Structured Web Documents in XML

Adapted from slides from Grigoris Antoniou and Frank van Harmelen

Outline

Role of XML in the Semantic Web

- The Semantic Web involves ideas and languages at a fairly abstract level, e.g.: for defining ontologies, publishing data using them
- XML is a
  - Source of many key SW concepts & technology bits;
  - Potential alternative for sharing data that newer schemes must improve on; and
  - Common serialization for SW data

To paraphrase Jamie Zawinski

Some people, when confronted with a problem, think, "I know, I'll use XML."

Now they have two problems.

“Some people, when confronted with a problem, think "I know, I'll use regular expressions." Now they have two problems.”
-- Wikiquote

History

- XML's roots are in SGML
  - Standard Generalized Markup Language
  - A *metalanguage* for defining document markup languages
  - Extensible, but complicated, verbose, hard to parse, …
- HTML was defines using SGML, ~1990 by TBL
  - A markup language, not a markup *metalanguage*
- XML proposal to W3C in July 1996
  - Simplified SGML to greatly expand power and flexibility of Web
- Evolving series of W3C recommendations
  - Current recommendation: XML 5 (2008)

An HTML Example

```
<h2>Nonmonotonic Reasoning: Context-
   Dependent Reasoning</h2>
<i>by <b>V. Marek</b> and
   <b>M. Truszczynski</b></i><br>
Springer 1993<br>
ISBN 0387976892
```

The Same Example in XML

```
<book>
  <title>Nonmonotonic Reasoning: Context-Dependent
  Reasoning</title>
  <author>V. Marek</author>
  <author>M. Truszczynski</author>
  <publisher>Springer</publisher>
  <year>1993</year>
  <ISBN>0387976892</ISBN>
</book>
```

HTML versus XML: Similarities

- Both use **tags** (e.g. <h2> and </year>)
- Tags may be nested (tags within tags)
- Human users can read and interpret both HTML and XML representations "easily"

… **But how about machines?**

Problems Interpreting HTML Documents

Problems for an intelligent agent trying to retrieve the names of the authors of the book

- Authors’ names could appear immediately after the title
- or immediately after the word *“by”* (or *“van”* if it’s in Dutch)
- Are there two authors or just one, called “*V. Marek and M. Truszczynski*”?

```
<h2>Nonmonotonic Reasoning: Context-
Dependent Reasoning</h2>
<i>by <b>V. Marek</b> and <b>M.
Truszczynski</b></i><br>
Springer 1993<br>
ISBN 0387976892
```

HTML vs XML: Structural Information

- HTML documents don’t carry **structured information**: pieces document and their relations
- XML more easily accessible to machines since
  - Every piece of information is described
  - Relations defined through nesting structure
  - E.g., **<author>** tags appear within **<book>** tags, so they describe properties of a particular book

HTML vs XML: Structural Information

- A machine processing the XML document can assume (deduce/infer) that
  - **author** element refers to enclosing **book** element
  - Without using background knowledge, *proximity* or other heuristics
- XML allows definition of constraints on values
  - E.g., a year must be a integer of four digits

HTML vs. XML: Formatting

- HTML representation provides more than XML representation:
  - Formatting of the document is described
- Main use of an HTML document is to display information: it must define formatting
- **XML: separation of content from display**
  - same information can be displayed in different ways
  - Presentation specified by documents using other XML standards (CSS, XSL)

HTML vs. XML: Another Example

In HTML

```
<h2>Relationship matter-energy</h2>
<i> E = M × c^2 </i>
```

In XML

```
<equation>
    <gloss>Relationship matter energy </gloss>
    <leftside> E </leftside>
    <rightside> M × c^2 </rightside>
</equation>
```

HTML vs. XML: Different Use of Tags

- All HTML documents use the same tags
  - HTML tags come from a finite, pre-defined collection
  - Define properties for display: font, color, lists …
- XML documents can use completely different tags
  - XML tags not fixed: user definable tags
  - XML is a *meta markup language*, i.e., a language for defining markup languages

XML Vocabularies

- Applications must agree on common vocabularies to communicate and collaborate
- Communities and business sectors define their specialized vocabularies
  - mathematics (MathML)
  - bioinformatics (BSML)
  - human resources (HRML)
  - Syndication (RSS)
  - Vector graphics (SVG)
  - …

Outline

The XML Language

An XML document consists of

- A **prolog**
- A number of **elements**
- An optional **epilog** (not discussed, not used much)

Prolog of an XML Document

The prolog consists of

- An XML declaration and
- An optional reference to external structuring documents

```
<?xml version="1.0" encoding="UTF-16"?>

<!DOCTYPE book SYSTEM "book.dtd">
```

XML Elements

- Elements are the *things* the XML document talks about
  - E.g., books, authors, publishers, …
- An element consists of:
  - An opening tag
  - The content
  - A closing tag

<lecturer> David Billington </lecturer>

XML Elements

- Tag names can be chosen almost freely
- First character must be a letter, underscore, or colon
- No name may begin with the string “xml” in any combination of cases
  - E.g. “Xml”, “xML”

Content of XML Elements

- Content is what’s between the tags
- It can be text, or other elements, or nothing

```
<lecturer>
  <name>David Billington</name>
  <phone> +61 – 7 – 3875 507 </phone>
</lecturer>
```

- If there is no content, then element is called empty; it can be abbreviated as follows:

```
<lecturer/>  =  <lecturer></lecturer>
```

XML Attributes

- An empty element isn’t necessarily meaningless
  - It may have properties expressed as ***attributes***
- An **attribute** is a name-value pair inside the opening tag of an element

<lecturer
  name="David Billington"
  phone="+61 – 7 – 3875 507" />

XML Attributes: An Example

```
<order  orderNo="23456“
        customer="John Smith"
        date="October 15, 2017" >
  <item itemNo="a528" quantity="1" />
  <item itemNo="c817" quantity="3" />
</order>
```

The Same Example without Attributes

```
<order>
  <orderNo>23456</orderNo>
  <customer>John Smith</customer>
  <date>October 15, 2017</date>
  <item>
      <itemNo>a528</itemNo>
      <quantity>1</quantity>
  </item>
  <item>
      <itemNo>c817</itemNo>
      <quantity>3</quantity>
      </item>
</order>
```

XML Elements vs. Attributes

- Attributes can be replaced by elements
- When to use elements and when attributes is a mostly matter of taste
- **But attributes <u>cannot</u> be nested**

Further Components of XML Docs

- **Comments**
  - A piece of text that is to be ignored by parser

```
<!-- This is a comment -->
```

- **Processing Instructions (PIs)**
  - Define procedural attachments

```
<?stylesheet type="text/css“
  href="mystyle.css"?>
```

Well-Formed XML Documents

Constraints on syntactically correct documents:

