XML and Web Data

Facts about the Web

- Growing fast
- Popular
- Semi-structured data
  - Data is presented for ‘human’-processing
  - Data is often ‘self-describing’ (including name of attributes within the data fields)

Figure 17.1
A student list in HTML.

```html
<html>
  <head><title>Student List</title></head>
  <body>
    <h1>Students</h1>
    <table>
      <tr><th>Name</th><th>Id</th><th>Address</th></tr>
      <tr><td>John Doe</td><td>111111111</td><td>123 Main St</td></tr>
      <tr><td>Joe Public</td><td>666666666</td><td>666 Hollow Rd</td></tr>
    </table>
  </body>
</html>
```

Figure 17.2
Student list in object form.

```json
Object
  ([12345, {"Students": [{"John Doe": "111111111", [123,"Main St"]},
                         {"Joe Public": "666666666", [666,"Hollow Rd"]}]})
```

Vision for Web data

- **Object-like** – it can be represented as a collection of objects of the form described by the conceptual data model
- **Schemaless** – not conformed to any type structure
- **Self-describing** – necessary for machine readable data
XML – Overview

- Simplifying the data exchange between software agents
- Popular thanks to the involvement of W3C (World Wide Web Consortium – independent organization www.w3c.org)

XML – Characteristics

- Simple, open, widely accepted
- HTML-like (tags) but extensible by users (no fixed set of tags)
- No predefined semantics for the tags (because XML is developed not for the displaying purpose)
- Semantics is defined by stylesheet (later)

Figure 17.3
XML representation of the student list.

XML element

- Begin with a opening tag of the form
  `<XML_element_name>`
- End with a closing tag
  `</XML_element_name>`
- The text between the beginning tag and the closing tag is called the content of the element

XML Documents

- User-defined tags:
  `<tag> info </tag>`
- Properly nested: `<tag1>.. <tag2>… </tag1></tag2>` is not valid
- Root element: an element contains all other elements
- Processing instructions `<?command ….?>`
- Comments `<!-- comment -->`
- CDATA type
- DTD
Relationship between XML elements

- Child-parent relationship
  - Elements nested directly in an element are the children of this element (Student is a child of PersonList, Name is a child of Student, etc.)
- Ancestor/descendant relationship: important for querying XML documents (extending the child/parent relationship)

XML elements & Database Objects

- XML elements can be converted into objects by
  - considering the tag’s names of the children as attributes of the objects
  - Recursive process

XML elements & Database Objects

- Differences: Additional text within XML elements

XML elements & Database Objects

- Differences: XML elements are ordered

XML Attributes

- Can occur within an element (arbitrary many attributes, order unimportant, same attribute only one)
- Allow a more concise representation
- Could be replaced by elements
- Less powerful than elements (only string value, no children)
- Can be declared to have unique value, good for integrity constraint enforcement (next slide)

XML Attributes

- Can be declared to be the type of ID, IDREF, or IDREFS
- ID: unique value throughout the document
- IDREF: refer to a valid ID declared in the same document
- IDREFS: space-separated list of strings of references to valid IDs
**Well-formed XML Document**

- It has a root element
- Every opening tag is followed by a matching closing tag, elements are properly nested
- Any attribute can occur at most once in a given opening tag, its value must be provided, quoted

**So far**

- Why XML?
- XML elements
- XML attributes
- Well-formed XML document

**Namespaces and DTD**

**Namespaces**

- For avoiding naming conflicts
- Name of every XML tag must have two parts:
  - namespace: a string in the form of a uniform resource identifier (URI) or a uniform resource locator (URL)
  - local name: as regular XML tag but cannot contain `:`
- Structure of an XML tag:
  \[\text{namespace}:\text{local\_name}\]
Namespaces

- An XML namespace is a collection of names, identified by a URI reference, which are used in XML documents as element types and attribute names. XML namespaces differ from the "namespaces" conventionally used in computing disciplines in that the XML version has internal structure and is not, mathematically speaking, a set.
Source: www.w3c.org

Uniform Resource Identifier

- URI references which identify namespaces are considered identical when they are exactly the same character-for-character. Note that URI references which are not identical in this sense may in fact be functionally equivalent. Examples include URI references which differ only in case, or which are in external entities which have different effective base URIs.
Source: www.w3c.org

