Tutorial on XML

21st April, 2004

Chun-Nan Hsu 許鈞南
Institute of Information Science
Academia Sinica
Taipei, Taiwan
Tutorial on XML

Contents

- XML
- DOM
- DTD
- CSS
- XSL / XSLT
- XML Schema
- RDF
- Logics / Description logics
- Ontology
- OWL
Tutorial on XML

Contents

- XML
- DOM
- DTD
- XSL / XSLT
- XML Schema
- RDF
- Logics / Description logics
- Ontology
- OWL
Why XML?

• XML = eXtensible Markup Language
• Designed by W3C
• Designed for two main purposes:
  – For machines/computer programs to “understand” the data on the Web
  – For data exchange, interoperability across various applications
• Applications
  – Data integration: answering queries by accessing multiple Web sites
  – Serving as the basis of standard data formats
What information can we see…

WWW2002
The eleventh international world wide web conference
Sheraton waikiki hotel
Honolulu, hawaii, USA
7-11 may 2002
1 location 5 days learn interact
Registered participants coming from
australia, canada, chile denmark, france, germany, ghana, hong kong, india,
ireland, italy, japan, malta, new zealand, the netherlands, norway,
singapore, switzerland, the united kingdom, the united states, vietnam,
zaire
Register now
On the 7th May Honolulu will provide the backdrop of the eleventh
international world wide web conference. This prestigious event …

Speakers confirmed
Tim berners-lee
Tim is the well known inventor of the Web, ...
Ian Foster
Ian is the pioneer of the Grid, the next generation internet …
What information can a machine see...
Solution: XML markup with “meaningful” tags?

<name>

<location>

$date>

<slogan>

<participants>

<introduction>

<speaker>

<bio>
But What About…

<introduction>5LNPŒLˆ "†Š 2" OL O 0H` +†"†z‹z‹ ŠPzz ‡ˆ†™PKL OL IHJRKˆ†‡ †M OL LzL™L"O P"Lˆ"HP†"Hz ІˆzK ŠPKL ŠLI J†"MLˆL"JL 7OPŒ ‡ˆLŒPNP†‹Œ L™L" g 6‡LHRLˆŒ J†"MPˆLK</introduction>

<speaker>7P ILˆ"LˆŒzLL</speaker>

<bio>7P PŒ OL ŠLzz R"†Š" P"™L"†ˆ †M OL >LI</bio>
Machine sees...
Need to Add “Semantics”

- **External agreement** on meaning of annotations
  - E.g., Dublin Core, Gene Ontology, etc.
    - Agree on the meaning of a set of annotation tags
  - Problems with this approach
    - Inflexible
    - Limited number of things can be expressed

- **Use Ontologies to specify meaning of annotations**
  - Ontologies provide a vocabulary of terms
  - New terms can be formed by combining existing ones
  - Meaning *(semantics)* of such terms is formally specified
  - Can also specify relationships between terms in multiple ontologies

XML is the first step toward adding “semantics” to the Web
Components of XML

- **Document Type Definition**
- **XML Document**
- **eXensible Style Language**

**What’s the Schema**

**What’s the Content**

**How to Format**

**What We see**

An XML page

---

Institute of Information Science
Academia Sinica, Taiwan
XML Syntax

• **Elements and Attributes**
  
  ```xml
  <person sex="female">
    <lastname>Tove</lastname>
    <firstname>Jani</firstname>
    <age>24</age>
    <mail>janit@nowhere.com</mail>
  </person>
  ```

• **Prolog**
  
  – **XML declaration:** version info, encoding declaration
    ```xml
    <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
    ```
  
  – **Document type declaration**
    ```xml
    <!DOCTYPE DOCBOOK SYSTEM "http://www.davenport.org/docbook">
    ```
What is and what is not XML

• XML is free and extensible
  – XML tags are not predefined. You must “invent” your own tags
• XML is a complement to HTML, not a replacement for HTML
• XML does not DO anything, you need to write or use a piece of software to actually process XML data
• XML tags are case sensitive
• XML elements must be properly nested
• XML documents must have a root tag
Tutorial on XML

Contents

- XML
- DOM
- DTD
- XSL / XSLT
- XML Schema
- RDF
- Logics / Description logics
- Ontology
- OWL
What is DOM

• DOM = the XML Document Object Model is a programming interface for XML documents
• With XML DOM, a programmer can create an XML document, navigate its structure, and add, modify or delete its elements
• A program calls an XML parser to load an XML document into the memory of your computer. When the document is loaded, its information can be retrieved and manipulated by access the DOM
• The DOM represents a tree view of the XML document
Tutorial on XML