- Only one outermost element (**root element**)
- Each element contains opening and corresponding closing tag (except self-closing tags like <foo/>)
- Tags may not overlap

  <author><name>Lee Hong</author></name>
- Attributes within an element have unique names
- Element and tag names must be permissible

  e.g.: can’t use strings beginning with digit "2ndbest"

The Tree Model of XML Docs

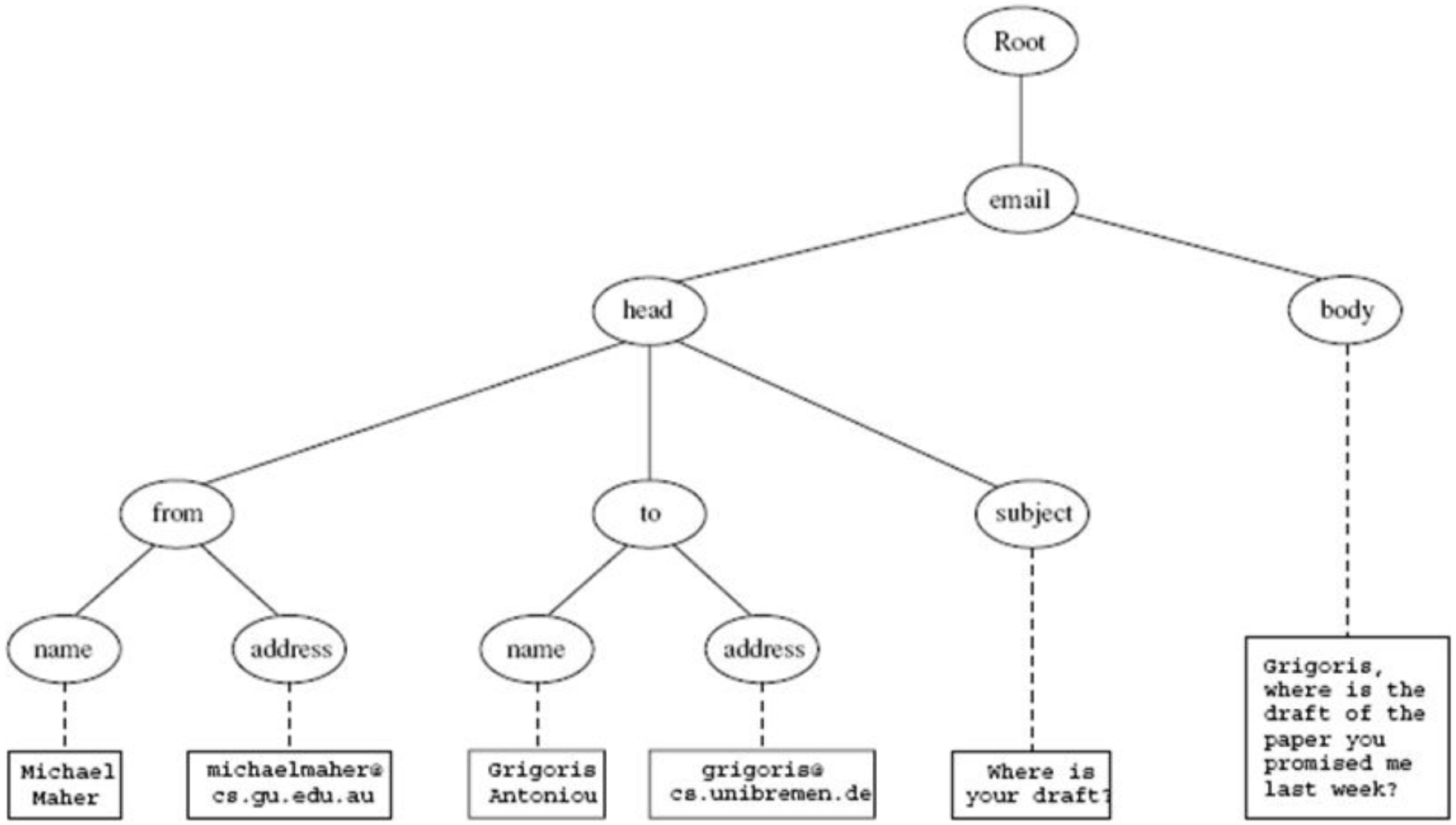
The tree representation of an XML document is an **ordered** labeled tree:

- There is exactly one root
- There are no cycles
- Each non-root node has exactly one parent
- Each node has a label.
- The order of elements is important
- … but the order of attributes is not

Tree Model of XML Documents

```
<email>
  <head>
      <from name="Michael Maher"
            address="michaelmaher@cs.gu.edu.au" />
      <to name="Grigoris Antoniou"
            address="grigoris@cs.unibremen.de" />
      <subject>Where is your draft?</subject>
  </head>
  <body>
      Grigoris, where is the draft of the paper you
     promised me last week?
  </body>
</email>
```

Tree Model of XML Documents


Root
email
head
body
from
to
subject
name
address
name
address
Michael Maher
michaelmaher@ cs.gu.edu.au
Grigoris Antoniou
grigoris@ cs.unibremen.de
Where is your draft?
Grigoris, where is the draft of the paper you promised me last week?

Outline

Structuring XML Documents

- Some XML documents must follow constraints defined in a “template” that can…
  - define all *element* and *attribute names* that may be used
  - define the *structure*
    - what values an attribute may take
    - which elements may or must occur within other elements, etc.
- If such structuring information exists, the document can be **validated**

Structuring XML Documents

- An XML document is **valid** if
  - it is well-formed XML
  - respects the structuring information it uses
- Ways to define structure of XML documents:
  - **DTDs** (*Document Type Definition*) came first, was based on SGML's approach
  - **XML Schema** (aka *XML Schema Definition*, XSD) is more recent and expressive
  - RELAX NG and DSDs are two alternatives

DTD: Element Type Definition

```
<lecturer>
        <name>David Billington</name>
        <phone> +61 – 7 – 3875 507 </phone>
</lecturer>
```

DTD for above element (and all **lecturer** elements):

```
<!ELEMENT lecturer (name, phone) >
<!ELEMENT name (#PCDATA) >
<!ELEMENT phone (#PCDATA) >
```

The Meaning of the DTD

```
<!ELEMENT lecturer (name, phone) >
<!ELEMENT name (#PCDATA) >
<!ELEMENT phone (#PCDATA) >
```

- The element types **lecturer**, **name**, and **phone** may be used in the document
- A **lecturer** element contains a **name** element and a **phone** element, in that order (*sequence*)
- A **name** element and a **phone** element may have any content
  - In DTDs, **#PCDATA** is only atomic element type and stands for "*parsed character data*"

Disjunction in Element Type Definitions

- We say that **lecturer** elements contains *either* a **name** element *or* a **phone** element like:

  <!ELEMENT lecturer (name | phone)>

- A **lecturer** element contains a **name** element and a **phone** element in *any order*

  **<!ELEMENT
     lecturer((name,phone)|(phone,name))>**

- Do you see a problem with this approach?