Namespace - Example

```xml
<item xmlns="http://www.acmeinc.com/jp#supplies" xmlns:toy="http://www.acmeinc.com/jp#toys">
  <name>backpack</name>
  <feature>
    <toy:item>
      <toy:name>cyberpet</toy:name>
    </toy:item>
  </feature>
</item>
```

Two namespaces are used: the two URLs
XML = defined the default namespace,
xmlls:toy = defined the second namespace

Namespace declaration

- Defined by
  ```xml
  xml : prefix = declaration
  ```
- Tags belonging to a namespace should be prefixed with "prefix:"
- Tags belonging to the default namespace do not need to have the prefix
- Have its own scope

Namespace declaration

```xml
<item xmlns="http://www.acmeinc.com/jp#supplies">
  <name>backpack</name>
  <feature>
    <toy:item>
      <toy:name>cyberpet</toy:name>
    </toy:item>
  </feature>
</item>
```

```xml
<item xmlns="http://www.acmeinc.com/jp#supplies2">
  <name>notebook</name>
  <feature>
    <toy:item>
      <toy:name>sticker</toy:name>
    </toy:item>
  </feature>
</item>
```

Document Type Definition

- Set of rules (by the user) for structuring an XML document
- Can be part of the document itself, or can be specified via a URL where the DTD can be found
- A document that conforms to a DTD is said to be valid
- Viewed as a grammar that specifies a legal XML document, based on the tags used in the document
DTD Components

- **A name** – must coincide with the tag of the root element of the document conforming to the DTD
- A set of **ELEMENTs** – one ELEMENT for each allowed tag, including the root tag
- **ATTLIST** statements – specifies the allow attributes and their type for each tag
- *, +, ? – like in grammar definition
  - * : zero or finitely many number
  - + : at least one
  - ? : zero or one

DTD Components – Element

<!ELEMENT Name definition>

Name of the element
definition can be: EMPTY, (#PCDATA), or element list (e1,c2, ...,en) where the list (e1,c2, ...,en) can be shortened using grammar like notation

DTD Components – Element

<!ELEMENT Name(e1, ...,en)>

n

1

Name of the element

<!ELEMENT PersonList (Title, Contents)>

<!ELEMENT Contents(Person *)>

Figure 17.5
A DTD for the report document in Figure 17.4.

<!DOCTYPE Report [  
  <!ELEMENT Report (Students, Classes, Courses)>  
  <!ELEMENT Students (Student)>  
  <!ELEMENT Classes (Class)>  
  <!ELEMENT Courses (Course)>  
  <!ELEMENT Student (Name, Status, Credits)>  
  <!ELEMENT Name (FirstName, LastName)>  
  <!ELEMENT First (#PCDATA)>  
  <!ELEMENT Credits EMPTY>  
  <!ELEMENT Class (CourseCode, Semester, ClassNumber)>  
  <!ELEMENT Course (CourseName)>  
  <!ELEMENT ClassNumber EMPTY>  
  <!ATTLIST ClassCode (#REQUIRED)>  
  <!ATTLIST ReportDate #IMPLIED>  
  <!ATTLIST StudentStatus ID #REQUIRED>  
  <!ATTLIST CourseCode ID #REQUIRED>  
  <!ATTLIST Credits ClassCode IDREFS #IMPLIED>  
  <!ATTLIST Credits Semester IDREF #REQUIRED>  
  <!ATTLIST ClassNumber IDREFS #IMPLIED>  
]>

Arbitrary number
DTD as Data Definition Language?

- Can specify exactly what is allowed on the document
- XML elements can be converted into objects
- Can specify integrity constraints on the elements
- Is is good enough?