Contents

• XML
• DOM
• DTD
• XSL / XSLT

• XML Schema
• RDF
• Logics / Description logics
• Ontology
• OWL
What is DTD

- DTD = document type declaration
- The purpose of a DTD is to define the legal building blocks of an XML document. It defines the document structure with a list of legal elements.
- When you try to open an XML document, the XML Parser will validate your XML document according to your XML document’s DTD.
- Internet Explorer 5.0+ can validate your XML document against a DTD
Declaring DTD

- **DTD can be included in your XML document**
  ```xml
  <!DOCTYPE label[
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT address (#PCDATA)>
  ]>
  <label>
    <name> Rock N. Robyn</name>
    <address> Jay Bird Street ... </address>
  </label>
  ```

- **DTD can be included from an external DTD**
  ```xml
  <!DOCTYPE LAVEL SYSTEM "http://www.sgmlsource.com/dtds/lavel.dtdfile
  <LABEL>
    ---
  </LABEL>
  ```
Element Type Declarations

- Example
  
  ```
  <!ELEMENT memo (to, from, body)>
  ```

- Element type content specification
  
  - **EMPTY content**: `<!ELEMENT empty-element EMPTY>`
  
  - **ANY content**: `<!ELEMENT any-element ANY>`
    
    This element can contain any combination of parsable data
  
  - **Mixed content**
    
    ```
    <!ELEMENT emph (#PCDATA)>
    <!ELEMENT title (#PCDATA|emph)]->
    <title>this is a <emph>XML</emph> book</title>
    ```

- Element content

- Content models
  
  - Occurrence Indicators: `?`(optional), `*`(optional and repeatable), `+`(required and repeatable)
    
    ```
    <!ELEMENT FIGURE (CAPTION?, (CODE|TABLE|FLOW-CHART))>
    ```
Tutorial on XML

Contents

- XML
- DOM
- DTD
- XSL / XSLT
- XML Schema
- RDF
- Logics / Description logics
- Ontology
- OWL
Extensible Stylesheet Language (XSL)

- XSL is a language for expressing stylesheets
  - XSLT
    - Transformation of XML document
    - [http://www.w3.org/TR/xslt](http://www.w3.org/TR/xslt)
  - XPath
    - An expression language used by XSLT to locate elements and/or attributes within an XML document
    - [http://www.w3.org/TR/xpath](http://www.w3.org/TR/xpath)
  - XSL-FO (Formatting Objects)
    - Specifies the formatting properties for rendering the document
    - [http://www.w3.org/TR/XSL/XSL-FO](http://www.w3.org/TR/XSL/XSL-FO)
    - XSL-FO is now formally named XSL
XSL Transformations (XSLT)

- XSLT applies user-defined transformations to an XML document
  - Transformed output can be:
    - HTML, XML, WML, etc.
XSLT Stylesheet Elements

- Matching and selection templates
  - Xsl: template
  - Xsl:apply-templates
  - Xsl:value-of
- Branching elements
  - Xsl:for-each
  - Xsl:if
  - Xsl:choose
    - Xsl:when
    - Xsl:otherwise
  - Xsl:sort
XSLT for-each Element

- **Xsl: for-each select=“expression”**
  - Process each node selected by the XPath expression
XSLT if Element

- **Xsl: if test=“expression”**
  - Evaluates the expression to a boolean and if true, applies the template body
  - **XSLT has no if-else construct (use choose)**
XSLT choose Element

- Xsl: choose
  - Select any number of alternatives
  - Instruction to use in place of *if-else* or *switch* construct found in other programming languages
Sort Element

- The sort element specifies that elements are sorted on a given property

```xml
<list>
  <item>ZZZ</item>
  <item>AAA</item>
  <item>MMM</item>
</list>

<xsl:template match="list">
  <xsl:apply-templates>
    <xsl:sort/>
  </xsl:apply-templates>
</xsl:template>

<list>
  <item code="Z">…</item>
  <item code="A">…</item>
  <item code="M">…</item>
</list>

<xsl:template match="list">
  <xsl:apply-templates>
    <xsl:sort select="@code"/>
  </xsl:apply-templates>
</xsl:template>
```
XSLT output Element

- **Xsl: output**
  - Controls the format of the stylesheet output
  - Useful attributes
    - Method = “[html| xml| text]”
    - Indent = “[yes| no]”
    - Version = “version”
    - Doctype-public = “string”
    - Encoding = “string”
- **Example**
  
