

Chapter 2

Structured Web Documents in XML



Adapted from slides from Grigoris Antoniou and Frank van Harmelen

Outline

- (1) Introduction
- (2) XML details
- (3) Structuring
 - DTDs
 - XML Schema
- (4) Namespaces
- (5) Accessing, querying XML documents: XPath
- (6) Transformations: XSLT

History

- XML's roots are in SGML
 - Standard Generalized Markup Language
 - A metalanguage for defining document markup languages
 - Very extensible, but very complicated
- HTML was defined using SGML
 - It's a markup language, not a markup metalanguage
- XML proposal to W3C in July 1996
 - Idea: a simplified SGML could greatly expand the power and flexibility of the Web
 - First XML Meeting, August 1996, Seattle
- Evolving series of W3C recommendations

(1) Introduction

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
```

(1) Introduction

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>
```

(1) Introduction

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 quite easily
- ... **But how about machines?**

(1) Introduction

Problems Interpreting HTML Documents

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
- Are there two authors?
- Or just one, called "V. Marek and M. Truszczynski"?

(1) Introduction

HTML vs XML: Structural Information

- HTML documents do not contain **structural information**: pieces of the document and their relationships.
- XML more easily accessible to machines because
 - Every piece of information is described.
 - Relations are also defined through the nesting structure.
 - E.g., the **<author>** tags appear within the **<book>** tags, so they describe properties of the particular book.

(1) Introduction

HTML vs XML: Structural Information

- A machine processing the XML document would be able to deduce that
 - the **author** element refers to the enclosing **book** element
 - rather than by proximity considerations
- XML allows the definition of constraints on values
 - E.g. a year must be a number of four digits

(1) Introduction

HTML vs. XML: Formatting

- The HTML representation provides more than the XML representation:
 - The formatting of the document is also described
- The 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)

(1) Introduction

HTML vs. XML: Another Example

In HTML

```
<h2>Relationship matter-energy</h2>  
<i> E = M × c2 </i>
```

In XML

```
<equation>  
  <gloss>Relationship matter energy </gloss>  
  <leftside> E </leftside>  
  <rightside> M × c2 </rightside>  
</equation>
```

(1) Introduction

HTML vs. XML: Different Use of Tags

- Both HTML documents use the same tags
- The XML documents use completely different tags
- HTML tags define display: color, lists ...
- XML tags not fixed: **user definable tags**
- XML **meta markup language**: language for defining markup languages

(1) Introduction

XML Vocabularies

- Web applications must agree on common vocabularies to communicate and collaborate
- Communities and business sectors are defining their specialized vocabularies
 - mathematics (MathML)
 - bioinformatics (BSML)
 - human resources (HRML)
 - Syndication (RSS)
 - Vector graphics (SVG)
 - ...

(1) Introduction

Outline

- (1) Introduction
- (2) Detailed Description of XML**
- (3) Structuring
 - DTDs
 - XML Schema
- (4) Namespaces
- (5) Accessing, querying XML documents: XPath
- (6) Transformations: XSLT

(2) XML details

The XML Language

An XML document consists of

- a **prolog**
- a number of **elements**
- an optional **epilog** (not discussed, not used much)

(2) XML details

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">
```

(2) XML details

XML Elements

- 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>
```

(2) XML details

XML Elements

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

(2) XML details

Content of XML Elements

- Content may 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 the element is called empty; it is abbreviated as follows:

```
<lecturer/> for <lecturer></lecturer>
```

(2) XML details

XML Attributes

- An empty element is not necessarily meaningless
 - It may have some properties in terms of attributes
- An attribute is a name-value pair inside the opening tag of an element

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

(2) XML details

XML Attributes: An Example

```
<order orderNo="23456"
      customer="John Smith"
      date="October 15, 2002" >
  <item itemNo="a528" quantity="1" />
  <item itemNo="c817" quantity="3" />
</order>
```