Inadequacy of DTP as a Data Definition Language

- Goal of XML: for specifying documents that can be exchanged and automatically processed by software agents
- DTD provides the possibility of querying Web documents but has many limitations (next slide)

Inadequacy of DTP as a Data Definition Language

- Designed without namespace in mind
- Syntax is very different than that of XML
- Limited basic types
- Limited means for expressing data consistency constrains
- Enforcing referential integrity for attributes but not elements
- XML data is ordered; not database data
- Element definitions are global to the entire document

XML Schema

XML Schema – Main Features

- Same syntax as XML
- Integration with the namespace mechanism (different schemas can be imported from different namespaces and integrated into one)
- Built-in types (similar to SQL)
- Mechanism for defining complex types from simple types
- Support keys and referential integrity constraints
- Better mechanism for specifying documents where the order of element types does not matter

XML Document and Schema

A document conforms to a schema is called an instance of this schema and is said to be schema valid.

XML processor does not check for schema validity.
XML Schema and Namespaces

- Describes the structure of other XML documents
- Begins with a declaration of the namespaces to be used in the schema, including
  - http://www.w3.org/2001/XMLSchema
  - http://www.w3.org/2001/XMLSchema-instance
  - targetNamespace (user-defined namespace)

http://www.w3.org/2001/XMLSchema

- Identifies the names of tags and attributes used in a schema (names defined by the XML Schema Specification, e.g., schema, attribute, element)
- Understood by all schema aware XML processor
- These tags and attributes describe structural properties of documents in general

http://www.w3.org/2001/XMLSchema

- complexType
- element
- sequence
- schema
- integer
- boolean
- string

The names defined in XMLSchema

http://www.w3.org/2001/XMLSchema-instance

- Used in conjunction with the XMLSchema namespace
- Identifies some other special names which are defined in the XML Schema Specification but are used in the instance documents

Target namespace

- identifies the set of names defined by a particular schema document
- is an attribute of the schema element (targetNamespace) whose value is the name space containing all the names defines by the schema
Types

- Simple types (See Slides 56-68 of [RC])
  - Primitive
  - Deriving simple types
- Complex types

[RC] – Roger Costello’s Slide on XML-Schema

Built-in Datatypes (From [RC])

- Primitive Datatypes
  - string
  - boolean
  - decimal
  - float
  - double
  - array
  - dateTime
  - date
  - gYear
  - gMonth
  - gDay
  - time
  - dateTime
  - date
  - gYearMonth
  - gMonthDay
  - gDay
  - P1Y2M3DT10H30M12.3S

- Atomic, built-in
  - "Hello World"
  - {true, false}
  - 7.08
  - 12.56E3
  - 12
  - 12560
  - 0
  - -0
  - INF
  - -INF
  - NAN

- Built-in Datatypes (cont.)

- Derived types
  - normalizedString
  - token
  - language
  - ID
  - IDREF
  - IDREFS
  - ENTITY
  - ENTITIES
  - NMTOKEN
  - NMTOKENS
  - Name
  - NCName
  - xml:lang
  - date
  - dateTime
  - time
  - duration

- Subtype of primitive datatype
  - ID
  - IDREF
  - IDREFS
  - ENTITY
  - ENTITIES
  - NMTOKEN
  - NMTOKENS
  - Name
  - NCName
  - xml:lang

Include statement

```xml
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://xyz.edu/Admin">
  <include schemaLocation="http://xyz.edu/StudentTypes.xsd"/>
  <include schemaLocation="http://xyz.edu/ClassTypes.xsd"/>
  <include schemaLocation="http://xyz.edu/CoursTypes.xsd"/>
</schema>
```

Include the schema in the location ... to this schema
(good for combining)
**Built-in Datatypes (cont.)**

- Derived types
  - negativeInteger
  - long
  - int
  - short
  - byte
  - nonNegativeInteger
  - unsignedLong
  - unsignedInt
  - unsignedShort
  - unsignedByte
  - positiveInteger

  Note: the following types can only be used with attributes (which we will discuss later):
  - ID, IDREF, IDREFS, NMTOKEN, NMTOKENS, ENTITY, and ENTITIES.

