



RDF Schema and OWL

RDF Vocabulary Description Language 1.0: RDF Schema Web Ontology Language

*Adapted from course notes of
Rob Brennan, TCD, Declan O'Sullivan, TCD and Myungjin Lee, LIST*

From RDF to RDFS

RDF gives us:

- Graph-based data model
- URI-based vocabulary and node identification
- Datatypes/Literals
- RDF expression of simple facts (as a graph)
- Simple entailment (graphs merging, graph equivalence)
 - `rdf:Property`
 - `rdf:type`

Adding some structure to knowledge

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

- Nothing stops an application from giving a network element 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

How do we normally add structure in modelling?

Object Orientation

- Modelling (e.g. UML)
 - *Classes, Associations, Attributes, Instances*
 - *Subclassing, Aggregation*
 - *Cardinalities, Constraints*
- Programming (e.g. Java)
 - *Classes, instances, Methods*
 - *Inheritance*
 - *Typing*

Information

- XML
 - *Elements, subelements, attributes*
- E-R Modelling
 - *Entities, Relationships, Subclassing*
 - *Cardinalities*
 - *Constraints*

Seven steps in Knowledge Modelling

1. Determine scope and domain of model
2. Consider reusing elements of existing models
3. Enumerate important terms in the model
4. Define the classes and class hierarchy
5. Define the characteristics of each class
 - Predicates
 - Properties
6. Define the constraints on characteristics
 - Value type (String, Number, Boolean, Enummerated, Instance)
7. Define the instances

Variation of Protégé 101 methodology

RDF and RDFS

RDFS extends RDF with '[schema vocabulary](#)', e.g.

- rdfs:Class
- rdfs:subClassOf
- rdfs:subPropertyOf
- rdfs:range
- rdfs:domain

And the annotation properties:

- rdfs:label
- rdfs:comment

And the utility properties:

- rdfs:seeAlso
- rdfs:isDefinedBy

rdfs:Class

rdfs:Class = the set of classes defined in RDFS

To define a class:

Assign a class identifier (URI) to be a member of the set rdfs:Class (also a URI) using the property rdf:type (also a URI) to make this assignment in a triple statement:

Sensors rdf:type rdfs:Class .

Turtle shorthand:

Sensors **a** rdfs:Class .

However could also just declare a rdfs:subClass or use a rdfs:domain or rdfs:range declaration...

