Structured Web Documents in XML

CSE 595 – Semantic Web
Stony Brook University

http://www.cs.stonybrook.edu/~cse595
Lecture Outline

- Introduction
- Detailed Description of XML
- Structuring
  - DTDs
  - XML Schema
- Namespaces
- Accessing, querying XML documents: XPath
- Transformations: XSLT
An HTML Example

<h2>Nonmonotonic Reasoning: Context-Dependent Reasoning</h2>

<i>by V. Marek</i> and <i>M. Truszczynski</i>

Springer 1993

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>
</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 quite easily
- … But how about machines?
Problems with Automated Interpretation of 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”?
HTML vs XML: Structural Information

• HTML documents do not contain structural information, i.e., pieces of the document and their relationships.
  • HTML has only presentation
• 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.
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. `year` must be a number of four digits
HTML vs XML: Formatting

- The HTML representation provides more presentation than the XML representation:
  - The formatting of the document is also described
- The main use of an HTML document is to display information, therefore, it must define formatting
- XML: separation of content from display
  - same information can be displayed in different ways
HTML vs XML: Another Example

• In HTML
  
  `<h2>Relationship force-mass</h2>
  `<i> F = M × a </i>`

• In XML
  
  `<equation>
    `<meaning>Relationship force-mass</meaning>
    `<leftside> F </leftside>
    `<rightside> M × a </rightside>
  </equation>"
HTML vs XML: Different Use of Tags

- In both previous HTML docs we have the same tags
- In XML completely different (for different meanings)
- HTML tags define display: color, lists ...
- XML tags not fixed: user definable tags
- XML is a meta markup language: language for defining markup languages
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)
  - ...

@ Semantic Web Primer
Lecture Outline

- Introduction
- Detailed Description of XML
- Structuring
  - DTDs
  - XML Schema
- Namespaces
- Accessing, querying XML documents: XPath
- Transformations: XSLT
The XML Language

• An XML document consists of
  • a prolog
  • a number of elements
  • an optional epilog
Prolog of an XML Document

- The prolog consists of:
  - an XML declaration
    ```xml
    <?xml version="1.0" encoding="UTF-16"?>
    ```
  - an optional reference to external structuring documents
    ```xml
    <!DOCTYPE book SYSTEM "book.dtd">
    ```
The XML declaration

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

- It specifies that the current document is an XML document, and defines the version and the character encoding used in the particular system (such as UTF-8, UTF-16, and ISO 8859-1).
- The character encoding is not mandatory, but its specification is considered good practice.
- Sometimes we also specify whether the document is self-contained – that is, whether it does not refer to external structuring documents:

```xml
<?xml version="1.0" encoding="UTF-16" standalone="no"?>
```
The optional reference to external structuring documents

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

- Here the structuring information is found in a local file called `book.dtd`
- Instead, the reference might be a URL.
- If only a locally recognized name or only a URL is used, then the label `SYSTEM` is used.
- If, however, one wishes to give both a local name and a URL, then the label `PUBLIC` should be used instead.
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>Paul Fodor</lecturer>
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”
Content of XML Elements

- Content may be text, or other elements, or nothing
  
  `<lecturer>
    <name>Paul Fodor</name>
    <phone> +1 (123)456−7890 </phone>
  </lecturer>`

- If there is no content, then the element is called empty; it is abbreviated as follows:
  
  `<lecturer/>`
  
  for
  
  `<lecturer></lecturer>`
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

```xml
<lecturer name="Paul Fodor" phone="+1 (123) 456-7890"/>
```
XML Attributes: An Example

```xml
<order orderNo="23456"
       customer="John Smith"
       date="January 1, 2020">
  <item itemNo="a528" quantity="1"/>
  <item itemNo="c817" quantity="3"/>
</order>
```
The Same Example without Attributes

```xml
<order>
  <orderNo>23456</orderNo>
  <customer>John Smith</customer>
  <date>January 1, 2020</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 matter of taste
• But attributes cannot be nested
Further Components of XML Docs

• Comments
  • A piece of text that is to be ignored by parser
  
  ```xml
  <!-- This is a comment -->
  ```

• Processing Instructions (PIs)
  • provide a mechanism for passing information to an application about how to handle elements.
  • The general form is: `<?target instruction?>`
  • Define procedural attachments
  
  ```xml
  <?stylesheet type="text/css" href="mystyle.css"?>
  ```

• PIs offer procedural possibilities in an otherwise declarative environment.
Well-Formed XML Documents

• An XML document is well-formed if it is syntactically correct.

• Some syntactic rules:
  • Only one outermost element (called root element)
  • Each element contains an opening and a corresponding closing tag
  • Tags may not overlap
    \[
    \text{<author><name>Lee Hong</name>></author>}
    \]
  • Attributes within an element have unique names
  • Element and tag names must be permissible
The Tree Model of XML Documents: An Example

- An XML document is well-formed if it is syntactically correct.