(2) XML details

The Same Example without Attributes

```
<order>
  <orderNo>23456</orderNo>
  <customer>John Smith</customer>
  <date>October 15, 2002</date>
  <item>
    <itemNo>a528</itemNo>
    <quantity>1</quantity>
  </item>
  <item>
    <itemNo>c817</itemNo>
    <quantity>3</quantity>
  </item>
</order>
```

(2) XML details

XML Elements vs. Attributes

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

(2) XML details

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"?>
```

(2) XML details

Well-Formed XML Documents

Syntactically correct documents must adhere to many rules

- Only one outermost element (the **root element**)
- Each element contains an opening and a corresponding closing tag
- Tags may not overlap
- `<author><name>Lee Hong</author></name>`
- Attributes within an element have unique names
- Element and tag names must be permissible

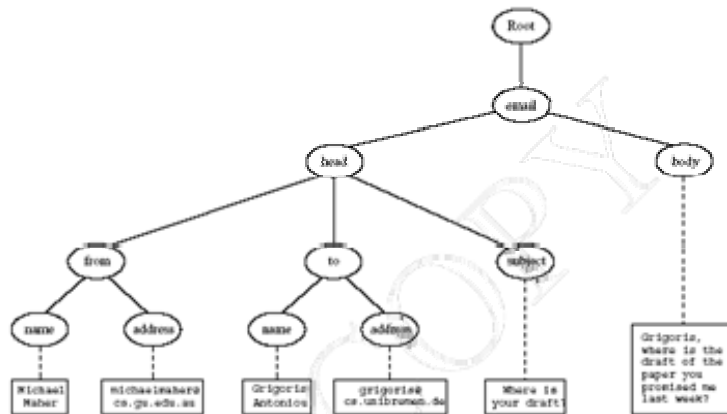
(2) XML details

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>
```

(2) XML details

Tree Model of XML Documents



(2) XML details

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 important

(2) XML details

Outline

- (1) Introduction
- (2) Detailed Description of XML
- (3) **Structuring**
 - **DTDs**
 - XML Schema
- (4) Namespaces
- (5) Accessing, querying XML documents: XPath
- (6) Transformations: XSLT

Structuring XML Documents

- Define all the 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**

(3) Structure

Structuring XML Documents

- An XML document is **valid** if
 - it is well-formed
 - respects the structuring information it uses
- There are several ways of defining the structure of XML documents:
 - **DTDs** (Document Type Definition) came first, was based on SGML's approach.
 - **XML Schema** (aka XML Schema Definition or XSD) is a more recent W3C recommendation and offers extended possibilities
 - RELAX NG and DSDs are two alternatives

(3) Structure

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) >
```

(3) Structure: DTDs

The Meaning of the DTD

- 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 the only atomic type for elements
 - PCDATA = “parsed character data”

(3) Structure: DTDs

Disjunction in Element Type Definitions

- We express that a **lecturer** element contains *either* a **name** element *or* a **phone** element as follows:
<!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?

(3) Structure: DTDs

Example of an XML Element

```
<order orderNo="23456"
      customer="John Smith"
      date="October 15, 2002">
  <item itemNo="a528" quantity="1"/>
  <item itemNo="c817" quantity="3"/>
</order>
```

(3) Structure: DTDs

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 >
```

(3) Structure: DTDs

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**:
 - Specifies how many item elements can be in an order
 - **?**: appears zero times or once
 - *****: appears zero or more times
 - **+**: appears one or more times
 - No cardinality operator means exactly once

(3) Structure: DTDs

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 triplets of attribute name, attribute type, and value type
- *Attribute name*: A name that may be used in an XML document using a DTD

(3) Structure: DTDs

DTD: Attribute Types

- Similar to predefined data types, but limited selection
- 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**, a reference to another element with an ID attribute carrying the same value as the IDREF attribute (~ DB foreign key)
 - **IDREFS**, a series of IDREFs
 - **(v1) . . . (vn)**, an enumeration of all possible values
- Limitations: no dates, number ranges etc.

