Introduction to
Web Ontology Language (OWL)

Owen.Conlan@scss.tcd.ie
Athanasios.Staikopoulos@scss.tcd.ie
Today’s Web

- **Web of documents** – processed by humans
- Currently, users search for data on the Web asking questions like “which documents contain these words or phrases”
Semantic Web

- **Web of things** – processed by machines
- Search in not based on word matching but on related items and relationships
Stack Architecture for Semantic Web
Semantic Web Technologies

• Set of technologies and frameworks that enable such integration (the Web of Data) possible

  • Semantic annotation and retrieval: RDF, RDFS
  • Storing the Semantic Web: Repositories
  • Querying the Semantic Web: SPARQL
  • Reasoning on the Semantic Web: OWL, reasoning tools
Representing knowledge

There are a number of options

- As **objects**, using the well-accepted techniques of object-oriented analysis and design to capture a model
- As **clauses**, going back to the early days of AI and Lisp
- As **XML**, using the industry-standard structured mark-up language
- As **graphs**, making use of the things we know about graph theory
- As some **combination** of these

We are looking for: extensibility, ease of use, ease of querying

Which would *you* choose?
We can use the **nodes** of a graph for facts and the **arcs** as (binary) relationships between them

- Arcs are typically called **predicates** or **relationships** in this view
- The set of arcs intersecting a node tells us the information we know about that fact or entity
How do we use graphs to represent knowledge?

A “key” from which to hang the different facts

A graph representation of a person's details:
- Name: Joe Bloggs
- ORI: F35
- Phone: +353 1 896 1234
- Office: Student 1

Another graph representation of a person's details:
- Name: Owen Conlan
- Office: Software Engineering
- Phone: +353 1 896 1234

A graph representation of a student's details:
- Name: Student 1
- Office: Knowledge Engineering
- Phone: +353 1 896 1234

A graph representation of a student's details:
- Name: Student 2
Graphs as knowledge – 2

Things to note

- **Scaling** – the same graph can represent a load of different knowledge simultaneously
- **Agreement** – need to know what the various predicates “mean”
- **Structure** – you need to know what nodes are related by a predicate
- **Plurality** – the same relationship may appear several times
- **Symmetry** – the same predicates can be used for common information, despite minor changes
- **Asymmetry** – relationships are inherently directed, which sometimes makes things awkward

...and this can get very tricky

...and this can be difficult to keep straight

For example both lecturers and students have names

So a knowledge (context) graph is inherently directed
Two ways to view a graph

As nodes and arcs
- Nodes store facts, edges store relationships between them

As triples
- A three-place relationship of “subject, predicate, object”
- The node and edge structure is induced by the triples – each triple defines an edge, the different subjects/objects are the population of nodes, one node per individual string
Resource Description Framework (RDF)

- RDF is a W3C recommendation that enables encoding, exchange and reuse of structured metadata
- **Resource**: anything we want to talk about
- RDF is **graphical formalism** for expressing data models about “something” using statements expressed as triples
- **RDF Triples**: a labelled connection between two resources, a labelled arc in a graph
- An RDF model is an **unordered collection of statements**, each with a subject, predicate and object
- RDF describes the semantics of information in a **machine-accessible** way
RDF Triples

- Graph representation of a triple

- This can be read as
  - $s$ has a property $p$ with a value $o$ (left to right)
  - $o$ is the value of $p$ for $s$ (right to left)
  - The $p$ of $s$ is $o$ (as directed relationship)
Example RDF Triples as Graphs

- unv:Person1234 = http://www.scss.tcd.ie/owen.conlan
- dbpedia:Dublin = http://dbpedia.org/resource/Dublin
Example RDF Triples as XML

- Triples of assertions can be expressed using XML tags

- E.g. “Cabernet Sauvignon grape”, “is a type of”, “Wine grape”

  ```xml
  <rdf:Description rdf:about="Cabernet Sauvignon grape">
    <rdf:type rdf:resource="#Wine grape" />
  </rdf:Description>
  ```

- Each resource can be assigned a different Universal Resource Identifier (URI)
  - *Thus different meanings for the same term can be assigned different URIs*

