Knowledge Engineering

Semester 2, 2004-05

Michael Rovatsos
mrovatso@inf.ed.ac.uk

Lecture 15/16 – Knowledge Engineering and the Semantic Web
4th March/8th March 2005

Where are we?

Last time . . .

- Distributed rational decision making
- Automated negotiation and mechanism design
- Electronic Auctions as an example

Today . . .

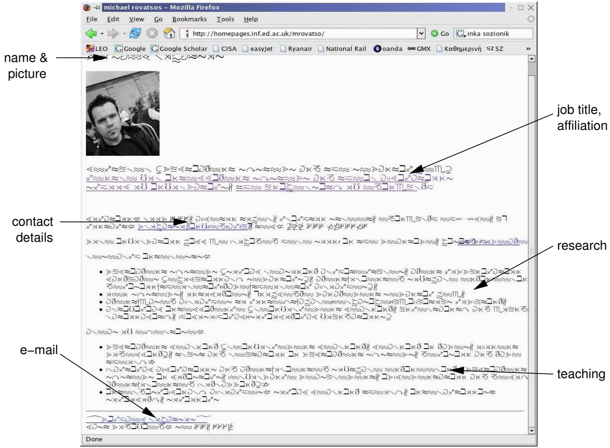
- Knowledge Engineering & The Semantic Web

The Web

- The current Web landscape: A collection of files/documents
  - mostly text, some multimedia, some databases, some (simple) services
- HTML: Modest compliance with standards (thanks to robustness of browsers)
- Hyperlinks: Annotated with text, sometimes barely understandable even for humans
- Capabilities:
  - Simple information retrieval (scalability?)
  - Fairly simple transactions/services (play chess, buy a book)
- All the relevant data is (or will soon be) on the Web, but in a form suitable for human processing only (it seems)

The Problem

This is what my homepage looks like to a machine:



Example

- We would like the Web to be used for automating more complex tasks:

  > *Why can't my online calendar and bank account negotiate with my garage's to arrange a mutually convenient time and price to repair my leaking tyre?*

- How can my agent find/parse/extract garage's free times?
- Which of my appointments are critical/flexible? Even if I annotated entries, what if the garage's timetable doesn't have such a concept?
- Lots of constraints:
  - How long will it take to get to the garage?
  - Would I pay extra if they come to collect the car?
  - Can they repair the door lock too?

The Vision

- Tim Berners-Lee: "I have a dream for the Web ... and it has two parts"
- The first Web enables communication between people
- The new Web will bring computers into the action

  Step 1 – Describe: putting data on the Web in machine-understandable form – a Semantic Web

    - RDF (based on XML)
    - Master list of terms used in a document (RDF Schema)
    - Each document mixes global standards and local agreed-upon terms (namespaces)

  Step 2 – Infer and reason: apply logic inference

    - Operate on partial understanding
    - Answering why

The Semantic Web

- What is the Semantic Web?

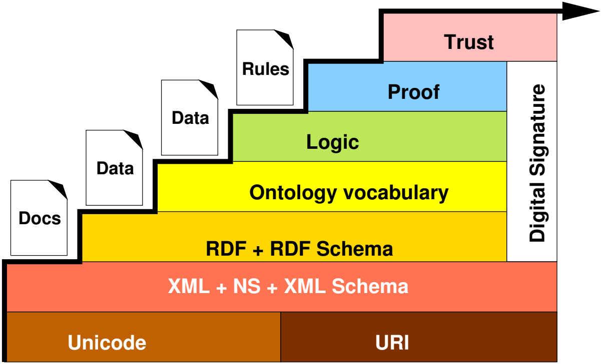
  The idea of representing Web content in a form that is more easily machine-processable and to use intelligent techniques to take advantage of these representations

- Semantic Web technologies:
  - Explicit meta-data: try to capture the meaning of data by annotating it with information about the content
  - Ontologies: facilitate organisation/navigation & search, bridge gaps between terminologies
  - Logic: reasoning about the meta-data using ontological knowledge
  - Agents: the programs that are going to use all this

The Semantic Web

- Ontologies & Logic & Agents: The reason why the SW is relevant from a KE perspective (esp. knowledge synthesis) perspective
- The SW is a lot about standards, languages, notation, etc.
  ➡ We will focus mostly on aspects relevant from a KE perspective
- Example application areas
  - Personal agents
  - Business-to-business eCommerce
  - Knowledge management