**Simple types**

- **Primitive types (see built-in)**
- **Type constructors:**
  - List:
    <simpleType name="myIdrefs">
      <list itemType="IDREF" />
    </simpleType>
  - Union:
    <simpleType name="myIdrefs">
      <union memberTypes="phone7digits phone10digits" />
    </simpleType>
  - Restriction:
    <simpleType name="phone7digits">
      <restriction base="integer">
        <minInclusive value="1000000" />
        <maxInclusive value="9999999" />
      </restriction>
    </simpleType>

**Simple Types for Report Document**

- <simpleType name="studentIds">
    <list itemType="studentRef" />
  </simpleType>
- <simpleType name="courseCode">
    <restriction base="ID">
      <pattern value="[A-Z][0-9][A-Z][0-9]" />
    </restriction>
  </simpleType>
- <simpleType name="courseRef" />

**Type Declaration for Elements & Attributes**

- **Type declaration for simple elements and attributes**
  - <element name="CrsName" type="string" />

Specify that CrsName has value of type **string**
Type Declaration for Elements & Attributes

- Type declaration for simple elements and attributes
  
  `<element name="status" type="adm:studentStatus"/>

  Specify that status has value of type `studentStatus` that will be defined in the document

Example for the type `studentStatus`

  `<simpleType name="studentStatus">
   <restriction base="string">
     <enumeration value="U1"/>
     <enumeration value="U2"/>
     ...
     <enumeration value="G5"/>
   </restriction>
  </simpleType>`

Complex Types

- Use to specify the type of elements with children or attributes
- Opening tag: `complexType`
- Can be associated to a name in the same way a simple type is associated to a name

Complex Types

- Special Case: element with simple content and some attributes/no child with some attributes
  
  `<complexType name="CourseTakenType">
   <attribute name="CrsCode" type="adm:courseRef"/>
   <attribute name="Semester" type="string"/>
  </complexType>`

Complex Types

- Combining elements into group -- `<all>`
  
  `<complexType name="AddressType">
   <all>
     <element name="StreetName" type="string"/>
     <element name="StreetNumber" type="string"/>
     <element name="City" type="string"/>
   </all>
  </complexType>

The three elements can appear in arbitrary order! (NOTE: `<all>` requires special care – it must occur after `<complexType>` - see book for invalid situation)

Complex Types

- Combining elements into group -- `<sequence>`
  
  `<complexType name="NameType">
   <sequence>
     <element name="First" type="string"/>
     <element name="Last" type="string"/>
   </sequence>
  </complexType>

The two elements must appear in order
Complex Types

- Combining elements into group – `<choice>`
  `<complexType name="addressType">`
  `<choice>`
  `<element name="POBox" type="string">`
  `<sequence>`
    `<element name="Name" type="string">`
    `<element name="Number" type="string">`
  </sequence>`
  `</choice>`
  `</complexType>`

Either POBox or Name and Number is needed

- Can also refer to local type like – allowing different elements to have children with the same name (next slides)
  `[studentType – courseType] both have the “Name” element`
  `[studentType – personNameType] both have the “Name” element`

Figure 17.7
Definition of the complex type studentType.

```xml
<complexType name="studentType">
  <sequence>
    <element name="Name" type="..."/>
    <element name="Status" type="..."/>
    <element name="CrsTaken" type="..."/>
  </sequence>
  <attribute name="StudId" type="..."/>
</complexType>
```

Complex Types

- Importing schema: like `include` but does not require schemaLocation
  instead of
  `<include schemaLocation="http://xyz.edu/CoursTypes"/>`
  we can use
  `<import namespace="http://xyz.edu/CoursTypes"/>`

Complex Types

- Deriving new complex types by extension and restriction (for modifying imported schema)
  `<import namespace="http://xyz.edu/CoursTypes"/>`
  `<complexType name="courseType">`
  `<complexContent>`
    `<extension base="..."/>
    `<element name="syllabus" type="string"/>`
  </extension>`
  `</complexContent></complexType>`

The type that is going to be extended
Figure 17.8A
Student types at http://xyz.edu/StudentTypes.xsd.

Figure 17.8B (continued)
Student types at http://xyz.edu/StudentTypes.xsd.