(3) Structure: DTDs

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

(3) Structure: DTDs

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 >
```

(3) Structure: DTDs

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>
```

(3) Structure: DTDs

A DTD for an Email Element

```
<!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>
```

(3) Structure: DTDs

A DTD for an Email Element

```
<!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>
```

(3) Structure: DTDs

Interesting Parts of the DTD

- A **head** element contains (in that order):
 - a **from** element
 - at least one **to** element
 - zero or more **cc** elements
 - a **subject** element
- In **from**, **to**, and **cc** elements
 - the **name** attribute is not required
 - the **address** attribute is always required

(3) Structure: DTDs

Interesting Parts of the DTD

- A **body** element contains
 - a **text** element
 - possibly followed by a number of **attachment** elements
- The **encoding** attribute of an **attachment** element must have either the value “**mime**” or “**binhex**”
 - “**mime**” is the default value

(3) Structure: DTDs

Remarks on DTDs

- A DTD can be interpreted as an Extended Backus-Naur Form (EBNF)
 - **<!ELEMENT email (head,body)>**
 - is equivalent to **email ::= head body**
- Recursive definitions possible in DTDs
 - **<!ELEMENT bintree**
((bintree root bintree)|emptytree)>

(3) Structure: DTDs

Outline

- (1) Introduction
- (2) Detailed Description of XML
- (3) Structuring**
 - DTDs
 - **XML Schema**
- (4) Namespaces
- (5) Accessing, querying XML documents: XPath
- (6) Transformations: XSLT

XML Schema

- Significantly richer language for defining the structure of XML documents
- Syntax is based on XML itself
 - separate tools to handle 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)
- W3C published the XML Schema recommendation in 2001

(3) Structure: XML Schema

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

(3) Structure: XML Schema

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

(3) Structure: XML Schema

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**

(3) Structure: XML Schema

Data Types

- There are 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.
- There are also **user-defined data types**
 - **simple data types**, which cannot use elements or attributes
 - **complex data types**, which can use these

(3) Structure: XML Schema

Complex Data Types

Complex data types are defined from already 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

(3) Structure: XML Schema

A Data Type Example

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

(3) Structure: XML Schema

Data Type Extension

Already existing data types can be extended by new elements or attributes. Example:

```
<complexType name="extendedLecturerType">
  <extension base="lecturerType">
    <sequence>
      <element name="email" type="string"
        minOccurs="0" maxOccurs="1"/>
    </sequence>
    <attribute name="rank" type="string"
      use="required"/>
  </extension>
</complexType>
```

(3) Structure: XML Schema

Resulting Data Type

```
<complexType name="extendedLecturerType">
  <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="title" type="string" use="optional"/>
  <attribute name="rank" type="string" use="required"/>
</complexType>
```

(3) Structure: XML Schema

Data Type Extension

A **hierarchical relationship** exists between the original and the extended type

- Instances of the extended type are also instances of the original type
- They may contain additional information, but neither less information, nor information of the wrong type

(3) Structure: XML Schema

Data Type Restriction

- An existing data type may be restricted by adding constraints on certain values
- Restriction is not the opposite from extension
 - Restriction is not achieved by deleting elements or attributes
- The following **hierarchical relationship** still holds:
 - Instances of the restricted type are also instances of the original type
 - They satisfy at least the constraints of the original type

(3) Structure: XML Schema

Example of Data Type Restriction

```
<complexType name="restrictedLecturerType">
  <restriction base="lecturerType">
    <sequence>
      <element name="firstname" type="string"
        minOccurs="1" maxOccurs="2"/>
    </sequence>
    <attribute name="title" type="string"
      use="required"/>
  </restriction>
</complexType>
```

(3) Structure: XML Schema

Restriction of Simple Data Types

```
<simpleType name="dayOfMonth">
  <restriction base="integer">
    <minInclusive value="1"/>
    <maxInclusive value="31"/>
  </restriction>
</simpleType>
```