Semantic Web Technologies: The Layer Cake



Layered approach: downward compatibility + upward partial understanding

Semantic Web Languages

- Unicode: basic character encoding system, platform-independent & suitable for any language (better than ASCII)
- Uniform Resource Identifiers (URIs): "points" in information space, essentially strings that identify resources (usually URLs, but can be any unique identifier)
- XML: surface syntax for structured documents (no semantic constraints)
- XML Schema: describes the structure of XML documents

Semantic Web Languages

- RDF: data model for objects and relationships, basically statements of the form $\langle \mathit{object}, \mathit{attribute}, \mathit{value} \rangle$ (usually represented in an XML syntax)
- RDF Schema: vocabulary description language for describing properties and classes of RDF resources with semantics of generalisation hierarchy
- OWL: richer vocabulary description language
  - description of relations between classes (e.g. disjointness)
  - cardinality restrictions
  - typing of properties
  - characteristics of properties
  - enumerated classes

  (builds on theory of description logics) s

XML

- HTML: A language for markup of Web pages (tags, attributes, links)

```
<h2>Knowledge Engineering</h2>
<h2>About </h2>
...
The module descriptor can be found
   <a href="http://foo.ed.ac.uk">here</a>.<br>
```

- XML: A metalanguage for markup, users can define their own tags

```
<?xml version="1.0" encoding="UTF-16"?>
<!DOCTYPE stafflisting SYSTEM "staff.dtd">
<lecturer>
   <name>Michael Rovatsos</name>
   <contact phone="+44 131 651 3263"
            email="mrovatso@inf.ed.ac.uk" />
</lecturer>
```

XML

- An XML document consists of
  - Prolog (XML declaration, (optionally) references e.g. to DTDs)
  - A single root element
  - Elements (ordered, often nested) with attributes (unique names, unordered, not nested!)
  - Processing instructions (e.g. CSS)
- Underlying tree data model
  - Unique root node
  - Nodes are labelled with element names
  - No cycles
- XML by itself is just hierarchically structured text

DTDs/XML Schema

- How can we specify the structure of a class of XML documents (say, for two communicating applications)?
- Two methods:
  - Document Type Definitions (DTD, old, restricted)
  - XML Schema (newer, more expressive, uses XML itself)
- DTDs define elements, their nesting structure, attributes and values
- XML Schema Definitions (XSD) are written in XML themselves and
  - allow use of more built-in and user-defined data types, enumerations
  - allow for reusing (extending/restricting) existing definitions

Example: DTD

```
<?xml version="1.0" encoding="UTF-16"?>
<order orderNo="23456"  agent="Mary Moore" customer="John Smith"
                date="October 15, 2002">
                <descr>John Smith's order handled by
                    Mary Moore</descr>
                <item itemNo="a528" quantity="1"/>
                <item itemNo="c817" quantity="3"/> </order>

<!ELEMENT order (descr|item+)>
<!ATTLIST order orderNo  ID     #REQUIRED
                agent    IDREF  #REQUIRED
                customer CDATA  #REQUIRED
                date     CDATA  #REQUIRED>
<!ELEMENT item EMPTY>
<!ATTLIST item  itemNo   ID     #REQUIRED
                quantity CDATA  #REQUIRED
                comments CDATA  #IMPLIED>
<!ELEMENT descr (#PCDATA)>
```

Example: XML Schema

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
   version="1.0">
  <xsd:element name="Bookstore">
    <xsd:complexType>
       <xsd:sequence>
         <xsd:element ref="Book" minOccurs="1" maxOccurs="unbounded"/>
       </xsd:sequence>
    </xsd:complexType> </xsd:element>
  <xsd:element name="Book">
    <xsd:complexType>
       <xsd:sequence>
         <xsd:element ref="Title" minOccurs="1" maxOccurs="1"/>
         <xsd:element ref="Author" minOccurs="1" maxOccurs="unbounded"/
       </xsd:sequence>
    </xsd:complexType> </xsd:element>
  <xsd:element name="Title" type="xsd:string"/>
  <xsd:element name="Author" type="xsd:string"/> </xsd:schema>
```

Example: XML Schema

An Example document for this schema definition:

```
<?xml version="1.0" encoding="UTF-16"?>
<Bookstore>
  <Book>
    <Title>How we did it</Title>
      <Author>Adam Ant</Author>
      <Author>Brigitte Bardot</Author>
      <Author>Chevy Chase</Author>
  </Book>
  <Book>
    <Title>How I failed</Title>
      <Author>Danny DeVito</Author>
  </Book>
</Bookstore>
```

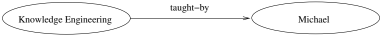
XML: Further Issues

- Becoming the de facto standard for document exchange
- Good tool support (parsers, validity checking, etc.)
- Can be used for transformation of structured information (XSLT)
- Supported by query languages
- But is this really part of the Semantic Web?
  ➡ No, just a basic technology the SW uses

RDF/RDF Schema

- XML allows for structuring documents, but not for specifying their semantics
  (many different ways of describing same information)
- Resource Description Framework (RDF): a data model (actually not a language) for describing Web resources via meta-data
- Uses an XML-based syntax, but this is not a necessary requirement of the RDF model
- RDF Schema (RDFS): language used to describe terminology used in RDF statements
- Unfortunate use of term "schema": XML Schema describes *structure* of XML documents, RDFS describes *vocabulary*
- RDF+RDFS provide a simple, lightweight ontology system

RDF Basics

- **Resources**: anything that can be associated with a URI (URI does not ensure access)
- **Properties**: a special kind of resource describing a relation between resources (e.g. "written by", "age of" etc.) also identified by URI
- **Statements**: object-attribute-value triples
  - Object=resource, attribute=property, value=resource or literal (atomic string value)
- A triple (x, P, y) can be interpreted as a predicate $P(x, y)$
  - Graphical model: 

  - An RDF document is equivalent to a graph consisting of such nodes and edges
- Essentially a semantic network (suffers from same problem of restriction to binary predicates)

Example: RDF

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:xsd="http://www.w3.org/2001/XLMSchema#"
        xmlns:uni="http://www.mydomain.org/uni-ns">
<rdf:Description rdf:about="949318">
  <uni:name>David Billington</uni:name>
  <uni:title>Associate Professor</uni:title>
  <uni:age rdf:datatype="&xsd:integer">27<uni:age>
  <uni:coursesTaught>
    <rdf:Bag>
       <rdf:_1 rdf:resource="#CIT1112"/>
       ...
    </rdf:Bag>
  </uni:coursesTaught>
</rdf:Description>
<rdf:Description rdf:about="CIT1111">
 <rdf:type rdf:resource="http://www.mydomain.org/uni-ns#course"/>
 <uni:courseName>Discrete Maths</uni:courseName>
 <uni:isTaughtBy rdf:resource="949318" /> </rdf:Description> </rdf:RDF>
```

RDF

- Use of rdf:About/rdf:ID to distinguish between descriptions of some object defined elsewhere or the definition itself
- But formally no notion of “defining” location
- Scope of descriptions provided within other descriptions is global
- Container elements: ordered (`rdf:Seq`), unordered (`rdf:Bag`), set of alternatives (`rdf:Alt`)
- Containers can't be closed ➡ use of collections (`rdf:List`, `rdf:First`, `rdf:Rest`)
- Reification: describe RDF statements themselves

```
<rdf:Statement rdf:ID="StatementAbout949318">
  <rdf:subject rdf:resource="#949318"/>
  <rdf:predicate
       rdf:resource="http://www.mydomain.org/uni-ns#name"/>
  <rdf:object>David Billington</rdf:object>
</rdf:Statement>
```

RDF Schema

- RDF allows users to define own vocabularies to describe resources ➡ RDF Schema can be used to supply the semantics for these
- Main features:
  - Classes (connection to RDF instances via `rdf:type`)
  - Properties (can be restricted in domain and range via classes)
  - Class/property hierarchies and inheritance (RDFS fixes semantics of subclass property)
- Important difference to attributes in OOP: properties are defined globally rather than as an attribute of a class

RDF Schema

- Core classes
  - `rdfs:Resource`, the class of all resources
  - `rdfs:Class`, the class of all classes
  - `rdfs:Literal`, the class of all literals (strings)
  - `rdf:Property`, the class of all properties.
  - `rdf:Statement`, the class of all reified statements
- Core properties
  - `rdf:type`, which relates a resource to its class
  - `rdfs:subClassOf`, which relates a class to one of its superclasses
  - `rdfs:subPropertyOf`, relates a property to one of its superproperties
  - `rdfs:domain`, which specifies the domain of a property
  - `rdfs:range`, which specifies the range of a property
- Utility properties: `rdfs:seeAlso`, `rdfs:isDefinedBy`, `rdfs:comment` `rdfs:label`

Example: RDF Schema

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
 <rdfs:Class rdf:about="#staffMember">
  <rdfs:comment> The class of staff members. </rdfs:comment>
 </rdfs:Class>