Example of an XML Element

```
<order orderNo="23456"
            customer="John Smith"
            date="October 15, 2017">
    <item itemNo="a528" quantity="1” />
    <item itemNo="c817" quantity="3” />
</order>
```

The Corresponding DTD

```
<!ELEMENT order (item+)>
<!ATTLIST order
        orderNo     ID        #REQUIRED
        customer    CDATA     #REQUIRED
        date        CDATA     #REQUIRED >

<!ELEMENT item EMPTY>
<!ATTLIST item
        itemNo      ID        #REQUIRED
        quantity    CDATA     #REQUIRED
        comments    CDATA     #IMPLIED >
```

Comments on the DTD

- The **item** element type is defined to be empty
  - i.e., it can contain no elements
- **+** (after **item)** is a **cardinality operator**:
  - It specifies how many item elements can be in an order
  - **?**: zero times or once
  - *****: zero or more times
  - **+**: one or more times
  - No cardinality operator: once

```
<!ELEMENT order (item+)>
<!ATTLIST
   order orderNo ID #REQUIRED
   customer CDATA #REQUIRED
   date CDATA #REQUIRED >
<!ELEMENT item EMPTY>
<!ATTLIST
    item itemNo ID #REQUIRED
    quantity CDATA #REQUIRED
    comments CDATA #IMPLIED >
```

Comments on the DTD

- In addition to defining elements, we define attributes
- This is done in an **attribute list** containing:
  - Name of the element type to which the list applies
  - A list of triples of attribute name, attribute type, and value type
- *Attribute name*: A name that may be used in an XML document using a DTD

DTD: Attribute Types

- Similar to predefined data types, but limited …
- The most important types are
  - **CDATA**, a string (sequence of characters)
  - **ID**, a name that is *unique* across the entire XML document (~ DB key)
  - **IDREF**, reference to another element with ID attribute carrying same value as IDREF attribute (~ DB foreign key)
  - **IDREFS**, a series of IDREFs
  - **(v1| . . . |vn)**, an enumeration of all possible values
- Limitations: no dates, number ranges, etc.

DTD: Attribute Value Types

- **#REQUIRED**
  - Attribute must appear in every occurrence of the element type in the XML document
- **#IMPLIED**
  - The appearance of the attribute is optional
- **#FIXED "value"**
  - Every element must have this attribute
- **"value"**
  - This specifies the default value for the attribute

Referencing with IDREF and IDREFS

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

An XML Document Respecting the DTD

```
<family>
    <person id="bob" mother="mary" father="peter">
        <name>Bob Marley</name>
    </person>
    <person id="bridget" mother="mary">
        <name>Bridget Jones</name>
    </person>
    <person id="mary" children="bob bridget">
        <name>Mary Poppins</name>
    </person>
    <person id="peter" children="bob">
        <name>Peter Marley</name>
    </person>
</family>
```

Email Element DTD 1/2

```
<!ELEMENT email (head,body)>
<!ELEMENT head (from,to+,cc*,subject)>
<!ELEMENT from EMPTY>
<!ATTLIST from
    name        CDATA   #IMPLIED
    address     CDATA   #REQUIRED>
<!ELEMENT to EMPTY>
<!ATTLIST to
    name     CDATA  #IMPLIED
    address  CDATA  #REQUIRED>
```

Email Element DTD 2/2

```
<!ELEMENT cc EMPTY>
<!ATTLIST cc
        name      CDATA     #IMPLIED
        address   CDATA     #REQUIRED>
<!ELEMENT subject (#PCDATA) >
<!ELEMENT body (text,attachment*) >
<!ELEMENT text (#PCDATA) >
<!ELEMENT attachment EMPTY >
<!ATTLIST attachment
        encoding   (mime|binhex)     "mime"
        file           CDATA        #REQUIRED>
```

Outline

(1) Introduction

(2) Description of XML

(3) Structuring

- DTDs
- **XML Schema**

(4) Namespaces

(5) Accessing, querying XML documents: XPath

(6) Transformations: XSLT

XML Schema (XSD)

- XML Schema is a significantly richer language for defining the structure of XML documents
- Syntax based on XML itself, so separate tools to handle them not needed
- Reuse and refinement of schemas => can expand or delete existing schemas
- Sophisticated set of **data types**, compared to DTDs, which only supports strings
- XML Schema recommendation published by W3C in 2001, version 1.1 in 2012

XML Schema

- An XML schema is an element with an opening tag like

  <schema
  "http://www.w3.org/2000/10/XMLSchema"
  version="1.0">

- Structure of schema elements
  - Element and attribute types using data types

Element Types

```
<element name="email"/>
<element name="head“
    minOccurs="1“
    maxOccurs="1"/>
<element name="to" minOccurs="1"/>
```

Cardinality constraints:

- **minOccurs="x"** (default value 1)
- **maxOccurs="x"** (default value 1)
- Generalizations of *,?,+ offered by DTDs

Attribute Types

```
<attribute name="id" type="ID“ use="required"/>
<attribute name="speaks" type="Language"
    use="default" value="en"/>
```

- Existence: **use="x",** where **x** may be **optional** or **required**
- Default value: **use="x" value="..."**, where **x** may be **default** or **fixed**

Data Types

- Many **built-in data types**
  - Numerical data types: **integer**, **short,** etc.
  - String types: **string**, **ID**, **IDREF**, **CDATA,** etc.
  - Date and time data types: **time**, **month,** etc.
- Also **user-defined data types**
  - **simple data types**, which can’t use elements or attributes
  - **complex data types**, which can use them

Complex Data Types

Complex data types are defined from existing data types by defining some attributes (if any) and using:

- **sequence**, a sequence of existing data type elements (order is important)
- **all**, a collection of elements that must appear (order is not important)
- **choice**, a collection of elements, of which one will be chosen

XML Schema: The Email Example

```
<element name="email" type="emailType"/>


<complexType name="emailType">
    <sequence>
        <element name="head" type="headType"/>
        <element name="body" type="bodyType"/>
    </sequence>
</complexType>
```

XML Schema: The Email Example

```
<complexType name="headType">
    <sequence>
        <element name="from" type="nameAddress"/>
        <element name="to" type="nameAddress"
                minOccurs="1" maxOccurs="unbounded"/>
        <element name="cc" type="nameAddress"
                minOccurs="0" maxOccurs="unbounded"/>
        <element name="subject" type="string"/>
    </sequence>
</complexType>
```

