XML Summary
Why ...  

- Relational databases are not enough?  
  - Created for structured data  
- FOL knowledge bases are not enough?  
  - Complex reasoning
XML introduction

XML – Extensible Markup Language

- designed to describe semi-structured documents
- users may create their own tags (they can create their own specific languages)
- tags have no semantics indicating how to present documents through a Web browser
<? xml version="1.0" encoding="UTF-8" ?>
<book>
  <title>Semantic Web is Cool</title>
  <author>John Smith</author>
  <publisher>Springer</publisher>
  <year>1993</year>
</book>
XML prolog of a document

the prolog:
- an XML declaration
- an optional reference to external structuring documents

```xml
<?xml version="1.0" encoding="UTF-8"?>
```
XML elements

“things” the XML document talks about

- books, authors, publishers, ...

Each element contains up to three parts

- an opening tag, the content, a closing tag

  `<author>John Smith</author>`

- **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 (“Xml”, “xml”)
XML

content of elements

cContent may be text, or other elements, or nothing

<author>
  <name>John Smith</name>
  <phone> +1 − 780 − 492 5507 </phone>
</author>

if no content

<author/> for <author></author>
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

<author name="John Smith" phone="+1 − 780 − 492 5507"/>
Attributes vs. elements

- If the information in question could be itself marked up with elements, put it in an element.
- If the information is suitable for attribute form, but could end up as multiple attributes of the same name on the same element, use child elements instead.
- If the information is required to be in a standard DTD-like attribute type such as ID or IDREF, use an attribute.
- If the information should not be normalized for white space, use elements. (XML processors normalize attributes in ways that can change the raw text of the attribute value.)
XML

other components

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

- processing instructions (define procedural attachments)

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

well-formed documents

correct documents that obey some syntactic rules:

- there is only one outermost element (called root element)
- each element has an opening and a corresponding closing tag
- tags may not overlap

  `<author><name>Lee Hong</name></author>`

- attributes have unique names
- names of elements and tags must be permissible
<email>
  <head>
    <from name="John Smith"
        address="johnsmith@gmail.com"/>
    <to name="Jenny Doe"
        address="jennydoe@hotmail.com"/>
    <subject>How are you?</subject>
  </head>
  <body>
    Hi, it was nice ...
  </body>
</email>
Exercise: Draw the previous document as a tree!
XML

structure of documents

definition of all element and attribute names that may be used
definition of 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
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)
XML
structure of documents: DTD by example

<author>
  <name>John Smith</name>
  <phone> +1 − 780 − 492 5507 </phone>
</author>

DTD for above element (and all author elements):

<!ELEMENT author (name,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
XML Schema

- richer language for defining the structure of XML documents
- its syntax is based on XML itself
- sophisticated set of data types, compared to DTDs (which only supports strings)
- it is like an element with an opening tag like

```xml
<xsd:schema
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 version="1.0">
 ...
</xsd:schema>
```
XML Schema

element types - examples

<element name="email"/>

<element name="head" minOccurs="1" maxOccurs="1"/>

<element name="to" minOccurs="1"/>
XML Schema
attribute types - examples

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

existence: \texttt{use="x"}, where \texttt{x} may be \texttt{optional} or \texttt{required}

default value: \texttt{use="x" value="..."}, where \texttt{x} may be \texttt{default} or \texttt{fixed}
XML Schema
data types

built-in data types
- numerical data: `integer`, `Short`, ...
- string: `string`, `ID`, `IDREF`, `CDATA`, ...
- date and time: `time`, `Month`, ...

+ user-defined data types
- simple data types, which cannot use elements or attributes
- complex data types, which can use these

CDATA: text ignored by the XML-parser
XML Schema

data types (2)

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
### XML Schema file

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:x="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.example.org/contactExample"
  xmlns:tns="http://www.example.org/contactExample"
  elementFormDefault="qualified">

  <xs:element name="Contacts">
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded">
        <xs:element name="Person">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="PersonGivenName"/>
              <xs:element name="PersonFamilyName"/>
              <xs:element name="PersonBirthDate"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