Integrity Constraints
- ID, IDREF, IDREFS can still be used
- Specified using the attribute xpath (next)
- XML keys, foreign keys
- Keys are associated with collection of objects not with types

Integrity Constraints - Keys

```
<key name="PrimaryKeyForClass">
  <selector xpath="Classes/Class"/>
  <field xpath="CrsCode"/>
  <field xpath="Semester"/>
</key>
```

The key comprises of two elements (CrsCode and Semester) – both are children of Class

Integrity Constraints - Foreign key

```
<keyref name="XXX" refer="adm:PrimaryKeyForClass">
  <selector xpath="Students/Student/Crs Taken"/>
  <field xpath="@CrsCode"/>
  <field xpath="@Semester"/>
</keyref>
```

Source Collection: where the elements should satisfy the key specified by the "Prim ... Class"
XML Query Languages

- Market, convenience, …
- XPath, XSLT, XQuery: three query languages for XML
- XPath – simple & efficient
- XSLT – full feature programming language, powerful query capabilities
- XQuery – SQL style query language – most powerful query capabilities

XPath

- Idea comes from path expression of OQL in object databases
- Extends the path expressions with query facilities by allowing search condition to occur in path expressions
- XPath data model: view documents as trees (see picture), providing operators for tree traversing, use absolute and relative path expression
- A XPath expression takes a document tree, returns a set of nodes in the tree
XPath Expression - Examples

/Students/Student/CrsTaken – returns the set of references to the nodes that correspond to the elements CrsTaken
First or ./First refers to the node corresponds to the same child element First if the current position is Name
/Students/Student/CrsTaken/@CrsCode – the set of values of attributes CrsCode
/Students/Student/Name/First/text() – the set of contents of element First

Advanced Navigation

/Students/Student[1]/CrsTaken[2] – first Student node, second CrsTaken node
//CrsTaken – all CrsTaken elements in the tree (descendant-or-self)
Student/* - all e-children of the Student children of the current node
/Students/Student[search_expression] – all Student node satisfying the expressions; see what search_expression can be in the book!

XPointer

• Use the features of XPath to navigate within an XML document
• Syntax:
  someURL#xpointer(XPathExpr1)xpointer(XPathExpr2)...
  Example:
  http://www.foo.edu/Report.xml#xpointer((//Student[...]))

XSLT

• Part of XSL – an extensible stylesheet language of XML, a transformation language for XML: converting XML documents into any type of documents (HTML, XML, etc)
• A functional programming language
• XML syntax
• Provide instructions for converting/extracting information
• Output XML

XSLT Basics

• Stylesheet: specifies a transformation of one type of document into another type
• Specifies by a command in the XML document
<?xml version=“1.0”?>
<Report Date=“2002-03-01”>
  .... What parser should be used!
</Report>

XSLT - Example

<?xml version=“1.0”?>
<StudentList xmlns:xsl=“http://www.w3.org/1999/XSL/Transform” xsl:version=“1.0”>
  <xsl:copy-of select=“/Student/Name” />
</StudentList>

Result:
<StudentList>
  <Name>John</Name>
  <Last>Doe</Last>
</StudentList>
XSLT – Instructions

- copy-of
- if-then
- for-each
- value-of
- ...

XSLT – Instructions

```xml
<?xml version="1.0"?>
<StudentList xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xsl:version="1.0">
  <xsl:for-each select="/Student">
    <xsl:if test="count (CrsTaken) &gt; 1">
      <FullName> <xsl:value-of select="/Last"/>
        <xsl:value-of select="/First"/>
    </FullName>
    </xsl:if>
  </xsl:for-each>
</StudentList>
```

XSLT – Instructions

```xml
<?xml version="1.0"?>
<StudentList xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xsl:version="1.0">
  <xsl:for-each select="/Student">
    <xsl:if test="count (CrsTaken) &gt; 1">
      <FullName> <xsl:value-of select="/Last"/>
        <xsl:value-of select="/First"/>
    </FullName>
    </xsl:if>
  </xsl:for-each>
</StudentList>
```

XSLT – Template

- Recursive traversal of the structures of the document
- Often defined recursively
- Algorithm for processing a XSLT template (book)

Figure 17.12
Recursive stylesheet.

```xml
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xsl:version="1.0">
  <xsl:template match="/">
    <StudentList>
      <xsl:apply-templates/>
    </StudentList>
  </xsl:template>
  <xsl:template match="/Student">
    <xsl:apply-templates select="CrsTaken"/>
    <fullName>
      <xsl:value-of select="/Last"/>
      <xsl:value-of select="/First"/>
    </fullName>
  </xsl:template>
</xsl:stylesheet>
```