XML Schema: The Email Example

```
<complexType name="nameAddress">
        <attribute name="name" type="string"
          use="optional"/>
        <attribute name="address"
         type="string" use="required"/>
</complexType>
```

- Similar for bodyType

Outline

Namespaces

- XML namespaces provide uniquely named elements & attributes in an XML document
- XML document may use >1 DTD or schema
- Since each was developed independently, name collisions can occur
- Solution: use different prefix for each DTD or schema

  prefix:name
- **Namespaces even more important in RDF**

An Example

```
<vu:instructors xmlns:vu="http://www.vu.com/empDTD"
                xmlns:gu="http://www.gu.au/empDTD"
                xmlns:uky="http://www.uky.edu/empDTD" >
    <uky:faculty uky:title="assistant professor"
                 uky:name="John Smith"
                 uky:department="Computer Science"/>
    <gu:academicStaff  gu:title="lecturer"
                       gu:name="Mate Jones"
                       gu:school="Information Technology"/>
</vu:instructors>
```

Namespace Declarations

- Namespaces declared within elements for use in it and its children (elements and attributes)
- A namespace declaration has form:
  - **xmlns:prefix="location"**
  - **location** is the URL of the DTD or XML schema
- If no prefix specified: **xmlns="location"** then the **location** is used as the *default* prefix
- We'll see this same idea used in RDF

Outline

(1) Introduction

(2) Description of XML

(3) Structuring

- DTDs
- XML Schema

(4) Namespaces

(5) Accessing, querying XML docs: XPath

(6) Transformations: XSLT

Addressing & Querying XML Documents

- In relational databases, parts of a database can be selected and retrieved using SQL
  - Also very useful for XML documents
  - **Query languages**: XQuery, XQL, XML-QL
- The central concept of XML query languages is a **path expression**
  - Specifies how a node or set of nodes, in the tree representation, can be reached
- Useful for extracting data from XML

XPath

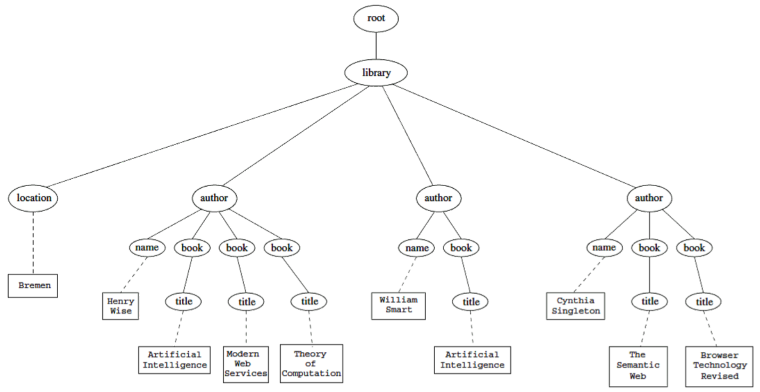
- XPath is core for XML query languages
- Language for addressing XML document parts
  - Operates on the tree data model of XML
  - Has a non-XML syntax
- Versions
  - XPath 1.0 (1999) is widely supported
  - XPath 2.0 (2007) more expressive subset of Xquery
  - XPath 3.1 (2017) current version, more features

Types of Path Expressions

- **Absolute** (starting at the root of the tree)
  - Syntactically they begin with the symbol **/**
  - It refers to the root of the document (one level above document’s root element)
- **Relative** to a context node

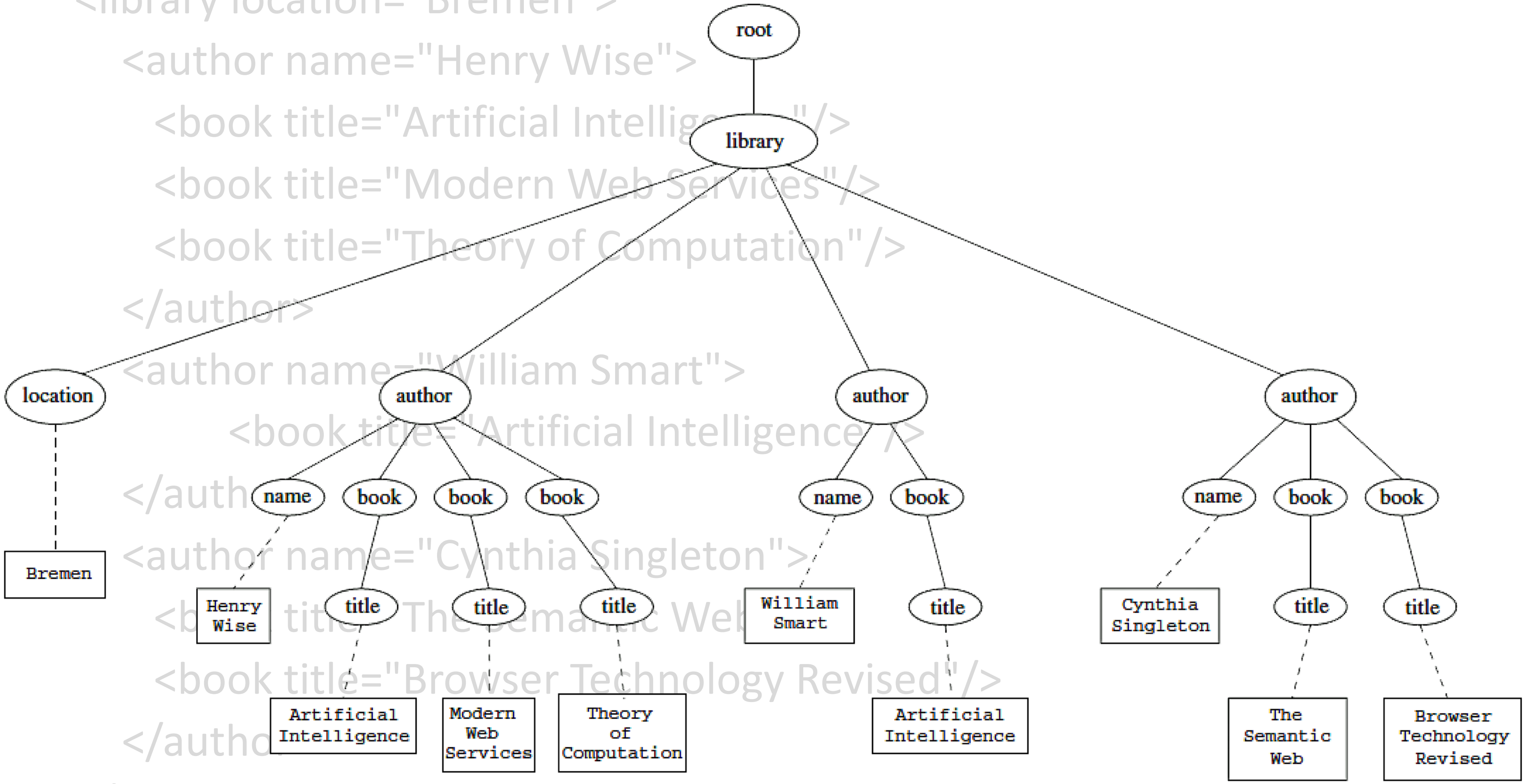
An XML Example

```
<library location="Bremen">
    <author name="Henry Wise">
      <book title="Artificial Intelligence"/>
      <book title="Modern Web Services"/>
      <book title="Theory of Computation"/>
    </author>
    <author name="William Smart">
            <book title="Artificial Intelligence"/>
    </author>
    <author name="Cynthia Singleton">
      <book title="The Semantic Web"/>
      <book title="Browser Technology Revised"/>
    </author>
</library>
```