  `<xsl: output method="html"
    doctype-public="-//W3C//DTD HTML 4.0 Transitional//EN”>`
Variable Element

- Variables may be declared and used in XSLT
  - `<xsl:variable name="colour">red</xsl:variable>`
  - Can also use `<value-of select="attribute"/>

- Variable is referenced with $ notation
  - `<xsl:value-of select="$colour"/>

- Can be used in output elements too
  - `<ajr:glyph colour="($colour)"/>"
XSLT Advantages and Disadvantages

• Advantages
  – Easy to merge XML data into a presentation
  – More resilient to changes in the details of the XML documents than low-level DOM and SAX
  – Database queries can be retuned in XML

• Disadvantages
  – Memory intensive and suffers a performance penalty
  – Difficult to implement complicated business rules
  – Have to learn a new language
  – Can’t change the value of variables (requires recursion)
  – Hard to debug
  – Don’t have full power of, say, Java inside templates

• No database access, hashtables, methods, objects, etc.
XSLT Parsers

- Apache Xalan  
  - http://xml.apache.org/xalan
- Oracle  
  - http://technet.oracle.com/tech/xml
- Saxon  
- Microsoft  
  - http://www.microsoft.com/xml
Tutorial on XML

Contents

• XML
• DOM
• DTD
• XSL / XSLT

• XML Schema
• RDF
• Logics / Description logics
• Ontology
• OWL
"Well Formed" XML documents

- A "Well Formed" XML document has correct XML syntax.
- All XML elements must have a closing tag
  `<p>This is a paragraph</p>`
- XML tags are case sensitive
  `<Message>This is incorrect</message>`
- All XML elements must be properly nested
  `<b><i>This text is bold and italic</i></b>`
- Attribute values must always be quoted
  `<note date="12/11/2002">`
"Valid" XML documents

- A "Valid" XML document is a "Well Formed" XML document, which also conforms to the rules of a Document Type Definition (DTD) or XML Schema.
XML Namespaces (1)

- Name Conflicts
  - Since element names in XML are not fixed, very often a name conflict will occur when two different documents use the same names describing two different types of elements.

```xml
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>

<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```
XML Namespaces (2)

• Using Namespaces
  – xmlns:namespace-prefix="namespace"
  – Example : xmlns:f="http://www.w3schools.com/furniture"
  – The W3C namespace specification states that the namespace itself should be a Uniform Resource Identifier (URI).
  – This URI simply serves as a unique identifier of a namespace. The XML parser will not use the URI to access the Web.
  – However, the URI usually points to a Web site that contains information about the defined namespace.
  – When a namespace is defined in the start tag of an element, all child elements with the same prefix are associated with the same namespace.
XML Namespaces (3)

- Using Namespaces

```xml
<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>
```

```xml
<f:table xmlns:f="http://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```
What is an XML Schema?

• **Answer:** An XML vocabulary for expressing your data’s business rules  
  (Roger L. Costello)

• **An XML Schema:**
  – defines elements that can appear in a document
  – defines attributes that can appear in a document
  – defines which elements are child elements
  – defines the order of child elements
  – defines the number of child elements
  – defines data types for elements and attributes
  – defines default and fixed values for elements and attributes
Example

<location>
    <latitude>32.904237</latitude>
    <longitude>73.620290</longitude>
    <uncertainty units="meters">2</uncertainty>
</location>

Is this data valid?
1. The **latitude** must be a decimal with a value between -90 to +90
2. The **longitude** must be a decimal with a value between -180 to +180
3. For both latitude and longitude the number of digits to the right of the decimal point must be **exactly six digits**.
4. The value of uncertainty must be a **non-negative** integer
5. The uncertainty units must be either **meters** or **feet**.

**We can express all these data constraints using XML Schemas !!**
Validating your data

<location>
  <latitude>32.904237</latitude>
  <longitude>73.620290</longitude>
  <uncertainty units="meters">2</uncertainty>
</location>

XML Schema validator → Data is ok!

- check that the latitude is between -90 and +90
- check that the longitude is between -180 and +180
- check that the fraction digits is 6 for lat and lon

XML Schema

AIIA Lab
Institute of Information Science
Academia Sinica TAIWAN
XML Schemas vs. DTDs (1)

- XML Schemas will be used in most Web applications as a replacement for DTDs.