Describing Classes

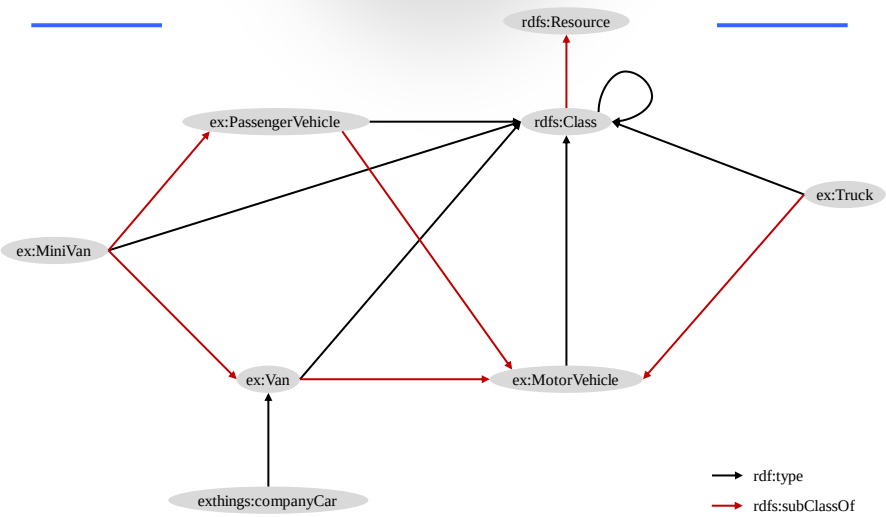
Class

- "kinds of things", the generic concept of a Type or Category

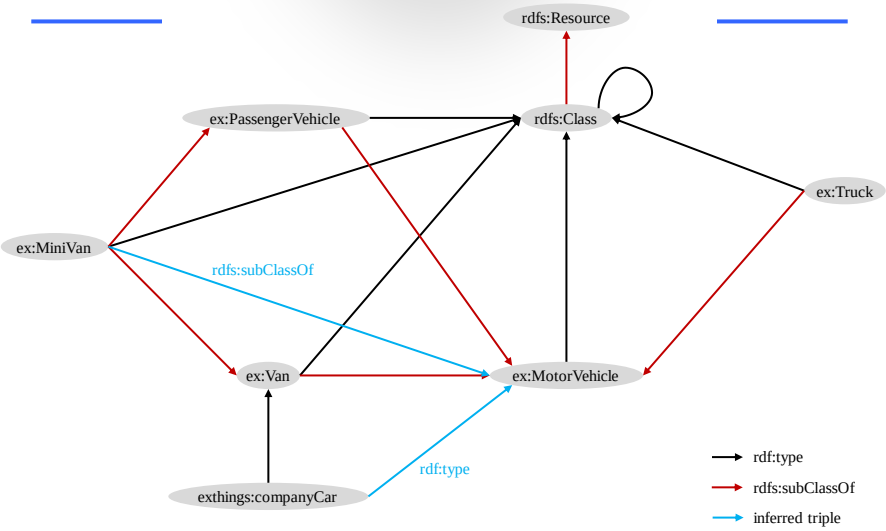
Vocabularies for describing classes

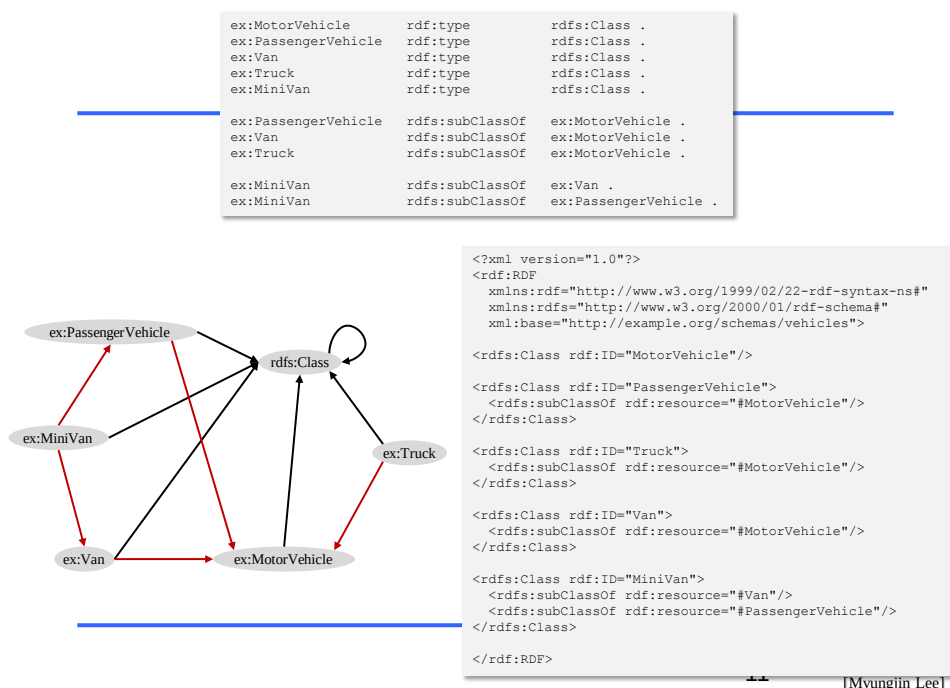
- rdfs:Class
 - the **class of resources**
- rdfs:Resource
 - **all things** described by RDF are called resources, and are instances of the class *rdfs:Resource*.
- rdf:type
 - to state that a resource is **an instance of a class**
- rdfs:subClassOf
 - to state that **all the instances of one class are instances of another**

A Vehicle Class Hierarchy



A Vehicle Class Hierarchy





Instances and rdfs:subClassOf

Instances should not be confused with subclasses

Instances = individual members of a class (e.g. a person name for Person class)

Subclasses = a subset of the members of a class

Depending on the domain being modelled a particular concept could be an instance or a subclass.

Sensors rdf:type rdfs:Class .