Tree Representation

```
<library location="Bremen">
  <author name="Henry Wise">
    <book title="Artificial Intelligence"/>
    <book title="Modern Web Services"/>
    <book title="Theory of Computation"/>
  </author>
  <author name="William Smart">
      <book title="Artificial Intelligence"/>
  </author>
  <author name="Cynthia Singleton">
    <book title="The Semantic Web"/>
    <book title="Browser Technology Revised"/>
  </author>
</library>
```



Examples of Path Expressions in XPath

- **Q1: /library/author**
  - Addresses all **author** elements that are children of the **library** element node immediately below root
  - **/t1/.../tn**, where each **ti+1** is a child node of **ti**, is a path through the tree representation
- **Q2: //author**
  - Consider all elements in document and check whether they are of type **author**
  - Path expression addresses all **author** elements anywhere in the document

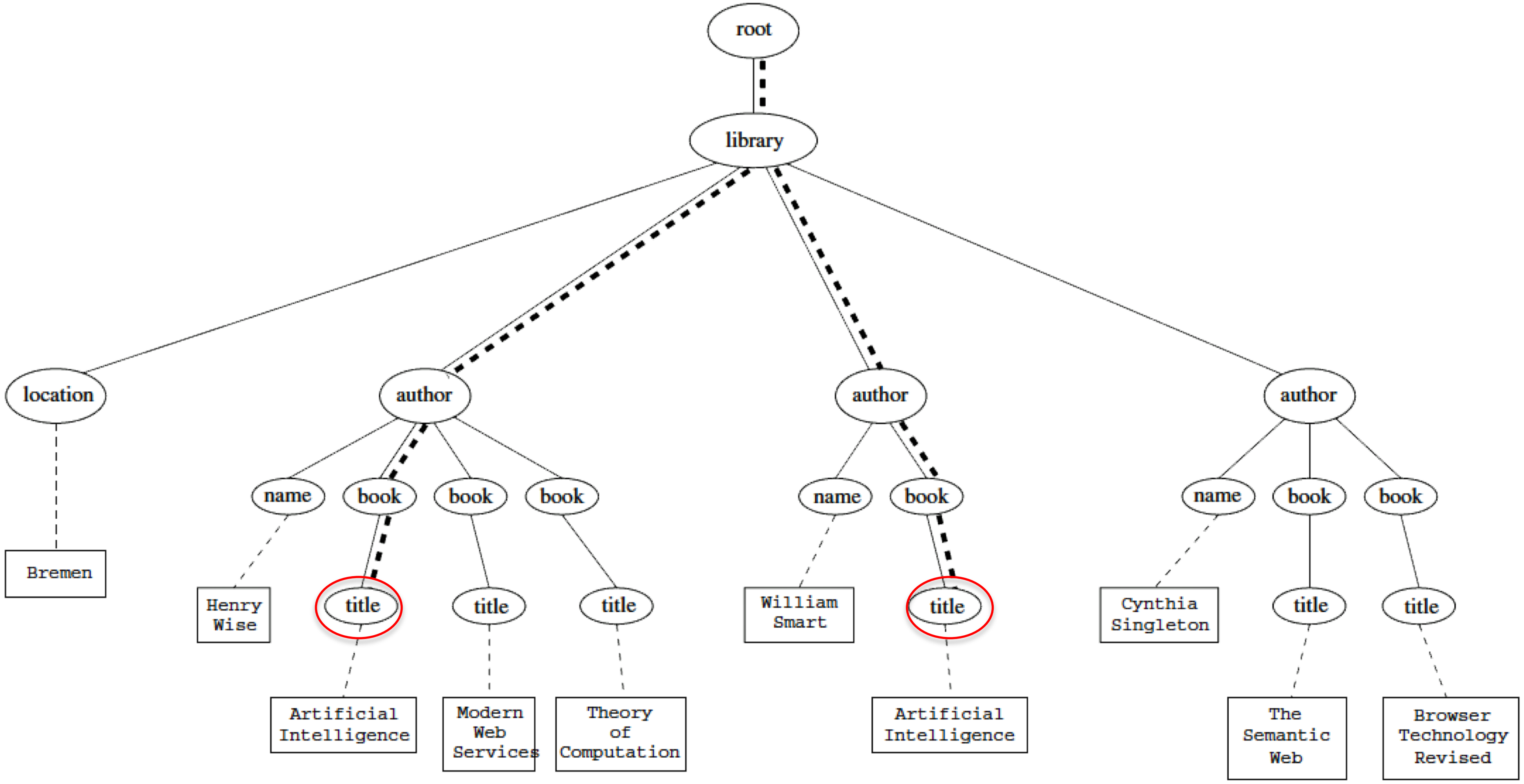
Examples of Path Expressions in XPath

- **Q3: /library/@location**
  - Addresses location attribute nodes within library element nodes
  - The symbol **@** is used to denote attribute nodes

- **Q4: //book/@title="Artificial Intelligence”**
  - Adresses all title attribute nodes within book elements anywhere in the document that have the value “Artificial Intelligence”

Tree Representation of Query 4

//book/@title="Artificial Intelligence”