- Reference: RDF Primer. W3C draft technical note, 2002
URI (Uniform Resource Identifier)

- URI is used for identifying (naming) resources on the Web
- URLs (Uniform Resource Locators) are a particular type of URI, the resources can be accessed on the Web
- URIs unlike URLs are not limited to identifying things that have network locations
- In RDF, URIs often have fragment identifiers to point at specific parts of a document:
  - http://www.somedomain.com/some/path/to/file#fragmentID
- URIs are unambiguous, Web provides a global namespace
- Different URI schemas – http, mailto, ftp, urn …
XML to RDF

- **Modify XML to a RDF document**

XML

```xml
<?xml version="1.0"?>
<River id="Shannon"
   xmlns="http://www.scss.tcd.ie/rivers">
   <length>360 kilometers</length>
   <startingLocation>Cuilcagh Mountain, County Cavan</startingLocation>
   <endingLocation>Limerick</endingLocation>
</River>
```

RDF

```xml
<?xml version="1.0"?>
<rdf:Description rdf:about="http://www.scss.tcd.ie/rivers/Shannon"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns="http://www.scss.tcd.ie/rivers#">
   <length>360 kilometers</length>
   <startingLocation>Cuilcagh Mountain, County Cavan</startingLocation>
   <endingLocation>Limerick</endingLocation>
</rdf:Description>
```
### RDF: XML-Based Syntax Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdf:RDF</td>
<td>root element of RDF documents, where a number of descriptions are defined</td>
</tr>
<tr>
<td>rdf:Description</td>
<td>element contains the description of the resource</td>
</tr>
<tr>
<td>rdf:type</td>
<td>instance of</td>
</tr>
<tr>
<td>rdf:Bag</td>
<td>an unordered container of resources</td>
</tr>
<tr>
<td>rdf:Seq</td>
<td>an ordered container of resources</td>
</tr>
<tr>
<td>rdf:Alt</td>
<td>Defines a set of alternative resources</td>
</tr>
</tbody>
</table>
### RDF: XML-Based Syntax Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdf:ID</td>
<td>indicating a new resource</td>
</tr>
<tr>
<td>rdf:about</td>
<td>referencing an existing resource</td>
</tr>
<tr>
<td>rdf:resource</td>
<td>allows property elements to be defined as resources</td>
</tr>
</tbody>
</table>
A cluster of facts

Given a common subject we can build a cluster of facts using nested predicate elements

```xml
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:s="http://www.dsg.cs.tcd.ie/xml/demo.html#">
    <rdf:Description about="http://www.dsg.cs.tcd.ie/">
        <s:about>Distributed Systems Group</s:about>
        <s:author>S. Punter</s:author>
        <s:phone>+353 1 123 4567</s:phone>
    </rdf:Description>
</rdf:RDF>
```

As long as we agree what the predicates mean, we can use whichever we want
## RDF Classes & Properties

### Classes
- rdf:XMLLiteral
- rdf:Property
- rdf:Alt
- rdf:Bag
- rdf:Seq
- rdf:List
- rdf:nil
- rdf:Statement

### Properties
- rdf:type
- rdf:first
- rdf:rest
- rdf:value
- rdf:subject
- rdf:predicate
- rdf:object
Identify the Resource

- URI references may be either absolute or relative
- When a (relative) URI reference consists of just a fragment identifier, it refers to the document that appears
- An element `rdf:Description` has
  - `rdf:about` attribute – references an existing resource
  - `rdf:ID` attribute – indicating a new resource
    - The value of `rdf:ID` is a "relative" URI
  - Without a name creating an anonymous resource
Blank Nodes

- RDF doesn’t require every resource in a statement to be identified with a URI
- **Blank nodes** are graph nodes that represent a resource for which we would like to make assertions, but have no way to address with a proper URI
- these resources are not visible outside – they are **anonymous**
- From a logic point of view, blank nodes represent an “existential” statement
Literals