 <rdfs:Class rdf:about="#lecturer">
  <rdfs:comment> All lecturers are staff members. </rdfs:comment>
  <rdfs:subClassOf rdf:resource="#staffMember"/>
 </rdfs:Class>

 <rdfs:Class rdf:ID="course">
        <rdfs:comment>The class of courses</rdfs:comment>
 </rdfs:Class>
...
```

Example: RDF Schema (contd)

```
...
 <rdf:Property rdf:ID="phone">
  <rdfs:domain rdf:resource="#staffMember"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Litera
 </rdf:Property>

 <rdf:Property rdf:ID="involves">
  <rdfs:domain rdf:resource="#course"/>
  <rdfs:range rdf:resource="#lecturer"/>
 </rdf:Property>

 <rdf:Property rdf:ID="isTaughtBy">
  <rdfs:comment>
    Inherits domain and range ("lecturer") from superproperty
  </rdfs:comment>
  <rdfs:subPropertyOf rdf:resource="#involves"/>
 </rdf:Property>
</rdf:RDF>
```

Semantics: Some Considerations

- If we describe, e.g. `subClassOf` in RDF as follows

```
<rdf:Property rdf:ID="subClassOf">
  <rdfs:domain rdf:resource="#Class"/>
  <rdfs:range rdf:resource="#Class"/>
</rdf:Property>
```

  we don't really have a definition for its semantics
- We have to provide a semantics for RDF/RDFS *outside* RDF/RDFS
- One method: Axiomatic Semantics (with first-order logic)
  - Write $Prop(p, r, v)$ for each RDF triple $\langle p, r, v \rangle$
  - Shorthand $Type(r, t) :\Leftrightarrow Prop(type, r, t)$
  - Axioms include (among others):
    - $Type(p, Property) \Rightarrow Type(p, Resource)$, $Type(c, Class) \Rightarrow Type(c, Resource)$
    - $Prop(p, r, v) \Rightarrow Type(p, Property)$
    - $Prop(subClassOf, c, c') \Rightarrow (Type(c, Class) \wedge Type(c', Class) \wedge \forall x(Type(x, c) \Rightarrow Type(x, c')))$

Semantics: Some Considerations

- Axiomatic semantics requires a FOL proof system
- Alternatively, use direct inference system in RDF/RDFS with the following kind of inference rules:
  - IF *E* contains certain triples THEN add to *E* certain additional triples
- Examples:

```
IF E contains the triples (u,rdfs:subClassOf,v) and
         (v,rdfs:subclassOf,w)
THEN E also contains the triple (u,rdfs:subClassOf,w)

IF E contains the triples (x,rdf:type,u) and
         (u,rdfs:subClassOf,v)
THEN E also contains the triple (x,rdf:type,v)
```

- Much simpler, but "closure" of triple store increases its size (wasteful and not really elegant)

OWL

- RDF (roughly) limited to binary ground predicates, RDFS (roughly) to subclass/property hierarchies and domain/range definitions for these
- Features missing in RDF(S):
  - Local scope of properties (once domain/range are defined, they hold true of all classes) ➡ new property required for each class with different range restrcitions
  - Disjointness of classes (only subclass relationship)
  - Boolean combinations of classes (using union, intersection, complement)
  - Cardinality restrictions (e.g. "a person has exactly two parents")
  - Special characteristics of properties (transitivity, uniqueness, inverse properties)
- Unfortunately, expressiveness of RDFS would lead to uncontrollable computational properties

OWL

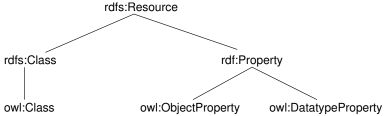
- Requirements for an ontology language:
  - well-defined syntax, formal semantics, sufficient expressive power, efficient reasoning support, convenience of expression
- Ontological reasoning should cover tests for:
  - Class membership ($x \in C \wedge C \subseteq D \Rightarrow x \in D$)
  - Equivalence of classes ($A = B \wedge B = C \Rightarrow A = C$)
  - Consistency (e.g. $A \subseteq B \cap C \wedge A \subseteq D \wedge B \cap D = \emptyset$ ➡ contradiction)
  - Classification (x satisfies certain conditions ➡ infer $x \in A$ for some class A)

OWL Flavours

- Different sub-languages to fulfill different requirements
- OWL Full: all modelling primitives can be used, fully downward compatible with RDF ➡ undecidable
- OWL DL: application of OWL's constructors to each other disallowed, maps to well-studied description logic ➡ lose compatibility with RDF but efficient reasoning support
- OWL Lite: no enumerated classes, disjointness statements, arbitrary cardinalities ➡ easier to understand and to implement tools for
- OWL Full ➡ OWL DL ➡ OWL Lite: Compatibility wrt documents and conclusions
- OWL builds on RDF and uses RDF's XML-based syntax

OWL

- OWL uses RDF's XML based syntax (though other syntactic forms have been proposed)
- Instances declared using RDF, OWL constructors are specialisations of their RDF counterparts:



- Object properties related objects to objects, data type properties relate objects to datatype values
- `owl:Thing`/`owl:Nothing` used to denote most general/empty class

Example: OWL