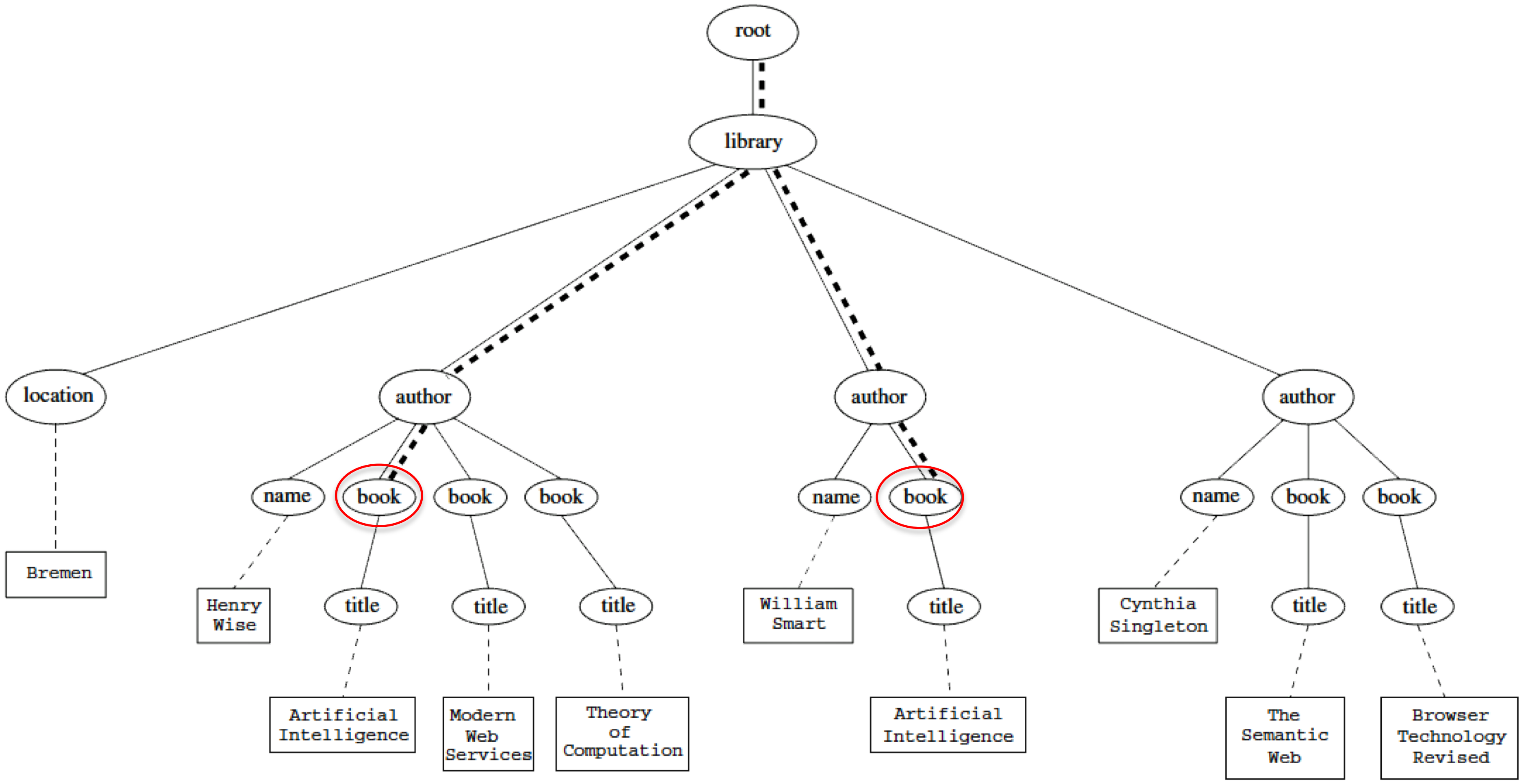


Examples of Path Expressions in XPath

- **Q5: /book[@title="Artificial Intelligence"]**
  - Addresses all books with title “Artificial Intelligence”
  - A test in brackets is a **filter expression** that restricts the set of addressed nodes.
  - Note differences between Q4 and Q5:
    - Query 5 addresses **book** elements, the **title** of which satisfies a certain condition.
    - Query 4 collects **title** attribute nodes of **book** elements

Tree Representation of Query 5

/book[@title="Artificial Intelligence"]



Examples of Path Expressions in XPath

- Q6: Address **first** author element node in the XML document

```
//author[1]
```

- Q7: Address **last** book element within the first author element node in the document

```
//author[1]/book[last()]
```

- Q8: Address all book element nodes **without a title** attribute

```
//book[not @title]
```

Outline

Displaying XML Documents

```
<author>
    <name>Grigoris Antoniou</name>
    <affiliation>University of Bremen</affiliation>
    <email>ga@tzi.de</email>
</author>
```

may be displayed in different ways:

Grigoris Antoniou	*Grigoris Antoniou*
University of Bremen	University of Bremen
ga@tzi.de	*ga@tzi.de*

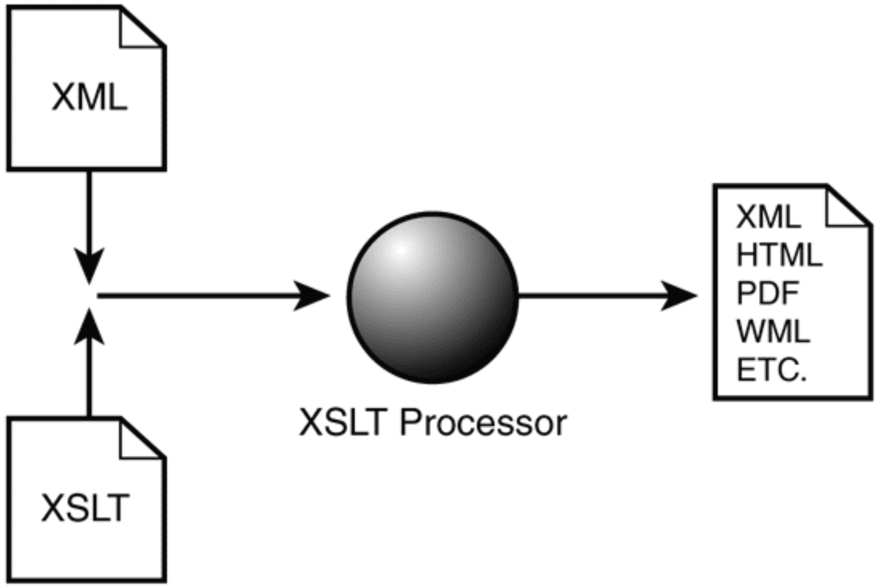
Idea: use an external *style sheet* to transform an XML tree into an HTML or XML tree

Style Sheets

- Style sheets can be written in various languages
  - E.g. CSS2 (cascading style sheets level 2)
  - XSL (extensible stylesheet language)
- XSL includes
  - a transformation language (XSLT)
  - a formatting language
  - Both are XML applications

XSL Transformations (XSLT)

- XSLT specifies rules to transform XML document to
  - another XML document
  - HTML document
  - plain text



- Output document may use same DTD/schema, or completely different vocabulary
- XSLT can be used independently of formatting language

XSLT Use Cases

- Move data & metadata from one XML representation to another
- Share information between appli-cations using different schemas
- Processing XML content for ingest into a program or database
- The following example show XSLT used to display XML documents as HTML

```
<?xml version="1.0"
<xsl:stylesheet xmln
<!-- created 2005-12-12-->
  <xsl:include href="xslt_
  <xsl:output method="xml"
  <xsl:template match="/">
  <root>
   Heuristic:<xsl:value-of
   <p>The leading manufact
  </root>
  </xsl:template>
 </xsl:stylesheet>
```