(3) Structure: XML Schema

Data Type Restriction: Enumeration

```
<simpleType name="dayOfWeek">
  <restriction base="string">
    <enumeration value="Mon"/>
    <enumeration value="Tue"/>
    <enumeration value="Wed"/>
    <enumeration value="Thu"/>
    <enumeration value="Fri"/>
    <enumeration value="Sat"/>
    <enumeration value="Sun"/>
  </restriction>
</simpleType>
```

(3) Structure: XML Schema

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>
```

(3) Structure: XML Schema

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>
```

(3) Structure: XML Schema

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

(3) Structure: XML Schema

Outline

- (1) Introduction
- (2) Detailed Description of XML
- (3) Structuring
 - DTDs
 - XML Schema
- (4) Namespaces**
- (5) Accessing, querying XML documents: XPath
- (6) Transformations: XSLT

Namespaces

- An XML document may use more than one DTD or schema
- Since each structuring document was developed independently, name clashes may appear
- The solution is to use a different prefix for each DTD or schema
 - **prefix:name**

(4) Namespaces

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>
```

(4) Namespaces

Namespace Declarations

- Namespaces are declared within an element and can be used in that element and any of its children (elements and attributes)
- A namespace declaration has the form:
 - **xmlns:prefix="location"**
 - **location** is the address of the DTD or schema
- If a prefix is not specified: **xmlns="location"** then the **location** is used by default

(4) Namespaces

Outline

- (1) Introduction
- (2) Detailed Description of XML
- (3) Structuring
 - DTDs
 - XML Schema
- (4) Namespaces
- (5) Accessing, querying XML documents: 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 a set of nodes, in the tree representation of the XML document can be reached

(5) XPath

XPath

- XPath is core for XML query languages
- Language for addressing parts of an XML document.
 - It operates on the tree data model of XML
 - It has a non-XML syntax

(5) XPath

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 (situated one level above the root element of the document)
- **Relative** to a context node

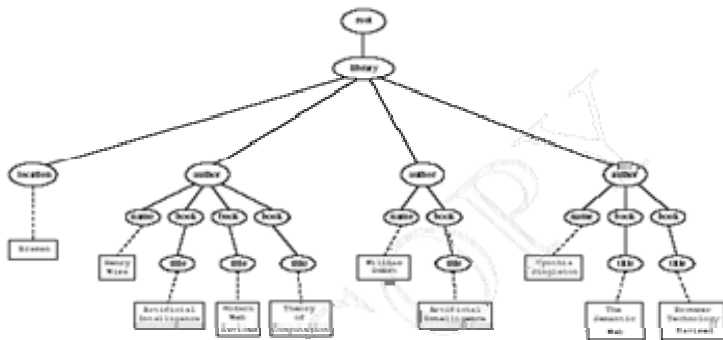
(5) XPath

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>
```

(5) XPath

Tree Representation



(5) XPath

Examples of Path Expressions in XPath

- **Q1: Address all author elements**
/library/author
- Addresses all **author** elements that are children of the **library** element node, which resides immediately below the root
- **/t1.../tn**, where each **ti+1** is a child node of **ti**, is a path through the tree representation

(5) XPath

Examples of Path Expressions in XPath

- Q2: Address all author elements

//author

- Here // says that we should consider all elements in the document and check whether they are of type **author**
- This path expression addresses all **author** elements anywhere in the document

(5) XPath

Examples of Path Expressions in XPath

- Q3: Address the location attribute nodes within library element nodes

//library/@location

Note: The symbol @ is used to denote attribute nodes

- Q4: Address all title attribute nodes within book elements anywhere in the document, which have the value "Artificial Intelligence"

//book/@title="Artificial Intelligence"