- XML Schema has Support for Data Types
  - It is easier to validate the correctness of data
  - It is easier to work with data from a database
  - It is easier to define restrictions on data
  - It is easier to define data patterns
  - It is easier to convert data between different data types
  - A Schema supports object-oriented concepts
XML Schemas vs. DTDs (2)

- XML Schemas use XML Syntax
  - You don’t have to learn another language
  - You can use your XML editor to edit your Schema files
  - You can use your XML parser to parse your Schema files
  - You can transform your Schema with XSLT
XML Schemas vs. DTDs (3)

- Using DTD
  ```xml
  <?xml version="1.0"?>
  <!DOCTYPE note SYSTEM "http://www.w3schools.com/dtd/note.dtd">
  <note> … </note>
  ```

- Using Schema
  ```xml
  <?xml version="1.0"?>
  <note xmlns="http://www.w3schools.com"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation= "http://www.w3schools.com/schema/note.xsd">
  <note> … </note>
  ```
A Simple XML Document

<?xml version="1.0"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>

<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
Define a Simple Element (1)

- The syntax
  - `<xs:element name="xxx" type="yyy"/>

- Example
  - `<lastname>Refsnes</lastname>
    <age>34</age>
    <dateborn>1968-03-27</dateborn>

  - `<xs:element name="lastname" type="xs:string"/>
    <xs:element name="age" type="xs:integer"/>
    <xs:element name="dateborn" type="xs:date"/>`
Define a Simple Element (2)

- Common XML Schema Data Types
  - String, Decimal, Integer, Boolean, Date, Time
- Declare Default and Fixed Values
  - `<xs:element name="color" type="xs:string" default="red"/>`
  - `<xs:element name="color" type="xs:string" fixed="red"/>`
Define an Attribute (1)

- The syntax
  - `<xs:attribute name="xxx" type="yyy"/>`
- Example
  - `<lastname lang="EN">Smith</lastname>`
  - `<xs:attribute name="lang" type="xs:string"/>`
Define a Attribute (2)

- Common XML Schema Data Types
- Declare Default and Fixed Values for Attributes
- Creating Optional and Required Attributes
  - `<xs:attribute name="lang" type="xs:string" use="optional"/>
  - `<xs:attribute name="lang" type="xs:string" use="required"/>
Deriving New Types (1)

- Restrictions on Values

```xml
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="100"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```
Deriving New Types (2)

• Restrictions on a Set of Values

```xml
<xs:element name="car">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Audi"/>
      <xs:enumeration value="Golf"/>
      <xs:enumeration value="BMW"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```
Deriving New Types (3)

- Restrictions on a Series of Values

```xml
<xsd:element name="letter">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[a-z]"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

<xsd:element name="letter">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="([a-z])*"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```
Deriving New Types (4)

- Restrictions on White Space Characters

```xml
<xs:element name="address">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:WhiteSpace value="preserve"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Preserve, Replace, Collapse
Deriving New Types (5)

• Restrictions on Length

```xml
<xs:element name="address">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:length value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

• Some Useful Datatypes
  – Country Code (ISO 3166), Language Code (ISO 639), Domain Name, URL …
Define Complex Elements

- What is a Complex Element?
  - A complex element is an XML element that contains other elements and/or attributes.
  - There are four kinds of complex elements:
    - empty elements
    - elements that contain only other elements
    - elements that contain only text
    - elements that contain both other elements and text
empty elements

```xml
<product prodid="1345" />

<xs:element name="product" type="prodtype"/>
<xs:complexType name="prodtype">
    <xs:attribute name="prodid" type="xs:positiveInteger"/>
</xs:complexType>
```
elements that contain only other elements

```xml
<xs:element name="person" type="persontype"/>
<xs:complexType name="persontype">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string" maxOccurs="10"/>
  </xs:sequence>
</xs:complexType>

<person>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</person>
```
elements that contain both other elements and text

<xs:element name="letter" type="letterytype"/>
<xs:complexType name="letterytype" mixed="true">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="orderid" type="xs:positiveInteger"/>
    <xs:element name="shipdate" type="xs:date"/>
  </xs:sequence>
</xs:complexType>

<letter>
  Dear Mr.<name>John Smith</name>.
  Your order <orderid>1032</orderid> will be shipped on <shipdate>2001-07-13</shipdate>.
</letter>
Group indicators

```xml
<xs:group name="persongroup">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
    <xs:element name="birthday" type="xs:date"/>
  </xs:sequence>
</xs:group>