XSLT Transformation into HTML

```
<author>
  <name>Grigoris Antoniou</name>
  <affiliation>University of Bremen
     </affiliation>
  <email>ga@tzi.de</email>
</author>
```

```
<xsl:template match="/author">
  <html>
      <head><title>An author</title></head>
      <body bgcolor="white">
            <b><xsl:value-of select="name"/></b><br/>
            <xsl:value-of select="affiliation"/><br/>
            <i><xsl:value-of select="email"/></i>
      </body>
  </html>
</xsl:template>
```

Style Sheet Output

```
<author>
  <name>Grigoris Antoniou</name>
  <affiliation>University of Bremen</affiliation>
  <email>ga@tzi.de</email>
</author>
```

```
<xsl:template match="/author"> <html>
      <head><title>An author</title></head>
      <body bgcolor="white">
        <b><xsl:value-of select="name"/></b><br/>
        <xsl:value-of select="affiliation"/><br/>
        <i><xsl:value-of select="email"/></i>
      </body>
  </html></xsl:template>
```

```
<html>
   <head><title>An author</title></head>
   <body bgcolor="white">
         <b>Grigoris Antoniou</b><br/>
         University of Bremen<br/>
         <i>ga@tzi.de</i>
   </body>
</html>
```

Observations About XSLT

- XSLT documents are XML documents
  - XSLT sits on top of XML
- The XSLT document defines a **template**
  - In this case, an HTML document with placeholders for content to be inserted
- **xsl:value-of** retrieves value of an element and copies it into output document
  - It places some content into the template

Auxiliary Templates

- We may have an XML document with details of several authors
- It is a waste of effort to treat each **author** element separately
- In such cases, a special template is defined for **author** elements, which is used by the main template

Example of an Auxiliary Template

```
<authors>
  <author>
        <name>Grigoris Antoniou</name>
        <affiliation>University of Bremen</affiliation>
        <email>ga@tzi.de</email>
  </author>
  <author>
        <name>David Billington</name>
        <affiliation>Griffith University</affiliation>
        <email>david@gu.edu.net</email>
  </author>
</authors>
```

Example of an Auxiliary Template

```
<xsl:template match="/">
  <html>
    <head><title>Authors</title></head>
    <body bgcolor="white">
      <xsl:apply-templates select="author"/>
      <!-- apply templates for AUTHORS children -->
    </body>
  </html>
</xsl:template>
```

Example of an Auxiliary Template

```
<xsl:template match="authors">
      <xsl:apply-templates select="author"/>
</xsl:template>


<xsl:template match="author">
  <h2><xsl:value-of select="name"/></h2>
  <p> Affiliation:<xsl:value-of select="affiliation"/><br/>
  Email: <xsl:value-of select="email"/> </p>
</xsl:template>
```

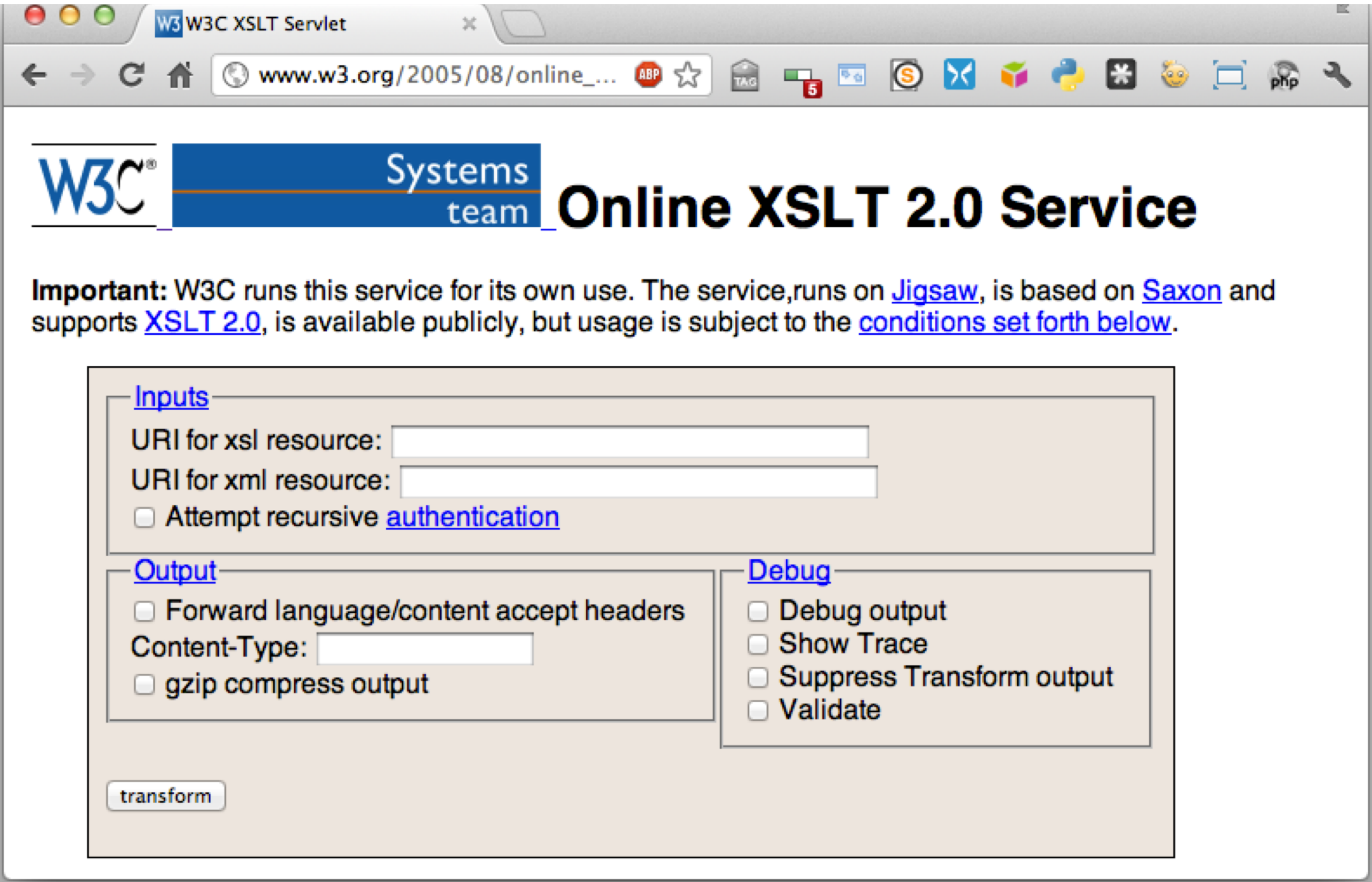
Multiple Authors Output

```
<html>
  <head><title>Authors</title></head>
  <body bgcolor="white">
      <h2>Grigoris Antoniou</h2>
      <p>Affiliation: University of Bremen<br/>
      Email: ga@tzi.de</p>
      <h2>David Billington</h2>
      <p>Affiliation: Griffith University<br/>
      Email: david@gu.edu.net</p>
  </body>
</html>
```

How to apply XSLT transforms

- When a modern browsers loads an XML file, it will will apply a linked XSLT and display the results (hopefully HTML!)
- Use an external Web service
- Use an XML editor
- Use a module or library for your favorite programming language

An XSLT Web Service



Online XSLT 2.0 Service

Important: W3C runs this service for its own use. The service,runs on Jigsaw, is based on Saxon and supports XSLT 2.0, is available publicly, but usage is subject to the conditions set forth below.