- Literals are objects that are not URIs but actual content
- There are two kinds of literals
  - **plain** (untyped) – have a lexical form and optionally a language tag
    “wellcome”@en
  - **Typed** – is formed by pairing a string with a particular datatype (e.g. from XML Schema)
    “27”^^http://www.w3.org/2001/XMLSchema#integer
- RDF has no built-in set of datatypes. RDF uses externally datatypes that are identifies by a URI

```
<rdf:Description rdf:about="http://.../isbn/51409X">
  <page_number rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">543</page_number>
  <price rdf:datatype="http://www.w3.org/2001/XMLSchema#float">6.99</price>
</rdf:Description>
```
RDF Serialization Formats

• There is a variety of data interchange formats
  • RDF/XML – the original (W3C Recommendation) and most frequently used serialization format
  • N-Triples - simple notation, easy-to-parse, line-based format that is not as compact as Turtle
  • N3 – similar to N-Triples, additional structures to reduce repetition
  • Turtle - a compact, human-friendly format.
  • RDFa - a way of annotating XHTML web pages with RDF data.
  • Json – a JSON based serialisation
  • ...
Common Vocabularies

Commonly used vocabulary namespaces in RDF

- RDF: http://www.w3.org/1999/02/22-rdf-syntax-ns#
- Dublin Core: http://purl.org/dc/elements/1.1/
- SKOS: http://www.w3.org/2004/02/skos/core#
- FOAF: http://xmlns.com/foaf/0.1/
Structuring the knowledge - Limitations

RDF provides a way of building graphs from triples, but **doesn’t constrain** the graph too much

- Nothing stops an application from giving a place a surname, for example, although this is probably nonsense

The problem is that RDF is an *untyped mechanism* for building graphs

- No knowledge of which triples are “allowed”, or what “thing” must be the subject/object of an arc

This is a problem in two distinct ways

- In *interpretation* – different people may interpret the predicates subtly differently and use them between values you can’t handle
- In *scaling* – hard for an application to get it right
RDF Schemas (RDFS)

- Officially: “RDF Vocabulary Description Language”
- RDF is domain independent – there are no assumptions about a particular domain, concepts etc.
- When compared to XML Schema, RDFS defines the vocabulary used in RDF data models, where the XML Schema constrains the structure of XML documents.
- RDFS extends RDF with “schema vocabulary”, e.g.:
  - Class, Property
  - type, subClassOf, subPropertyOf
  - range, domain
### RDFS Classes

<table>
<thead>
<tr>
<th>RDFS Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdfs:Resource</td>
<td>the class of all resources</td>
</tr>
<tr>
<td>rdfs:Class</td>
<td>the class of all classes</td>
</tr>
<tr>
<td>rdfs:Literal</td>
<td>the class of all literals (strings)</td>
</tr>
<tr>
<td>rdfs:Property</td>
<td>the class of all properties</td>
</tr>
<tr>
<td>rdfs:Datatype</td>
<td>the class of datatypes</td>
</tr>
</tbody>
</table>
### RDFS Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdfs:subClassOf</td>
<td>Relates class to one of its superclasses, declare hierarchies of classes, is transitive by definition</td>
</tr>
<tr>
<td>rdfs:subPropertyOf</td>
<td>relates a property to one of its superproperties, is transitive by definition</td>
</tr>
<tr>
<td>rdfs:domain</td>
<td>declares the class of the subject in a triple</td>
</tr>
<tr>
<td>rdfs:range</td>
<td>declares the class or datatype of the object in a triple</td>
</tr>
<tr>
<td>rdfs:comment</td>
<td>typically provides a longer text description of the resource</td>
</tr>
<tr>
<td>rdfs:label</td>
<td>associates the resource with a human-friendly name</td>
</tr>
<tr>
<td>rdfs:isDefinedBy</td>
<td>relates a resource to the place where its definition, typically an RDF schema</td>
</tr>
<tr>
<td>rdfs:seeAlso</td>
<td>relates a resource to another resource that explains it</td>
</tr>
</tbody>
</table>
These terms are the RDF Schema building blocks (constructors) used to create vocabularies:

\[
\begin{align*}
&\text{<Person, type, Class>} \\
&\text{<hasColleague, type, Property>} \\
&\text{<Professor, subClassOf, Person>} \\
&\text{<Carole, type, Professor>} \\
&\text{<hasColleague, range, Person>} \\
&\text{<hasColleague, domain, Person>}
\end{align*}
\]
RDFS Example - Graph Model