(5) XPath

Examples of Path Expressions in XPath

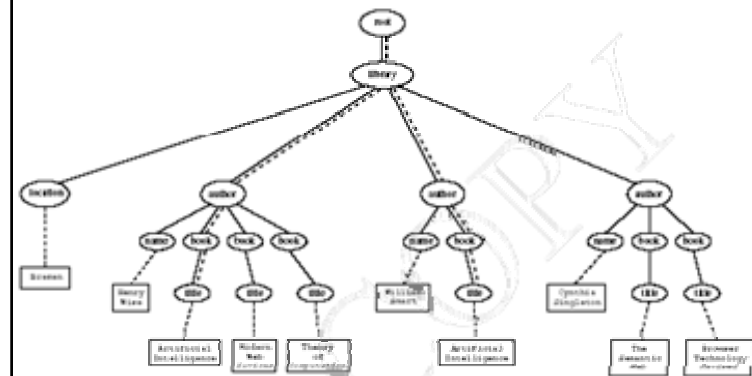
- Q5: Address all books with title "Artificial Intelligence"

/book[@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

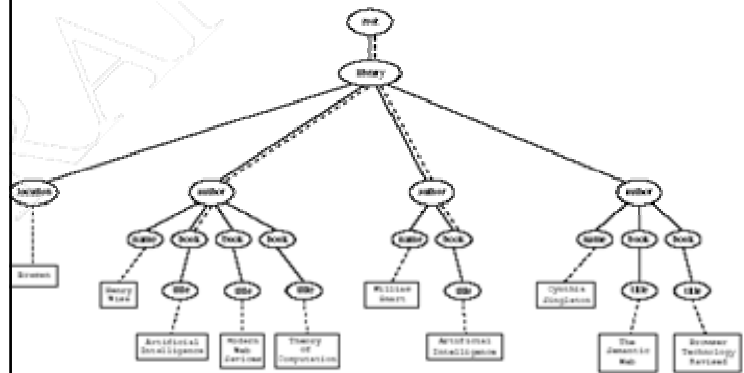
(5) XPath

Tree Representation of Query 4



(5) XPath

Tree Representation of Query 5



(5) XPath

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]`

(5) XPath

General Form of Path Expressions

- A **path expression** consists of a series of steps, separated by slashes
- A **step** consists of
 - An **axis specifier**,
 - A **node test**, and
 - An optional **predicate**

(5) XPath

General Form of Path Expressions

- An **axis specifier** determines the tree relationship between the nodes to be addressed and the context node
 - E.g. parent, ancestor, child (the default), sibling, attribute node
 - // is such an axis specifier: descendant or self

(5) XPath

General Form of Path Expressions

- A **node test** specifies which nodes to address
 - The most common node tests are element names
 - E.g., * addresses all element nodes
 - **comment()** addresses all comment nodes

(5) XPath

General Form of Path Expressions

- **Predicates** (or *filter expressions*) are optional and are used to refine the set of addressed nodes
 - E.g., the expression **[1]** selects the first node
 - **[position()=last()]** selects the last node
 - **[position() mod 2 =0]** selects the even nodes
- XPath has a more complicated full syntax.
 - We have only presented the abbreviated syntax

(5) XPath

Outline

- (1) Introduction
- (2) Detailed Description of XML
- (3) Structuring
 - DTDs
 - XML Schema
- (4) Namespaces
- (5) Accessing, querying XML documents: XPath
- (6) Transformations: XSLT**

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	<i>Grigoris Antoniou</i>
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

(5) XSLT transformations

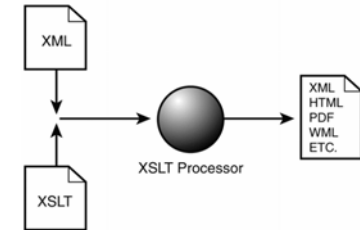
Style Sheets

- 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

(5) XSLT transformations

XSL Transformations (XSLT)

- XSLT specifies rules with which an input XML document is transformed to
 - another XML document
 - an HTML document
 - plain text
- The output document may use the same DTD or schema, or a completely different vocabulary
- XSLT can be used independently of the formatting language



(5) XSLT transformations

XSLT

- Move data and metadata from one XML representation to another
- XSLT is chosen when applications that use different DTDs or schemas need to communicate
- XSLT can be used for machine processing of content without any regard to displaying the information for people to read.
- In the following example we use XSLT only to display XML documents as HTML

