RDF Statements

- In technical terms, we have a **Resource**, a **Property**, and a **Property value** corresponding to subject, predicate and object of a Statement.
- Example:

  The artist of Empire Burlesque is Bob Dylan

- Resource: Empire Burlesque
- Property: artist
- Property value: Bob Dylan

Resources

- We can think of a resource as an object, a "thing" we want to talk about
  - E.g. authors, books, publishers, places, people, hotels
- We need to be able to uniquely identify them.
- Every resource has a **URI**, a Universal Resource Identifier
- A URI can be
  - a URL (Web address) or
  - some other kind of unique identifier

Properties

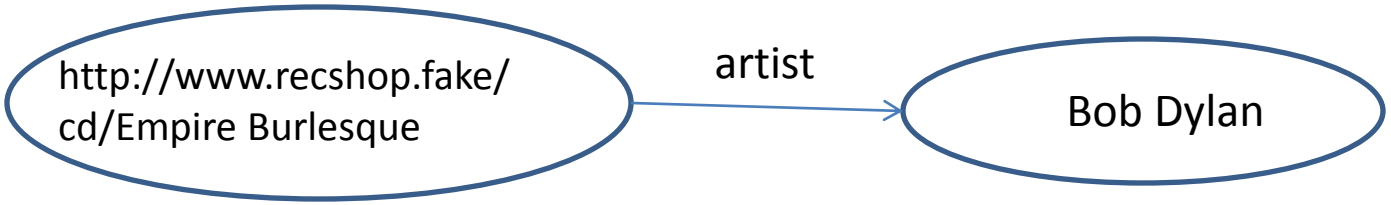
- Properties are a special kind of resources
- They describe relations between resources
  - E.g. “written by”, “age”, “title”, etc.
- Properties are also identified by URIs
- Advantages of using URIs:
  - A global, worldwide, unique naming scheme
  - Reduces the homonym problem of distributed data representation

RDF Example

- XML based

```
<?xml version="1.0"?>
<rdf:RDF >
<rdf:Description rdf:about="http://www.recshop.fake/cd/Empire Burlesque">
    <cd:artist>Bob Dylan</cd:artist>
    <cd:country>USA</cd:country>
    …
</rdf:RDF>
```

Semantic Net



- A directed graph with labeled nodes and arcs
  - **from** the resource (the **subject** of the statement)
  - **to** the value (the **object** of the statement)
- Known in AI as a *semantic net*
- Ideally, the value of a statement is a resource also
  - It may be linked to other resources

Example Evaluated

- How to avoid the Tower of Babel?
  - Ideally, all web-services use the same namespace
  - Agree on the same syntax for XML
  - Develop a single standard
  - Soon afterwards, MicroSoft develops its proprietary standard.
  - Still, a very small set of standards
- Alternatively, facilitate matching of namespace elements.

Matching Namespaces: RDFS

- RDFS stands for RDF Schema
- It enables the definition of classes and class hierarchies
- Additionally, we can define property hierarchies

RDFS Example

```
<rdfs:Class rdf:ID=“recordingMedium" />

<rdfs:Class rdf:ID=“cd">
 <rdfs:subClassOf rdf:resource="#recordingMedium"/>
</rdfs:Class>

<rdfs:Property rdf:ID=“artist”/>

<rdfs:Property rdf:ID=“starvingArtist">
 <rdfs:subPropertyOf rdf:resource="#artist"/>
</rdfs:Class>
```

OWL

- More sophisticated relationships
- Disjointness of classes
  - Sometimes we wish to say that classes are disjoint (e.g. **male** and **female**)
- Boolean combinations of classes
  - Sometimes we wish to build new classes by combining other classes using union, intersection, and complement
  - E.g. **person** is the disjoint union of the classes **male** and **female**

OWL

- Cardinality restrictions
  - E.g. a person has exactly two parents, a course is taught by at least one lecturer
- Special characteristics of properties
  - Transitive property (like “greater than”)
  - Unique property (like “is mother of”)
  - A property is the inverse of another property (like “eats” and “is eaten by”)

Owl Example

```
<owl:Ontology rdf:about="xml:base"/>

<owl:Class rdf:ID="animal">
    <rdfs:comment>Animals form a class</rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="plant">
    <rdfs:comment>Plants are disjoint from animals. </rdfs:comment>
    <owl:disjointWith rdf:resource="#animal"/>
</owl:Class>

<owl:Class rdf:ID="tree">
    <rdfs:comment>Trees are a type of plant. </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#plant"/>
</owl:Class>
```

Owl Example

```
<owl:Class rdf:ID="branch">
        <rdfs:comment>Branches are parts of trees. </rdfs:comment>
        <rdfs:subClassOf>
                <owl:Restriction>
                        <owl:onProperty rdf:resource="#is-part-of"/>
                        <owl:allValuesFrom rdf:resource="#tree"/>
                </owl:Restriction>
        </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="leaf">
        <rdfs:comment>Leaves are parts of branches. </rdfs:comment>
        <rdfs:subClassOf>
                <owl:Restriction>
                        <owl:onProperty rdf:resource="#is-part-of"/>
                        <owl:allValuesFrom rdf:resource="#branch"/>
                </owl:Restriction>
        </rdfs:subClassOf>
</owl:Class>
```

Owl Example

```
<owl:Class rdf:ID="carnivore">
    <rdfs:comment>Carnivores are exactly those animals
    that eat also animals.</rdfs:comment>
    <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#animal"/>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#eats"/>
            <owl:someValuesFrom
                rdf:resource="#animal"/>
        </owl:Restriction>
    </owl:intersectionOf>
</owl:Class>
```

Owl Example

```
<owl:Class rdf:ID="lion">
    <rdfs:comment>Lions are animals that eat
    only herbivores.</rdfs:comment>
    <rdfs:subClassOf rdf:type="#carnivore"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#eats"/>
            <owl:allValuesFrom
                rdf:resource="#herbivore"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

Owl Example

<lion rdf:ID=“22789" />

<lion rdf:ID=“Simba" />

Inference

- Class membership
  - If x is an instance of a class C, and C is a subclass of D, then we can infer that x is an instance of D
- Equivalence of classes
  - If class A is equivalent to class B, and class B is equivalent to class C, then A is equivalent to C, too

Inference

- Consistency
  - X instance of classes A and B, but A and B are disjoint
  - This is an indication of an error in the ontology
- Classification
  - Certain property-value pairs are a sufficient condition for membership in a class A; if an individual x satisfies such conditions, we can conclude that x must be an instance of A

Inference

- The richer the language is, the more inefficient the reasoning support becomes
- Sometimes it crosses the border of *noncomputability*
- We need a compromise:
  - A language supported by reasonably efficient reasoners
  - A language that can express large classes of ontologies and knowledge.

Inference

- Inference within a service
- Additionally, attempt to match hierarchies from different services to determine equivalent concepts.

Current Status: Inference

- Many reasoners
- Few that work
- Jena (Java based)
- Some python libraries that would facilitate building a reasoner

Evaluation of Semantic Services

- A lot of technology
- Few semantic web applications
- Applications are typically an ontology on top of a web service
- Reasoning is expensive
- We have not covered key technology such as: service discovery, matching and execution

References

- Book: A Semantic Web Primer, by Antoniou & van Harmelen
- Website for book: http://www.ics.forth.gr/isl/swprimer/
- RDF/S Tutorial: http://www.w3schools.com/rdf/default.asp
- OWL Guide: http://www.w3.org/TR/owl-guide/
- OWL Reference: http://www.w3.org/TR/owl-ref/
- Jena: http://jena.sourceforge.net/