<xs:attributeGroup name="personattrgroup">
  <xs:attribute name="firstname" type="xs:string"/>
  <xs:attribute name="lastname" type="xs:string"/>
  <xs:attribute name="birthday" type="xs:date"/>
</xs:attributeGroup>
```
Creating Schemas From Multiple Documents

• Advantages
  – We are able to modularize the development of our own schemas. (re-use common schema components)
  – We can construct from other author’s schemas

```xml
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" … >
    …
    <xs:element name="to" type="xs:string"/>
    …
</xs:schema>
```
Schema and XSLT (1)

- How does XSLT Works

(http://www.cs.rit.edu/~sls6076/xml/xslt.htm)
Schema and XSLT (2)

- XSLT allowing you to do …
  - Extract application information **form the schema**
  - Present instance **using information form a Schema**
  - Generate a basic schema form a XML document
Schema and Schematron (1)

- The Schematron differs in basic concept from other schema languages in that it is not based on grammars but on finding tree patterns in the parsed document.
- This approach allows many kinds of structures to be represented which are inconvenient and difficult in grammar-based schema languages.
- The Schematron allows you to develop and mix two kinds of schemas:
  - *Report* elements allow you to diagnose which variant of a language you are dealing with.
  - *Assert* elements allow you to confirm that the document conforms to a particular schema.
Schema and Schematron (2)

- The Schematron is based on a simple action:
  - First, find a context nodes in the document (typically an element) based on XPath path criteria;
  - Then, check to see if some other XPath expressions are true, for each of those nodes.
Tutorial on XML

Contents

• XML
• DOM
• DTD
• XSL / XSLT

• XML Schema
• RDF
• Logics / Description logics
• Ontology
• OWL
RDF and RDFS

- RDF stands for Resource Description Framework
- It is a W3C candidate recommendation (http://www.w3.org/RDF)
- RDF is **graphical formalism** (+ XML syntax + semantics)
  - for representing metadata
  - for describing the semantics of information in a machine-accessible way
- RDFS extends RDF with "**schema vocabulary**", e.g.:
  - Class, Property
  - type, subClassOf, subPropertyOf
  - range, domain
The RDF Data Model

- Statements are <subject, predicate, object> triples:
  <Ian, hasColleague, Uli>
- Can be represented as a graph:

- Statements describe properties of resources
- A resource is any object that can be pointed to by a URI:
  - a document, a picture, a paragraph on the Web;
  - a book in the library, a real person (?)
  - isbn://5031-4444-3333
  - ...
- Properties themselves are also resources (URIs)
URIs

- URI = Uniform Resource Identifier
- "The generic set of all names/addresses that are short strings that refer to resources"
- URLs (Uniform Resource Locators) are a particular type of URI, used for resources that can be accessed on the WWW (e.g., web pages)
- In RDF, URIs typically look like “normal” URLs, often with fragment identifiers to point at specific parts of a document:
  - http://www.somedomain.com/some/path/to/file#fragmentID
Linking Statements

• The subject of one statement can be the object of another
• Such collections of statements form a directed, labeled graph

Ian \(\rightarrow\) Uli \\
hasColleague

Uli \(\rightarrow\) Carole \\
hasColleague

Carole \(\rightarrow\) http://www.cs.mam.ac.uk/~sattler \\
hasHomePage

• Note that the object of a triple can also be a “literal” (a string)
RDF Syntax

- RDF has an XML syntax that has a specific meaning:
- Every `<Description>` element describes a resource
- Every attribute or nested element inside a `<Description>` is a property of that Resource
- We can refer to resources by using URIs