RDF/RDFS “Liberality”

No distinction between classes and instances (individuals)

<Species, type, Class>
<Lion, type, Species>
<Leo, type, Lion>

Properties can themselves have properties

<hasDaughter, subPropertyOf, hasChild>
<hasDaughter, type, familyProperty>

No distinction between language constructors and ontology vocabulary, so constructors can be applied to themselves/each other

$type, range, Class$
$<Property, type, Class>
$type, subPropertyOf, subClassOf$
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**
Web Ontology Language (OWL): Requirements

Desirable features identified for Web Ontology Language:

Extends existing Web standards
  • Such as XML, RDF, RDFS

Easy to understand and use
  • Should be based on familiar KR idioms

Formally specified - describes the meaning of knowledge precisely

Of “adequate” expressive power

Possible to provide automated reasoning support
OWL Language

Three species of OWL

- **OWL full** is union of OWL syntax and RDF
- **OWL DL** restricted to FOL fragment
- **OWL Lite** is “easier to implement” subset of OWL DL

OWL DL Benefits from many years of DL research

- Well defined **semantics**
- **Formal properties** well understood (complexity, decidability)
- Known reasoning **algorithms**
- **Implemented systems** (highly optimised)
Example of OWL Document

```xml
<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/XMLSchema#">
    <owl:Ontology rdf:about="">
        <rdfs:comment>An example OWL ontology</rdfs:comment>
        <rdfs:label>University Ontology</rdfs:label>
    </owl:Ontology>
    <owl:Class rdf:ID="academicStaffMember"></owl:Class>
    <owl:Class rdf:ID="associateProfessor">
        <rdfs:subClassOf rdf:resource="#academicStaffMember"/>
    </owl:Class>
    ...
</rdf:RDF>
```
Example: Defining terms and a subclass relationship

Define the term “Room”

```
<owl:Class rdf:ID="Room"/>
```

Define term “Restroom” and state that a Restroom is a type of Room

```
<owl:Class rdf:ID="Restroom">
  <rdfs:subClassOf rdf:resource="#Room"/>
</owl:Class>
```

Note: owl:Thing is a predefined OWL Class and is the root of all classes. Similarly, owl:Nothing is the empty class
Defining Classes

OWL provides several other mechanisms for defining classes

- **equivalentClass** allows you to state that two classes are synonymous
- **disjointWith** allows you to state that an instance of this class cannot be an instance of another
  - *E.g. Man and Woman could be stated as disjoint classes*

**Boolean combinations**

- **unionOf** allows you to specify that a class contains things that are from more than one class
  - *E.g. Restroom could be defined as a union of MensRoom and LadiesRoom*
- **intersectionOf** allows you to specify that a class contains things that are both in one and the other
- **complementOf** allows you to specify that a class contains things that are not other things
  - *E.g. Children are not SeniorCitizens*
Example: equivalentClass and unionOf

```xml
<owl:Class rdf:ID="AtomicPlaceInBuilding">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#AtomicPlace"/>
  </rdfs:subClassOf>
  <owl:equivalentClass>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Room"/>
        <owl:Class rdf:about="#Hallway"/>
        <owl:Class rdf:about="#Stairway"/>
        <owl:Class rdf:about="#OtherPlaceInBuilding"/>
      </owl:unionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```
Example disjointWith

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

In RDF Schema the rdf:Property is used both to

- Relate one Resource to another Resource
  - For example, a “accessRestrictedToGender” property can relate a Restroom to a Gender

- Relate a resource to a rdfs:Literal or datatype
  - For example, a “latitude” property relates a Room to a xsd:string type

OWL provides different statements for two cases

- **owl:ObjectProperty** is used to relate a resource to another