```xml
<?xml version="1.0" encoding="UTF-16"?>
<!DOCTYPE email SYSTEM "email.dtd">
<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>
```
The tree representation of this XML document is an ordered, labeled tree:
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
The Tree Model of XML Docs

• The order of attributes is not important:
  • the following two elements are equivalent:
    <person lastname="Woo" firstname="Jason"/>
    <person firstname="Jason" lastname="Woo"/>
  • This aspect is not represented properly in the tree.
    • In general, we would require a more refined tree concept; for example, we should also differentiate between the different types of nodes (element node, attribute node, etc.).
The Tree Model of XML Docs

• The figure also shows the difference between the root (representing the XML document), and the root element, in our case the `email` element.
• This distinction will play a role in addressing and querying XML documents.
Lecture Outline

- Introduction
- Detailed Description of XML
- Structuring
  - DTDs
  - XML Schema
- Namespaces
- Accessing, querying XML documents: XPath
- Transformations: XSLT
Structuring XML Documents

- An XML document is well-formed if it respects certain syntactic rules.
- However, those rules say nothing specific about the structure of the document.
- Imagine two applications that try to communicate, and that wish to use the same vocabulary.
- For this purpose it is necessary to define all the element and attribute names that may be used.
- The structure should also be defined: what values an attribute may take, which elements may or must occur within other elements, and so on.
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
  - We say that an XML document is **valid** if it is well-formed, uses structuring information, and respects that structuring information.
Structuring XML Documents

- An XML document is valid if
  - it is well-formed
  - respects the structuring information it uses

- There are two ways of defining the structure of XML documents:
  - DTDs (the older and more restricted way)
  - XML Schema (offers extended possibilities)
External and Internal DTDs

- Document Type Definition (DTD) is a set of markup declarations that define a document type.
- The components of a DTD can be defined in a separate file (*external DTD*) or within the XML document itself (*internal DTD*).
- Usually it is better to use external DTDs, because their definitions can be used across several documents; otherwise duplication is inevitable, and the maintenance of consistency over time becomes difficult.
DTD: Element Type Definition

```xml
<lecturer>
  <name>Paul Fodor</name>
  <phone> +1 (123)456-7890 </phone>
</lecturer>

- DTD for above element (and all lecturer elements):
  ```xml
  <!ELEMENT lecturer (name, phone)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT phone (#PCDATA)>
  ```
```
The Meaning of the DTD

- The meaning of this DTD is as follows:
  - 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.
**DTD: Disjunction in Element Type Definitions**

- We express that a lecturer element contains either a name element or a phone element as follows:

  ```xml
  <!ELEMENT lecturer (name|phone)>  
  ```

- It gets more difficult when we wish to specify that a lecturer element contains a name element and a phone element in any order. We can only use the trick:

  ```xml
  <!ELEMENT lecturer
    ((name,phone)|(phone,name))>  
  ```

- However, this approach suffers from practical limitations (imagine ten elements in any order).
Example of an XML Element

- Attributes: Consider the element:

```xml
<order orderNo="23456"
       customer="John Smith"
       date="January 1, 2020">
  <item itemNo="a528" quantity="1"/>
  <item itemNo="c817" quantity="3"/>
</order>
```
The Corresponding DTD

• A DTD for it looks like this:

```xml
<!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

• Compared to the previous example, a new aspect is that the item element type is defined to be **EMPTY**.

• Another new aspect is the appearance of + after item in the definition of the order element type.