(5) XSLT transformations

XSLT Transformation into HTML

```
<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>
```

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

(5) XSLT transformations

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>
```

(5) XSLT transformations

Observations About XSLT

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

(5) XSLT transformations

A Template

```
<html>
  <head><title>An author</title></head>
  <body bgcolor="white">
    <b>...</b><br>
    ...<br>
    <i>...</i>
  </body>
</html>
```

(5) XSLT transformations

Auxiliary Templates

- We 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

(5) XSLT transformations

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>
```

(5) XSLT transformations

Example of an Auxiliary Template (2)

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

(5) XSLT transformations

Example of an Auxiliary Template (3)

```
<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>
```

(5) XSLT transformations

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>
```

(5) XSLT transformations

Explanation of the Example

xsl:apply-templates element causes all children of the context node to be matched against the selected path expression

- e.g., if the current template applies to /, then element **xsl:apply-templates** applies to root element
- i.e. the **authors** element (/ is located above the root element)
- If current context node is the **authors** element, then element **xsl:apply-templates select="author"** causes the template for the **author** elements to be applied to all **author** children of the **authors** element

(5) XSLT transformations

Explanation of the Example

- It is good practice to define a template for each element type in the document
 - Even if no specific processing is applied to certain elements, the **xsl:apply-templates** element should be used
 - E.g. **authors**
- In this way, we work from the root to the leaves of the tree, and **all** templates are applied

(5) XSLT transformations

Processing XML Attributes

Suppose we wish to transform to itself the element:

```
<person firstname="John" lastname="Woo"/>
```

Wrong solution:

```
<xsl:template match="person">
  <person firstname="<xsl:value-of select="@firstname">"
    lastname="<xsl:value-of select="@lastname">"/>
</xsl:template>
```

(5) XSLT transformations

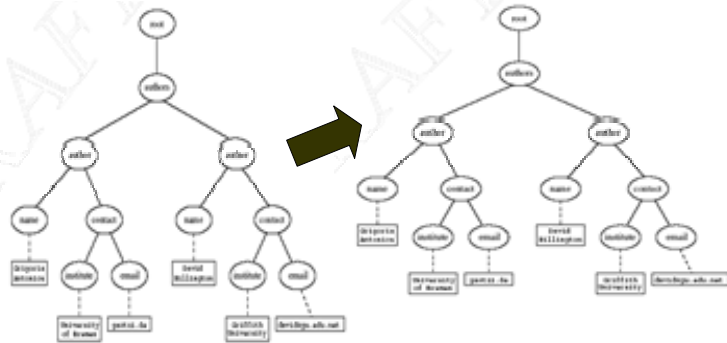
Processing XML Attributes

- Not well-formed because tags are not allowed within the values of attributes
- We wish to add attribute values into template

```
<xsl:template match="person">
  <person
    firstname="{@firstname}"
    lastname="{@lastname}" />
</xsl:template>
```

(5) XSLT transformations

Transforming an XML Document to Another



(5) XSLT transformations

Transforming an XML Document to Another

```
<xsl:template match="/">
  <?xml version="1.0" encoding="UTF-16"?>
  <authors>
    <xsl:apply-templates select="authors"/>
  </authors>
</xsl:template>

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

(5) XSLT transformations

Transforming an XML Document to Another

```
<xsl:template match="author">
  <name><xsl:value-of select="name"/></name>
  <contact>
    <institution>
      <xsl:value-of select="affiliation"/>
    </institution>
    <email><xsl:value-of select="email"/></email>
  </contact>
</xsl:template>
```

(5) XSLT transformations

Summary

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

For Discussion in Subsequent Chapters

- The nesting of tags does not have standard meaning
- The semantics of XML documents is not accessible to machines, only to people
- Collaboration and exchange are supported if there is underlying shared understanding of the vocabulary
- XML is well-suited for close collaboration, where domain- or community-based vocabularies are used
 - It is not so well-suited for global communication.