  ```xml
  <owl:ObjectProperty rdf:ID="accessRestrictedToGender">
    <rdfs:range rdf:resource="#Gender"/>
    <rdfs:domain rdf:resource="#Restroom"/>
  </owl:ObjectProperty>
  ```

- **owl:DatatypeProperty** is used to relate a resource to a rdfs:Literal or XML schema data type

  ```xml
  <owl:DatatypeProperty rdf:ID="latitude">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Place"/>
  </owl:DatatypeProperty>
  ```
Characterising Properties

OWL allows use of three of the RDFS statements

- `<rdfs:range>` used to indicate the possible value types for a property.
- `<rdfs:domain>` used to associate a property with a class.
- `<rdfs:subPropertyOf>` used to specialize a property.

Example

```
<owl:ObjectProperty rdf:ID="isTaughtBy">
  <rdfs:domain rdf:resource="#course"/>
  <rdfs:range rdf:resource="#academicStaffMember"/>
  <rdfs:subPropertyOf rdf:resource="#involves"/>
</owl:ObjectProperty>
```
Characterising Properties (cont)

**owl:equivalentProperty** – to define equivalence of properties

```xml
<owl:ObjectProperty rdf:ID="lecturesIn">  
    <owl:equivalentProperty rdf:resource="#teaches"/>
</owl:ObjectProperty>
```

**owl:inverseOf** – to relate inverse properties

```xml
<owl:ObjectProperty rdf:ID="teaches">  
    <rdfs:range rdf:resource="#course"/>  
    <rdfs:domain rdf:resource="#academicStaffMember"/>  
    <owl:inverseOf rdf:resource="#isTaughtBy"/>
</owl:ObjectProperty>
```
Restricting Properties

Associate the property (defined elsewhere) with the Class by using combination of `<rdfs:subClassOf> <owl:onProperty>` and defining local restrictions on that property using `<owl:Restriction>`

- `owl:allValuesFrom` - all values of the property must come from a specific class
- `owl:someValuesFrom` - at least one value of the property must come from a specific class
- `owl:hasValue` - states a specific value that the property specified
- `owl:minCardinality` - it has at least (individuals or data values)
- `owl:maxCardinality` – it has at most (individuals or data values)
- `owl:cardinality` – it has a specific number of (individuals or data values)
Example - Restricting Properties

```xml
<owl:Class rdf:ID="Restroom">
    <rdfs:subClassOf rdf:resource="#Room" />
    <rdfs:subClassOf>
        <owl:Restriction owl:cardinality="1">
            <owl:onProperty>
                <owl:ObjectProperty rdf:about="#accessRestrictedToGender" />
            </owl:onProperty>
            <owl:allValuesFrom>
                <owl:Class rdf:about="#Gender" />
            </owl:allValuesFrom>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:class>
```
Instances

- Instances of classes are declared as in RDF

```xml
<rdf:Description rdf:ID="949352">
    <rdf:type rdf:resource="#academicStaffMember"/>
</rdf:Description>

or equivalently

<academicStaffMember rdf:ID="949352"/>
```
Summary Example

<?xml version “1.0”?>
<Room rdf:ID=”LargeConferenceRoom”>
   <address rdf:resource=“G.02”/>
   <spatiallySubsumedBy rdf:resource=“O’Reilly Institute”/>
   <adjacentRoom rdf:resource=“SmallConferenceRoom”/>
   <coordinates rdf:resource=“44,55”/>
</Room>

Given the preceding definitions it can be inferred automatically:

1. The O’Reilly Institute *spatially subsumes* the Large Conf Room (since spatiallySubsumedBy is an inverse property)
2. The Small Conference Room is *adjacent to* Large Conference Room (since adjacentRoom is symmetric)
3. Only the Large Conference Room can be found at coordinates 44,55 (since coordinates is inverse functional)
4. The Large Conference Room has only one address (since address is functional)
References

[DARPA 2003]
• Tutorial by Costello and Jacobs of MITRE funded by DARPA

OWL web site with lots of information
http://www.w3c.org/2001/sw/WebOnt/

In particular the OWL Guide provides a description of OWL, with many examples:
• http://www.w3.org/TR/2004/REC-owl-guide-20040210/