Figure 17.14
XSLT stylesheet that converts attributes into elements.

```xml
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xsl:version="1.0">
  <xsl:template match="node()">
    <xsl:copy>
      <xsl:apply-templates select="@*"/>
    </xsl:copy>
  </xsl:template>
  <xsl:template match="@*">
    <fullName>
      <xsl:value-of select="/name(current())"/>
      <xsl:value-of select="/value-of-select/*"/>
    </fullName>
  </xsl:template>
</xsl:stylesheet>
```
**XQuery**

- Syntax similar to SQL
  - **FOR** variable declaration
  - **WHERE** condition
  - **RETURN** result

---

**XQuery - Example**

FOR $t$ IN document("http://xyz.edu/transcripts.xml")
  //Transcript
  WHERE $t$/CrsTaken/@CrsCode = "MA123"
  RETURN $t$/Student

Find all transcripts containing “MA123”
Return the set of Student’s elements of those transcripts

---

**Putting it in well-formed XML**

```
<StudentList>
  (FOR $t$ IN document("http://xyz.edu/transcripts.xml")
    //Transcript
    WHERE $t$/CrsTaken/@CrsCode = "MA123"
    RETURN $t$/Student)
</StudentList>
```

---

**Figure 17.15**

Transcripts at http://xyz.edu/transcripts.xml.

```
<StudentList>
  (FOR $t$ IN document("http://xyz.edu/transcripts.xml")
    //Transcript
    WHERE $t$/CrsTaken/@CrsCode = "MA123"
    RETURN $t$/Student)
</StudentList>
```

**Figure 17.16**

Construction of class rosters from transcripts: first try.

```
CLASSROSTER($c$)
  ($c$/CrsTaken/@CrsCode = "MA123"
   AND $c$/CrsTaken/@Semester = "F2008"
   "A" OR $c$/CrsTaken/@Semester = "F2009"
   "A"
  )
  FOR $c$ IN document("http://xyz.edu/transcripts.xml")
  //Transcript
  WHERE $c$/CrsTaken/@CrsCode = "MA123"
  RETURN $c$/Student
  ORDER BY ($c$/Student/@StudId)
</CLASSROSTER>
```

For each class $c$, find the students attending the class and output his information
= $\exists$ output one class roster for each $CrsTaken$ node $\exists$ possibly more than one if different students get different grade
Fix?

- Assume that the list of classes is available – write a different query
- Use the filter operation

Filtering

- Syntax: filter(argument1, argument2)
- Meaning: return a document fragment obtained by
  - deleting from the set of nodes specified by argument1 the nodes that do not occur in argument2
  - reconnecting the remaining nodes according to the child-parent relationship of the document specified by argument1
Advances Features

- User-defined functions
- XQuery and Data types
- Grouping and aggregation

Figure 17.18
Class rosters constructed with user-defined functions.

```xml
<result>
  <ClassRoster CrsCode="C001" Semester="Fall2009">
    <Student StudID="S001">
      <Name>John Doe</Name>
    </Student>
    <Student StudID="S002">
      <Name>Jane Smith</Name>
    </Student>
  </ClassRoster>
  <ClassRoster CrsCode="C002" Semester="Spring2010">
    <Student StudID="S003">
      <Name>Mike Johnson</Name>
    </Student>
    <Student StudID="S004">
      <Name>Lisa Brown</Name>
    </Student>
  </ClassRoster>
</result>
```

Figure 17.19
XQuery transformation that does the same work as the stylesheet in Figure 17.14.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<result>
  <ClassRoster CrsCode="C001" Semester="Fall2009">
    <Student StudID="S001">
      <Name>John Doe</Name>
    </Student>
    <Student StudID="S002">
      <Name>Jane Smith</Name>
    </Student>
  </ClassRoster>
  <ClassRoster CrsCode="C002" Semester="Spring2010">
    <Student StudID="S003">
      <Name>Mike Johnson</Name>
    </Student>
    <Student StudID="S004">
      <Name>Lisa Brown</Name>
    </Student>
  </ClassRoster>
</result>
```