```xml
<Description about="some.uri/person/ian_horrocks">
    <hasColleague resource="some.uri/person/uli_sattler"/>
</Description>
<Description about="some.uri/person/uli_sattler">
    <hasHomePage>http://www.cs.mam.ac.uk/~sattler</hasHomePage>
</Description>
<Description about="some.uri/person/carole_goble">
    <hasColleague resource="some.uri/person/uli_sattler"/>
</Description>
```
RDF Schema (RDFS)

- RDF gives a formalism for meta data annotation, and a way to write it down in XML, but it does not give any special meaning to vocabulary such as `subClassOf` or `type`.
  - Interpretation is an arbitrary binary relation.

- RDF Schema allows you to define vocabulary terms and the relations between those terms.
  - It gives “extra meaning” to particular RDF predicates and resources.
  - This “extra meaning”, or semantics, specifies how a term should be interpreted.
RDFS Examples

- RDF Schema terms (just a few examples):
  - Class
  - Property
  - type
  - subClassOf
  - range
  - domain

- These terms are the RDF Schema building blocks (constructors) used to create vocabularies:
  <Person,type,Class>
  <hasColleague,type,Property>
  <Professor,subClassOf,Person>
  <Carole,type,Professor>
  <hasColleague,range,Person>
  <hasColleague,domain,Person>
RDF/RDFS “Liberality”

• No distinction between classes and instances (individuals)
  
  \(<\text{Species}, \text{type}, \text{Class}>\>
  
  \(<\text{Lion}, \text{type}, \text{Species}>\>
  
  \(<\text{Leo}, \text{type}, \text{Lion}>\>

• Properties can themselves have properties
  
  \(<\text{hasDaughter}, \text{subPropertyOf}, \text{hasChild}>\>
  
  \(<\text{hasDaughter}, \text{type}, \text{familyProperty}>\>

• No distinction between language constructors and ontology vocabulary, so constructors can be applied to themselves/each other
  
  \(<\text{type}, \text{range}, \text{Class}>\>
  
  \(<\text{Property}, \text{type}, \text{Class}>\>
  
  \(<\text{type}, \text{subPropertyOf}, \text{subClassOf}>\>
RDF/RDFS Semantics

- RDF has “Non-standard” semantics in order to deal with this
- Semantics given by RDF Model Theory (MT)
Tutorial on XML

Contents

• XML
• DOM
• DTD
• XSL / XSLT

• XML Schema
• RDF
• Logics / Description logics
• Ontology
• OWL
Ontology Languages

- **Wide variety of languages for “Explicit Specification”**
  - **Graphical notations**
    - Semantic networks
    - Topic Maps (see http://www.topicmaps.org/)
    - UML
    - RDF
  - **Logic based**
    - Description Logics (e.g., OIL, DAML+OIL, OWL)
    - Rules (e.g., RuleML, LP/Prolog)
    - First Order Logic (e.g., KIF)
    - Conceptual graphs
    - (Syntactically) higher order logics (e.g., LBase)
    - Non-classical logics (e.g., Flogic, Non-Mon, modalities)
  - **Probabilistic/fuzzy**
- **Degree of formality varies widely**
  - Increased formality makes languages more amenable to machine processing (e.g., automated reasoning)
Many languages use “object oriented” model based on:

- **Objects/Instances/Individuals**
  - Elements of the domain of discourse
  - Equivalent to constants in FOL

- **Types/Classes/Concepts**
  - Sets of objects sharing certain characteristics
  - Equivalent to unary predicates in FOL

- **Relations/Properties/Roles**
  - Sets of pairs (tuples) of objects
  - Equivalent to binary predicates in FOL

- **Such languages are/can be:**
  - Well understood
  - Formally specified
  - (Relatively) easy to use
  - Amenable to machine processing
Description logics

- **First-order logics**
  - Constants, variables (e.g., x, y)
  - Predicates (e.g., p, q, r)
  - Operators (e.g., not, and, or, imply)
  - Quantifiers (e.g., for all, exists)

- **Description logics (DL)** are subsets of first-order logic with equality
  - Primitive concepts (i.e., unary predicates)
  - Roles (i.e., binary relations)

- DL were designed to model rich class hierarchies

- DL support two types of queries
  - Subsumption queries: one complex concept is always a subset of another?
  - Membership queries: whether a given individual is an instance of a given concept?
DL Semantics

- Interpretation function $I$ extends to **concept expressions** in an obvious(ish) way, i.e.:

\[
\begin{align*}
(C \cap D)^I &= C^I \cap D^I \\
(C \cup D)^I &= C^I \cup D^I \\
(\neg C)^I &= \Delta^I \setminus C^I \\
\{x\}^I &= \{x^I\} \\
(\exists R.C)^I &= \{x \mid \exists y. \langle x, y \rangle \in R^I \land y \in C^I\} \\
(\forall R.C)^I &= \{x \mid \forall y. \langle x, y \rangle \in R^I \Rightarrow y \in C^I\} \\
(\leq nR)^I &= \{x \mid \#\{y \mid \langle x, y \rangle \in R^I\} \leq n\} \\
(\geq nR)^I &= \{x \mid \#\{y \mid \langle x, y \rangle \in R^I\} \geq n\}
\end{align*}
\]
Tutorial on XML

Contents

- XML
- DOM
- DTD
- XSL / XSLT

- XML Schema
- RDF
- Logics / Description logics
- Ontology
- OWL
Ontology: Origins and History

Ontology in Philosophy

a philosophical discipline—a branch of philosophy that
deals with the nature and the organisation of reality

• Science of Being (Aristotle, Metaphysics, IV, 1)

• Tries to answer the questions:

  What characterizes being?

  Eventually, what is being?
Ontology in Computer Science

• An ontology is an engineering artifact:
  – It is constituted by a specific vocabulary used to describe a certain reality, plus
  – a set of explicit assumptions regarding the intended meaning of the vocabulary.

• Thus, an ontology describes a formal specification of a certain domain:
  – Shared understanding of a domain of interest
  – Formal and machine manipulable model of a domain of interest

“An explicit specification of a conceptualisation” [Gruber93]
Structure of an Ontology

Ontologies typically have two distinct components:

• Names for important concepts in the domain
  – Elephant is a concept whose members are a kind of animal
  – Herbivore is a concept whose members are exactly those animals who eat only plants or parts of plants
  – Adult_Elephant is a concept whose members are exactly those elephants whose age is greater than 20 years

• Background knowledge/constraints on the domain
  – Adult_Elephants weigh at least 2,000 kg
  – All Elephants are either African_Elephants or Indian_Elephants
  – No individual can be both a Herbivore and a Carnivore
Tutorial on XML

Contents

- XML
- DOM
- DTD
- XSL / XSLT

- XML Schema
- RDF
- Logics / Description logics
- Ontology
- OWL
Problems with RDFS

- **RDFS too weak to describe resources in sufficient detail**
  - No localised range and domain constraints
    - Can’t say that the range of hasChild is person when applied to persons and elephant when applied to elephants
  - No existence/cardinality constraints
    - Can’t say that all instances of person have a mother that is also a person, or that persons have exactly 2 parents
  - No transitive, inverse or symmetrical properties
    - Can’t say that isPartOf is a transitive property, that hasPart is the inverse of isPartOf or that touches is symmetrical
  - ...
- **Difficult to provide reasoning support**
  - No “native” reasoners for non-standard semantics
  - May be possible to reason via FO axiomatisation
From RDF to OWL

- Two languages developed to satisfy above requirements
  - **OIL**: developed by group of (largely) European researchers (several from EU OntoKnowledge project)
  - **DAML-ONT**: developed by group of (largely) US researchers (in DARPA DAML programme)
- Efforts merged to produce **DAML+OIL**
  - Development was carried out by “Joint EU/US Committee on Agent Markup Languages”
  - Extends (“DL subset” of) RDF
- DAML+OIL submitted to W3C as basis for standardisation
  - Web-Ontology (**WebOnt**) Working Group formed
  - WebOnt group developed **OWL** language based on DAML+OIL
  - 10 Feb 2004: OWL is a W3C Recommendation
**OWL example**

E.g., Person $\sqsubseteq \forall hasChild.Doctor \sqcup \exists hasChild.Doctor$:

```xml
<owl:Class>
  <owl:intersectionOf rdf:parseType="collection">
    <owl:Class rdf:about="#Person"/>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasChild"/>
      <owl:toClass>
        <owl:unionOf rdf:parseType="collection">
          <owl:Class rdf:about="#Doctor"/>
          <owl:Restriction>
            <owl:onProperty rdf:resource="#hasChild"/>
            <owl:hasClass rdf:resource="#Doctor"/>
          </owl:Restriction>
        </owl:unionOf>
      </owl:toClass>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```
## OWL Class Constructors

<table>
<thead>
<tr>
<th>Constructor</th>
<th>DL Syntax</th>
<th>Example</th>
<th>Modal Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>intersectionOf</td>
<td>$C_1 \sqcap \ldots \sqcap C_n$</td>
<td>Human $\sqcap$ Male</td>
<td>$C_1 \land \ldots \land C_n$</td>
</tr>
<tr>
<td>unionOf</td>
<td>$C_1 \sqcup \ldots \sqcup C_n$</td>
<td>Doctor $\sqcup$ Lawyer</td>
<td>$C_1 \lor \ldots \lor C_n$</td>
</tr>
<tr>
<td>complementOf</td>
<td>$\sim C$</td>
<td>$\sim$ Male</td>
<td>$\sim C$</td>
</tr>
<tr>
<td>oneOf</td>
<td>${x_1} \sqcup \ldots \sqcup {x_n}$</td>
<td>${john} \sqcup {mary}$</td>
<td>$x_1 \lor \ldots \lor x_n$</td>
</tr>
<tr>
<td>allValuesFrom</td>
<td>$\forall P.C$</td>
<td>$\forall$ hasChild.Doctor</td>
<td>$[P]C$</td>
</tr>
<tr>
<td>someValuesFrom</td>
<td>$\exists P.C$</td>
<td>$\exists$ hasChild.Lawyer</td>
<td>$\langle P \rangle C$</td>
</tr>
<tr>
<td>maxCardinality</td>
<td>$\leq nP$</td>
<td>$\leq$ hasChild</td>
<td>$[P]_{n+1}$</td>
</tr>
<tr>
<td>minCardinality</td>
<td>$\geq nP$</td>
<td>$\geq$ hasChild</td>
<td>$\langle P \rangle_n$</td>
</tr>
</tbody>
</table>