```
<rdf:RDF xmlns:owl ="http://www.w3.org/2002/07/owl#"
  xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:xsd ="http://www.w3.org/2001/XLMSchema#">
<owl:Ontology rdf:about="">
  <rdfs:comment>An example OWL ontology </rdfs:comment>
  <owl:priorVersion rdf:resource="http://www.mydomain.org/uni-ns-old"/>
  <owl:imports rdf:resource="http://www.mydomain.org/persons"/>
  <rdfs:label>University Ontology</rdfs:label> </owl:Ontology>

<owl:Class rdf:about="#associateProfessor">
  <owl:disjointWith rdf:resource="#professor"/>
  <owl:disjointWith rdf:resource="#assistantProfessor"/>
</owl:Class>

<owl:Class rdf:ID="faculty">
  <owl:equivalentClass rdf:resource="#academicStaffMember"/>
</owl:Class>
```

Example: OWL (contd.)

```
<owl:DatatypeProperty rdf:ID="age">
  <rdfs:range rdf:resource="http://www.w3.org/2001/
              XLMSchema#nonNegativeInteger"/> </owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="isTaughtBy">
  <owl:domain rdf:resource="#course"/>
  <owl:range rdf:resource="#academicStaffMember"/>
  <rdfs:subPropertyOf rdf:resource="#involves"/>
  <owl:inverseOf rdf:resource="#teaches"/>
</owl:ObjectProperty>

<owl:Class rdf:about="#firstYearCourse">
  <rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#isTaughtBy"/>
     <owl:allValuesFrom rdf:resource="#Professor"/>
    </owl:Restriction> </rdfs:subClassOf> </owl:Class> </rdf:RDF>
```

OWL: Further features

- Property restrictions:
  - Requiring specific values (`owl:hasValue`), existential/universal quantification (`owl:someValuesFrom`/`owl:allValuesFrom`)
  - Cardinalities (`owl:minCardinality`, `owl:maxCardinality`)
  - Special properties:
    - `owl:TransitiveProperty`, `owl:SymmetricProperty`, `owl:FunctionalProperty`, `owl:InverseFunctionalProperty`
- Boolean combinations:
  - `owl:complementOf` (has to use `owl:subClassOf`), `owl:unionOf`, `owl:intersectionOf`
- Enumerations: define class through its elements (`owl:oneOf`) (each instance can be a `owl:Thing`)
- No unique-names assumption:
  - for example, if two objects have different names and a property requires uniqueness, OWL reasoner would infer equality

OWL: Shortcomings/possible extensions

- Modules and imports
  - import functionality only allows for importing entire ontologies, not just parts of them
- No default reasoning mechanism
  - No consensus regarding use of defaults for non-monotonic reasoning (not even overriding mechanism of semantic networks used)
- Open-world assumption
  - reasonable on the WWW in a sense, but sometimes closed-world assumption is useful
- No unique names assumption
  - different names don't imply different objects; again, useful for the Web, but sometimes not
- No procedural attachments
- No property chaining
  - no way to define properties as general logical rules

Critique

- Is it going to work?
  - How likely is provision of sufficient meta-data?
    How will it be maintained?
  - How will different ontologies be aligned with each other?
  - Will agents be sufficiently intelligent?
  - Will they be sufficiently trustworthy?
  - Poor understanding of "open knowledge"
- From the KE perspective, what do SW technologies "buy us"?
  - Standardisation, tool support, etc.
  - From the point of view of reasoning nothing new so far
  - Yet challenging issues, e.g. ontology mapping, trust, etc.
  - A good application domain with interesting theoretical problems

Summary

- Introduction to the Semantic Web
- Key technologies and languages
- Structuring documents: XML/XML Schema
- Describing resources and class/property hierarchies: RDF/RDF Schema
- Describing ontologies: OWL
- Next time: **Knowledge evolution**