• It is one of the cardinality operators:
  • ?: appears zero times or once
  • *: appears zero or more times
  • +: appears one or more times
  • No cardinality operator means exactly once
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: is a name that may be used in an XML document using a DTD
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
  - **IDREF**, a reference to another element with an **ID** attribute carrying the same value as the **IDREF** attribute
  - **IDREFS**, a series of **IDREF**s
  - \((v_1 \mid \ldots \mid v_n)\), an enumeration of all possible values
- Limitations: no dates, number ranges etc.
  - For example, dates have to be interpreted as strings (**CDATA**); thus their specific structure cannot be enforced.
There are four value types:

- **#REQUIRED**
  - Attribute must appear in every occurrence of the element type in the XML document
  - In the previous example, itemNo and quantity must always appear within an item element.

- **#IMPLIED**
  - The appearance of the attribute is optional
  - In the example, comments are optional.

- **#FIXED "value"**
  - Every element must have this attribute, which always has the value given after #FIXED in the DTD.
  - A value given in an XML document is meaningless because it is overridden by the fixed value.

- **"value"**
  - This specifies the default value for the attribute
  - If a specific value appears in the XML document, it overrides the default value.
Referencing with IDREF and IDREFS

```xml
<!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

```xml
<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>
```
XML Entities

• An XML entity can play the role of
  • a placeholder for repeatable characters
  • a section of external data
  • a part of a declaration for elements
• We can use the entity reference &thisyear instead of the value "2018"

```xml
<!ENTITY thisyear "2018">
```

• At each place the current year needs to be included, we can use the entity reference &thisyear; instead.
• This way, updating the year value to "2019" for the whole document will only mean changing the entity declaration.
A DTD for an Email Element

```xml
<!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>
```
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>
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
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
Remarks on DTDs

- A DTD can be interpreted as an Extended Backus-Naur Form (EBNF)
  ```xml
  <!ELEMENT email (head, body)>  
  ```
  is equivalent to
  ```xml
  email -> head body
  ```

- Recursive definitions possible in DTDs
  ```xml
  <!ELEMENT bintree
   ( (bintree root bintree) | emptytree)> 
  ```
  A binary tree is the empty tree, or consists of a left subtree, a root, and a right subtree.
Lecture Outline

- Introduction
- Detailed Description of XML
- Structuring
  - DTDs
  - XML Schema
- Namespaces
- Accessing, querying XML documents: XPath
- Transformations: XSLT
XML Schema

- Significantly richer language for defining the structure of XML documents
- Its syntax is based on XML itself
  - not necessary to write separate tools
- Reuse and refinement of schemas
  - Expand or delete already existent schemas
- Sophisticated set of data types, compared to DTDs (which only supports strings)
An XML schema is an element with an opening tag like

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

• The element uses the schema of XML Schema found at the W3C website
• It is the foundation on which new schemas can be built
• The prefix `xsd` denotes the namespace of that schema
• If the prefix is omitted in the `xmlns` attribute, then we are using elements from this namespace by default

```xml
<schema
    xmlns="http://www.w3.org/2000/10/XMLSchema"
    version="1.0">
```
XML Schema

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

• The syntax of element types is
  ```xml
  <element name="..."/>
  ```
  and they may have a number of optional attributes, such as types
  ```xml
  type="..."
  ```
or cardinality constraints

• `minOccurs="x"` (default value 1)
• `maxOccurs="x"` (default value 1)
  • Generalizations of *, ?, + offered by DTDs
Element Types

- Examples:

```xml
<element name="email"/>
<element name="head" minOccurs="1" maxOccurs="1"/>
<element name="to" minOccurs="1" maxOccurs="1"/>
```
Attribute Types

• The syntax of attribute types is:
  
  `<attribute name="..."/>
  
and they may have a number of optional attributes, such as types
  
  `type="..."`

or existence (corresponds to `#REQUIRED` and `#IMPLIED` in DTDs)
  
  `use="x"`, where `x` may be `optional` or `required` or prohibited,

or a default value (corresponds to `#FIXED` and default values in DTDs).
Attribute Types

• Examples:

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

- There is a variety of built-in data types
  - Numerical data types: `integer`, `short`, `Byte`, `long`, `float`, `decimal`
  - String types: `string`, `ID`, `IDREF`, `CDATA`, `language`
  - Date and time data types: `time`, `date`, `gMonth`, `gYear`
Data Types

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

• We discuss complex types first, deferring discussion of simple data types until we talk about restrictions.
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
A Data Type Example

• Example:

```xml
<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>
```

• The meaning is that an element in an XML document that is declared to be of type `lecturerType` may have a `title` attribute; it may also include any number of `firstname` elements and must include exactly one `lastname` element.
Already existing data types can be extended by new elements or attributes. Example:

```xml
<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>
```
The resulting data type looks like this:

```xml
<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>
```
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
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.
Example of Data Type Restriction

```xml
<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>
```
Restriction of Simple Data Types

• Simple data types can also be defined by restricting existing data types
  • For example, we can define a type `dayOfMonth` that admits values from 1 to 31 as follows:

```xml
<simpleType name="dayOfMonth">
  <restriction base="integer">
    <minInclusive value="1"/>
    <maxInclusive value="31"/>
  </restriction>
</simpleType>
```
Data Type Restriction: Enumeration

- It is also possible to define a data type by listing all the possible values - example: data type `dayOfWeek`:

```xml
<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>
```
Here we define an XML schema for email, so that it can be compared to the DTD provided earlier:

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

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

• Similar for bodyType
XML Schema: The Email Example

<complexType name="nameAddress">
    <attribute name="name" type="string" use="optional"/>
    <attribute name="address" type="string" use="required"/>
</complexType>
XML Schema: The Email Example

• Some data types can be defined anonymously (the types for the attachment element and the encoding attribute).
• In general, if a type is used only once, it makes sense to define it anonymously for local use.
Lecture Outline

- Introduction
- Detailed Description of XML
- Structuring
  - DTDs
  - XML Schema
- Namespaces
- Accessing, querying XML documents: XPath
- Transformations: XSLT
Namespaces

- One of the main advantages of using XML as a universal (meta) markup language is that information from various sources may be accessed.
  - 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
An Example

- Example, consider an (imaginary) joint venture (*vu* for virtual university) of an American university (say, Stony Brook University, *sbu*), and, an Australian university (say, Griffith University, *gu*), to present a unified view for online students.
An Example

<?xml version="1.0" encoding="UTF-16"?>
<vu:instructors
    xmlns:vu="http://www.vu.com/empDTD"
    xmlns:sbu="http://www.stonybrook.edu/empDTD"
    xmlns:gu="http://www.gu.au/empDTD">
    <sbu:faculty
        sbu:title="assistant professor"
        sbu:name="John Smith"
        sbu:department="Computer Science"/>
    <gu:academicStaff
        gu:title="lecturer"
        gu:name="Mate Jones"
        gu:school="Information Technology"/>
</vu:instructors>
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.
An Example

```xml
<?xml version="1.0" encoding="UTF-16"?>
<vu:instructors
    xmlns:vu="http://www.vu.com/empDTD"
    xmlns="http://www.sbu.edu/empDTD">
    xmlns:gu="http://www.gu.au/empDTD"

    <faculty
        title="assistant professor"
        name="John Smith"
        department="Computer Science"/>

    <gu:academicStaff
        gu:title="lecturer"
        gu:name="Mate Jones"
        gu:school="Information Technology"/>
</vu:instructors>
```
Lecture Outline

• Introduction
• Detailed Description of XML
• Structuring
  • DTDs
  • XML Schema
• Namespaces
• Accessing, querying XML documents: XPath
• Transformations: XSLT
Addressing and Querying XML Documents

• In relational databases, parts of a database can be selected and retrieved using SQL
  • Same necessary 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
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
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
• Consider the following XML document:

```xml
<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>
```
Its Tree Representation
Examples of Path Expressions in XPath

- Address all author elements
  \(/library/author\)
- Addresses all author elements that are children of the library element node, which resides immediately below the root
- Absolute path expression general form:
  - \(/t_1/\ldots/t_n\), where each $t_{i+1}$ is a child node of $t_i$, is a path through the tree representation
Examples of Path Expressions in XPath

- An alternative solution for the previous example is
  \[//\textit{author}\]
  - Address all author elements
  - Here \\ says that we should consider all elements in the document and check whether they are of type \textit{author}
  - This path expression addresses all \textit{author} elements anywhere in the document
    - this expression and the previous one lead to the same result in our example; however, they may lead to different results, in general
Examples of Path Expressions in XPath

- Address the **location** attribute nodes within **library** element nodes
  
  `/library/@location`

- The symbol `@` is used to denote attribute nodes
Examples of Path Expressions in XPath

- Address all \textit{title} attribute nodes within \textit{book} elements anywhere in the document, which have the value “Artificial Intelligence”

//book/@title="Artificial Intelligence"
Tree Representation of Query
Examples of Path Expressions in XPath

- Address all books with title “Artificial Intelligence”
  
  \[//book[@title="Artificial Intelligence"]\]

- Test within square brackets: a filter expression
  - It restricts the set of addressed nodes.

- Difference with the previous query
  - Previous Query collects **title** attribute nodes of **book** elements
  - This Query addresses **book** elements, the **title** of which satisfies a certain condition
Tree Representation of Query
Examples of Path Expressions in XPath

- Address the first `author` element node in the XML document:
  
  ```xml
  //author[1]
  ```

- Address the last `book` element within the first `author` element node in the document:
  
  ```xml
  //author[1]/book[last()]
  ```

- Address all `book` element nodes without a `title` attribute:
  
  ```xml
  //book[not (@title)]
  ```
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.
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: it denotes descendant or self
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
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 for path expressions
Lecture Outline

• Introduction
• Detailed Description of XML
• Structuring
  • DTDs
  • XML Schema
• Namespaces
• Accessing, querying XML documents: XPath
• Transformations: XSLT
Displaying XML Documents

• So far we have not provided any information about how XML documents can be displayed
  • Such information is necessary because unlike HTML documents, XML documents do not contain formatting information.

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

• may be displayed in different ways:

Grigoris Antoniou
University of Bremen
ga@tzi.de

Grigoris Antoniou
University of Bremen
ga@tzi.de
Style Sheets

• The advantage is that a given XML document can be presented in various ways when different style sheets are applied to it.

• 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 with which an input XML document is transformed to:
  - another XML document,
  - an HTML document, or
  - plain text
- The output document may use the same DTD or schema, or a completely different vocabulary
  - Generally XSLT is chosen when applications that use different DTDs or schemas need to communicate
- One way of defining the presentation of an XML document is to transform it into an HTML document
XSLT

- Move data and metadata from one XML representation to another
- XSLT can be used for machine processing of content without any regard to displaying the information for people to read.
- In the following we use XSLT only to display XML documents
XSLT Transformation into HTML

```xml
<?xml version="1.0" encoding="UTF-16"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <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>
</xsl:stylesheet>
```
The output of this style sheet, applied to the previous XML document, produces the following HTML document (which now defines the presentation):

```html
<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 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
Auxiliary Templates

• Suppose 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
Example of an Input Document

<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>
<?xml version="1.0" encoding="UTF-16"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <html>
            <head><title>Authors</title></head>
            <body bgcolor="white">
                <xsl:apply-templates select="authors"/>
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
Example of an Auxiliary Template

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

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

• The output produced is:

```html
<html>
<head><title>Authors</title></head>
<body bgcolor="white">
  <h2>Grigoris Antoniou</h2>
  Affiliation: University of Bremen<br>
  Email: ga@tzi.de
  <p>
  <h2>David Billington</h2>
  Affiliation: Griffith University<br>
  Email: David@gu.edu.net
  <p>
</body>
</html>
```
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 the element **xsl:apply-templates** applies to the root element.
    - i.e. the **authors** element (/ is located above the root element)
  - If the current context node is the **authors** element, then the element **xsl:apply-templates select="author"** causes the template for the **author** elements to be applied to all author children of the **authors** element.
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
• In this way, we work from the root to the leaves of the tree, and all templates are applied
Processing XML Attributes

• Suppose we wish to transform to itself the element:
  <person firstname="John" lastname="Woo"/>

• Let us attempt the easiest task imaginable, a transformation of the element to itself. One might be tempted to write:

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

• However, this is not a well-formed XML document because tags are not allowed within the values of attributes.
Processing XML Attributes

- We wish to add attribute values into template
- In XSLT, data enclosed in curly brackets take the place of the `xsl:value-of` element
- The correct way to define a template for this example is

```xml
<xsl:template match="person">
  <person
    firstname="{@firstname}"
    lastname="{@lastname}"/>
</xsl:template>
```
Transforming an XML Document to Another

- Finally, we give a transformation example from one XML document to another, which does not specify the display.
Transforming an XML Document to Another

```xml
<?xml version="1.0" encoding="UTF-16"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<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>

</xsl:stylesheet>
```
Transforming an XML Document to Another

```xml
<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>
```

```xml
</xsl:stylesheet>
```
Summary

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

- 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.