ELEMENT – only children, no text
  - MIXED – children and text (PCDATA)
  - ANY
  - Content Model
    - <!Element PersonName (First, Middle, Last)>
    - <!Element Fruit (Apple | Orange)>
    - <!Element FruitBasket (Cherry+, Pineapple?, Orange*)>
    - <!Element Mixture (#PCDATA | ItemA | ItemB)*>

Constructing a DTD - Attributes

- Attributes are a way of associating properties or refinements with elements
- Syntax
  - <!ATTLIST element-name attribute-name attribute-type default-value>
- Standard attribute types
  - CDATA character data
  - ID unique identifier in XML document
  - (en1|en2|..) enumerated list
- Default values
  - value the default value
  - #REQUIRED attribute must be included with element
  - #IMPLIED attributed does not have to be included

Associating a DTD with XML source

- Can be internal:
  - <!DOCTYPE root-element [element-declarations]>
- or external:
  - <!DOCTYPE root-element SYSTEM “file-name”>
  - <!DOCTYPE root-element PUBLIC “tag-name” “url”>

DTD example

- XML Source
  - http://www.cs.cornell.edu/courses/CS431/2007sp/examples/xml_schema/invoice.xml
- DTD
  - http://www.cs.cornell.edu/courses/CS431/2007sp/examples/xml_schema/invoice.dtd

XML – XML schema

Problems with XML DTD's:

- DTD's are not extensible: Can import declarations from other DTD's (external entity) . Can not inherit or refine those declarations.

- A document must be valid according to 1 DTD: prevents building on elements from different DTDs

- Limited support of namespaces

Problems with DTD's (cont)

- Poor data typing: DTDs are mainly about "text". No provision for numeric data types, dates, times, strings conforming to regular expressions, URI's, …
- DTD's are defined in non-XML syntax => Can not use XML tools!

XML Schema

- W3C Recommendation
  - http://www.w3.org/XML/Schema#dev
- Very complex standard
  - Fortunately there is a primer
  - http://www.w3.org/TR/xmlschema-0/
  - Some really good online materials: http://www.w3schools.com/schema/

Interoperability & Extensibility

- XML schema are building blocks to interoperability between multiple data sources
  - Enforces shared markup
    - E.g., a <person> must have a <firstname> and <lastname>
  - Enforces shared types
    - E.g. <person age="18"> - age must be a number between 0 and 120
- XML schema are building blocks for extensibility
  - Reuse
  - Type derivation

Expressed in XML

- All tags are in the http://www.w3.org/2001/XMLSchema namespace
- Can be manipulated by standard XML tools

Simple Schema Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.example.org/SimpleSchema"
    elementFormDefault="qualified">
    <xs:element name="person">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="firstName"/>
                <xs:element name="lastName"/>
            </xs:sequence>
        </xs:complexType>

    </xs:element>
</xs:schema>
```

http://www.cs.cornell.edu/courses/CS431/2007sp/examples/xml_schema/SimpleSchema1.xsd

http://www.cs.cornell.edu/courses/CS431/2007sp/examples/xml_schema/Simple1.xml

Complex Types (contain sub-tree)

- Define the structure of the sub-tree within the element

```
<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
<xs:element name="employee" type="personinfo"/>
<xs:element name="student" type="personinfo"/>
<xs:element name="member" type="personinfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Controls on complex types

- sequence – specific order
- all – any order
- choice – only one

- cardinality – minOccurs, maxOccurs

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string"
      maxOccurs="10" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Simple Types (no sub-tree within)

- Define an element to have a simple value
  - Constrain value to a specific data type
  - Set of data types in xs namespace
- Syntax
  - <xs:element name="xxx" type="yyy"/>

- Examples
  - <xs:element name="lastname" type="xs:string"/>
  - <xs:element name="age" type="xs:number"/>
  - <xs:element name="age" type="xs:date"/>

Facets for simple values

- Restrictions on values within type context
  - E.g. range for an integer value, controlled set for string
- Examples

```
<xs:element name="age">

<xs:simpleType>
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="16"/>
    <xs:maxInclusive value="34"/>
  </xs:restriction>
</xs:simpleType>

</xs:element>
```

```
<xs:element name="car">

<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:enumeration value="Audi"/>
    <xs:enumeration value="Mercedes"/>
    <xs:enumeration value="Volvo"/>
  </xs:restriction>
</xs:simpleType>

</xs:element>
```

String types and patterns

```
<xs:element name="initials">

<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]"/>
  </xs:restriction>
</xs:simpleType>

</xs:element>
```

Another example – Restrictions on values

http://www.cs.cornell.edu/courses/CS431/2007sp/examples/xml_schema/SimpleSchema2.xsd

http://www.cs.cornell.edu/courses/CS431/2007sp/examples/xml_schema/Simple2.xml

Mixed Content

```
<xs:element name="letter">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="orderid" type="xs:positiveInteger"/>
      <xs:element name="shipdate" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
<letter>
Dear Mr.<name>John Smith</name>.
Your order <orderid>1032</orderid>
will be shipped on <shipdate>2001-07-13</shipdate>.
</letter>
```

Declaring attributes

- Define type
  - xs:string
  - xs:decimal
  - xs:integer
  - xs:boolean
  - xs:date
  - xs:time
- Define optional or required

```
<xs:attribute name="lang" type="xs:string" use="optional"/>
```

Use of attributes

- Always a complex type

```
<xs:element name="shoesize" type="shoetype"/>

<xs:complexType name="shoetype">
  <xs:simpleContent>
    <xs:extension base="xs:integer">
      <xs:attribute name="country" type="xs:string" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

```
<xsd:complexType name="PurchaseOrderType">
  <xsd:sequence>
    <xsd:element name="shipTo" type="USAddress"/>
    <xsd:element name="billTo" type="USAddress"/>
    <xsd:element ref="comment" minOccurs="0"/>
    <xsd:element name="items"  type="Items"/>
  </xsd:sequence>
  <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>
```

Example with attributes

- Memo Schema
  - http://www.cs.cornell.edu/courses/CS431/2007sp/examples/xml_schema/memo.xsd
- Instance Document
  - http://www.cs.cornell.edu/courses/CS431/2007sp/examples/xml_schema/memo.xml

Type Reuse
Extending a complex type

- Add values to sequence

```
<xs:element name="employee" type="fullpersoninfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="fullpersoninfo">
  <xs:complexContent>
    <xs:extension base="personinfo">
      <xs:sequence>
        <xs:element name="address" type="xs:string"/>
        <xs:element name="city" type="xs:string"/>
        <xs:element name="country" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Type Reuse:
Import a schema

- Syntax
  - <import namespace=URI schemaLocation=URL />
  - Note that this will introducing yet another namespace

Type Reuse Example

- Address schema
  - http://www.cs.cornell.edu/courses/CS431/2007sp/examples/xml_schema/address.xsd
- Person schema
  - http://www.cs.cornell.edu/courses/CS431/2007sp/examples/xml_schema/person.xsd
- Instance document
  - http://www.cs.cornell.edu/courses/CS431/2007sp/examples/xml_schema/person.xml