Inputs

URI for xsl resource:

URI for xml resource:

☐ Attempt recursive authentication

Output

☐ Forward language/content accept headers

Content-Type:

☐ gzip compress output

Debug

☐ Debug output

☐ Show Trace

☐ Suppress Transform output

☐ Validate

http://www.w3.org/2005/08/online_xslt/

CD Catalog example

```
<?xml-stylesheet type="text/xsl"
href="cdcatalog.xsl"?>
<catalog>
<cd>
  <title>Empire Burlesque</title>
  <artist>Bob Dylan</artist>
  <country>USA</country>
  <company>Columbia</company>
  <price>10.90</price>
  <year>1985</year>
</cd>
<cd>
  <title>Hide your heart</title>
  <artist>Bonnie Tyler</artist>
  <country>UK</country>
  <company>CBS Records</company>
…
</cd> …
```

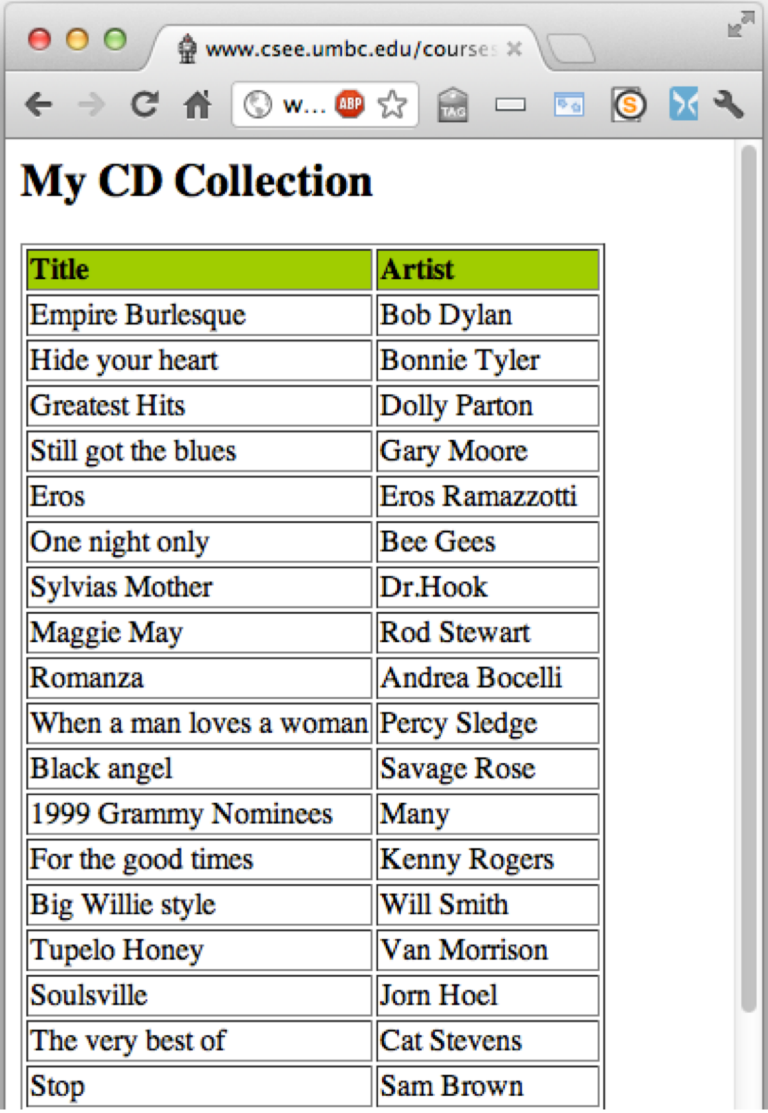
```
<xsl:template match="/">
 <html> <body>
 <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th align="left">Title</th>
      <th align="left">Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
     <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
     </tr>
    </xsl:for-each>
  </table>
 </body> </html>
</xsl:template>
</xsl:stylesheet>
```

See http://bit.ly/VQfLVV

Viewing an XML file in a Browser

- ~> **curl –L** https://www.csee.umbc.edu/courses/graduate/691/fall18/01/examples/xml/cdcatalog/cdcatalog.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="cdcatalog.xsl"?>
<catalog>
<cd>
 <title>Empire Burlesque</title>
 <artist>Bob Dylan</artist>
 <country>USA</country>
 <company>Columbia</company>
 <price>10.90</price>
 <year>1985</year>
</cd>
<cd>
 <title>Hide your heart</title>
 <artist>Bonnie Tyler</artist>
 <country>UK</country>
 <company>CBS Records</company>
 <price>9.90</price>
 <year>1988</year>
</cd>
...
```



My CD Collection

Title	Artist
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tyler
Greatest Hits	Dolly Parton
Still got the blues	Gary Moore
Eros	Eros Ramazzotti
One night only	Bee Gees
Sylvias Mother	Dr.Hook
Maggie May	Rod Stewart
Romanza	Andrea Bocelli
When a man loves a woman	Percy Sledge
Black angel	Savage Rose
1999 Grammy Nominees	Many
For the good times	Kenny Rogers
Big Willie style	Will Smith
Tupelo Honey	Van Morrison
Soulsville	Jorn Hoel
The very best of	Cat Stevens
Stop	Sam Brown

XML Summary

- XML is a metalanguage that allows users to define markup
- XML separates content and structure from formatting
- XML is (one of the) the de facto standard to represent and exchange structured information on the Web
- XML is supported by query languages

Comments for Discussion

- The nesting of tags has no standard meaning
- Semantics of XML documents is not accessible to machines and may or may not be for people
- Collaboration and exchange supported if there is underlying shared understanding of vocabulary
- XML is well-suited for close collaboration where domain or community-based vocabularies are used and less so for global communication
- Databases went from tree structures (60s) to relations (80s) and graphs (10s)