### A compliant XML instance file example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ex:Contacts xmlns:ex="http://www.example.org/contactExample"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.example.org/contactExample contactExample.xsd">
  <ex:Person>
    <ex:PersonGivenName/>
    <ex:PersonFamilyName/>
    <ex:PersonBirthDate/>
  </ex:Person>
</ex:Contacts>
```
XML example

- Model a XML-database about students with common information from the university domain (name, studentID, attended lectures, ..):
  1) Define an appropriate XML-Schema
  2) Given one instantiation (document) conforming the schema
XML Schema

namespaces

- a single XML document may use more than one DTD or schema
- in order to avoid clashes a different prefix for each DTD or schema can/should be used

prefix:name
XML Schema

namespaces – example

<...>
xmlns="http://www.ua.ca/basic.xsd"
xmlns:staff="http://www.ua.ca/staff.xsd">

<staff:faculty staff:title="professor"
staff:name="John Smith"
staff:department="ECE"/>

<academicStaff title="lecturer"
name="Jenny Doe"
school="Information Technology"/>

</...>
Working with XML

- Two possibilities
  1) Document Object Model
  2) Simple API for XML
Document Object Model (DOM)

- Can represent HTML, XHTML, XML, ...
- „It is all about traversing hierarchies“
- Not good for big documents!
  - Has to be completely parsed!
Document Object Model

Logical View

DOM (Core) Level One
Invoke "getName" to read instance variable name when using XML::DOM or XML4J

- Document
  - DocumentNode
  - DocumentType
- DocumentFragment
- Entity
  - EntityReference
  - ProcessingInstruction
- Node
  - Attr
  - CharacterData
  - NamedNodeMap
- Element
  - Node
  - Text
  - Comment
  - CDATASection
  - Document
  - DocumentFragment
  - DocumentType
  - Entity
  - EntityReference
  - ProcessingInstruction
- Notation
- XML:Parser
- "u_long" = "unsigned long"
Simple API for XML (SAX)

- Event-based API via callback-functions
  - „Opening tag“, „Closing tag“, „Attribute“,
  - No **complete** elements
- No formal specification
- Very fast and good for large documents

- Also hybrid solutions:
  - persistent DOM, cached SAX
  - StAX (more control, e.g. skip sections)
Simple API for XML (SAX)

```xml
<?xml version="1.0"?>
<data>
    <entry id="a1">
        <foo>abc</foo>
        <foo>xyz</foo>
    </entry>
    <entry id="a2">
        <foo>bar</foo>
        <foo>baz</foo>
    </entry>
    ...
</data>
```

```plaintext
startDocument
data
startElement (entry id="a1")
startElement (foo)
characters (abc)
endElement (foo)
startElement (foo)
characters (xyz)
endElement (foo)
startElement (entry)
startElement (entry id="a2")
startElement (foo)
characters (bar)
endElement (foo)
startElement (foo)
characters (baz)
endElement (foo)
endElement (entry)
...
data
endDocument
```
Plenty of Technologies on top of XML

- **XPath**: Address elements in a XML document
  - E.g.: `/bookstore/book[3]`
- **XQuery**: Complex query language (similar to SQL)
  - E.g.:
    - `for $x in doc("books.xml")/bookstore/book`  
      `where $x/price>30`  
      `order by $x/title`  
      `return $x/title`
- **Further**: XSL, XLink, XPointer, ...
XML: more than a tree!

- IDs/IDREFs let you model a whole graph!

```
<person ID="o123">
    <firstname>John</firstname>
    <lastname>Smith</lastname>
</person>
<person ID="o234"> ... </person>
<article author="o123 o234">
    <title> ... </title>
    <year> 1995 </year>
</article>
```
XML Critics

- 10-15 years ago:
  - Not well specified,
  - Overspecified,
  - no tool support,
  - slow,
  - non-intuitive,
  - complicated error handling, ...

- As of 2012:
  - Full specification is needlessly(?) complicated
    - Especially namespaces and string/whitespace normalization
MicroXML

- W3C community group as of 2012
  - Simplified namespaces
  - improved CDATA handling
  - a lot of work on string normalization

```html
<!DOCTYPE html>
<!xml-stylesheet type="text/css" href="mystyle.css"?>
<html lang="en">
  <!-- A comment -->
  <head>
    <title>Welcome page</title>
  </head>
  <body>
    <p>Welcome to <a href="ibm.com/developerworks/">IBM developerWorks</a>.</p>
  </body>
</html>
```
XML is a mature technology with many nasty details