- **XMLS datatypes** as well as classes in $\forall P.C$ and $\exists P.C$
  - E.g., $\exists$ hasAge.nonNegativeInteger
- Arbitrarily complex **nesting** of constructors
  - E.g., Person $\sqcap \forall$ hasChild.Doctor $\sqcup \exists$ hasChild.Doctor
### OWL Axioms

<table>
<thead>
<tr>
<th>Axiom</th>
<th>DL Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>subClassOf</td>
<td>$C_1 \sqsubseteq C_2$</td>
<td>Human $\sqsubseteq$ Animal $\cap$ Biped</td>
</tr>
<tr>
<td>equivalentClass</td>
<td>$C_1 \equiv C_2$</td>
<td>Man $\equiv$ Human $\cap$ Male</td>
</tr>
<tr>
<td>disjointWith</td>
<td>$C_1 \sqsubseteq \neg C_2$</td>
<td>Male $\sqsubseteq \neg$ Female</td>
</tr>
<tr>
<td>sameIndividualAs</td>
<td>${x_1} \equiv {x_2}$</td>
<td>${\text{President Bush}} \equiv {\text{G.W. Bush}}$</td>
</tr>
<tr>
<td>differentFrom</td>
<td>${x_1} \sqsubseteq \neg {x_2}$</td>
<td>${\text{john}} \sqsubseteq \neg {\text{peter}}$</td>
</tr>
<tr>
<td>subPropertyOf</td>
<td>$P_1 \sqsubseteq P_2$</td>
<td>hasDaughter $\sqsubseteq$ hasChild</td>
</tr>
<tr>
<td>equivalentProperty</td>
<td>$P_1 \equiv P_2$</td>
<td>cost $\equiv$ price</td>
</tr>
<tr>
<td>inverseOf</td>
<td>$P_1 \equiv P_2^-$</td>
<td>hasChild $\equiv$ hasParent$^-$</td>
</tr>
<tr>
<td>transitiveProperty</td>
<td>$P^+ \sqsubseteq P$</td>
<td>ancestor$^+$ $\sqsubseteq$ ancestor</td>
</tr>
<tr>
<td>functionalProperty</td>
<td>$\top \sqsubseteq \leq 1P$</td>
<td>$\top \sqsubseteq \leq 1\text{hasMother}$</td>
</tr>
<tr>
<td>inverseFunctionalProperty</td>
<td>$\top \sqsubseteq \leq 1P^-$</td>
<td>$\top \sqsubseteq \leq 1\text{hasSSN}$</td>
</tr>
</tbody>
</table>

- **Axioms (mostly) reducible to inclusion ($\sqsubseteq$)**

  $F \equiv G \iff \text{both } F \sqsubseteq G \text{ and } G \sqsubseteq F$
Why Separate Classes and Datatypes?

• **Philosophical reasons:**
  – Datatypes structured by *built-in predicates*
  – Not appropriate to form new datatypes using ontology language

• **Practical reasons:**
  – Ontology language remains *simple and compact*
  – **Semantic integrity** of ontology language not compromised
  – **Implementability** not compromised — can use hybrid reasoner
    • Only need sound and complete decision procedure for:
      \[ g^{T_1} \cap \ldots \cap g^{T_q} \]
      where \( g \) is a (possibly negated) datatype
Limitations of DL (and thus OWL)

- They express very little about the overlap between two classes
  - Given two classes, you can infer that one subsumes the other, that they are disjoint, or that they may have a non-empty overlap
  - However, the degree of the overlap cannot be described or inferred
  - This limitation causes a problem when you consider the performance of inference/query answering, or when the application domain requires
References

• About standards/tools:
  – http://www.w3.org/2001/sw/WebOnt/
  – http://www.w3schools.com/

• Discussion about Semantic Web
  – Michael Uschold, Where are the semantics in the Semantic Web?, *AI Magazine*, Fall 2003, pp 25—36

• Ontology