TempSensors rdf:type rdfs:Class;

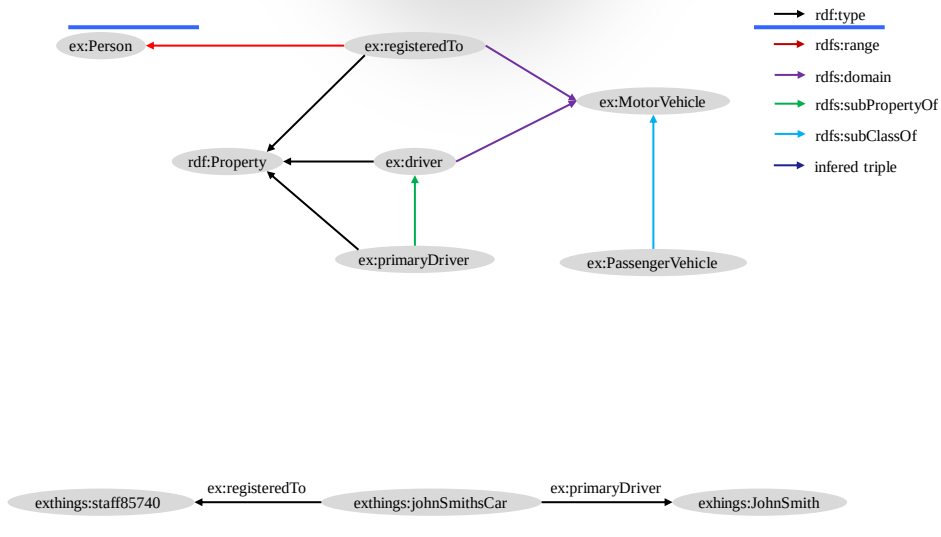
rdfs:subClassOf Sensors.

HumiditySensors rdf:type rdfs:Class;

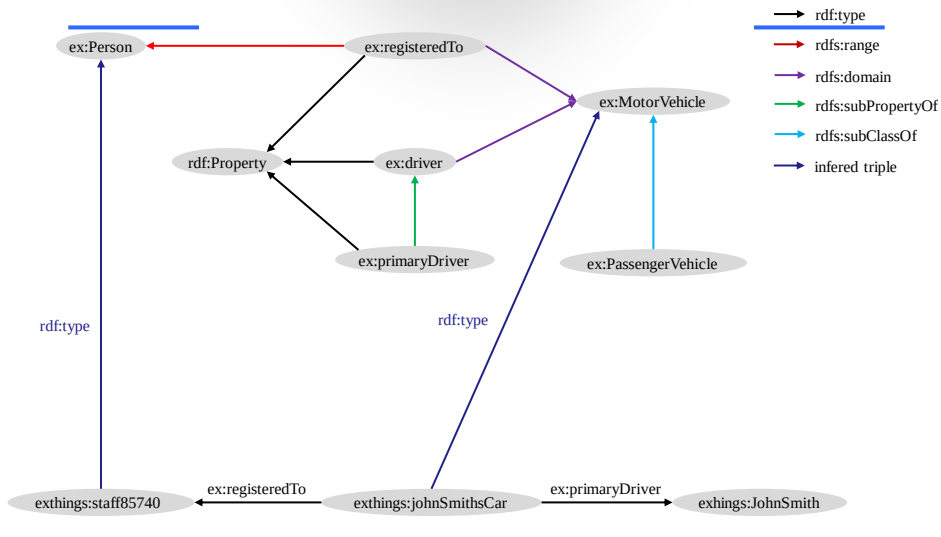
rdfs:subClassOf Sensors.

MelikesSensor ??? TempSensors.

An Instance of Vehicle Schema



An Instance of Vehicle Schema



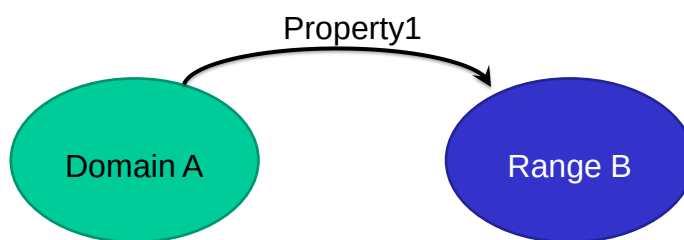
Describing Properties

Vocabularies for describing properties

- `rdf:Property`
 - the **class of RDF properties**
- `rdfs:subPropertyOf`
 - to state that **all resources related by one property are also related by another**
- `rdfs:domain`
 - to state that **any resource that has a given property is an instance of one or more classes**
- `rdfs:range`
 - to state that **the values of a property are instances of one or more classes**

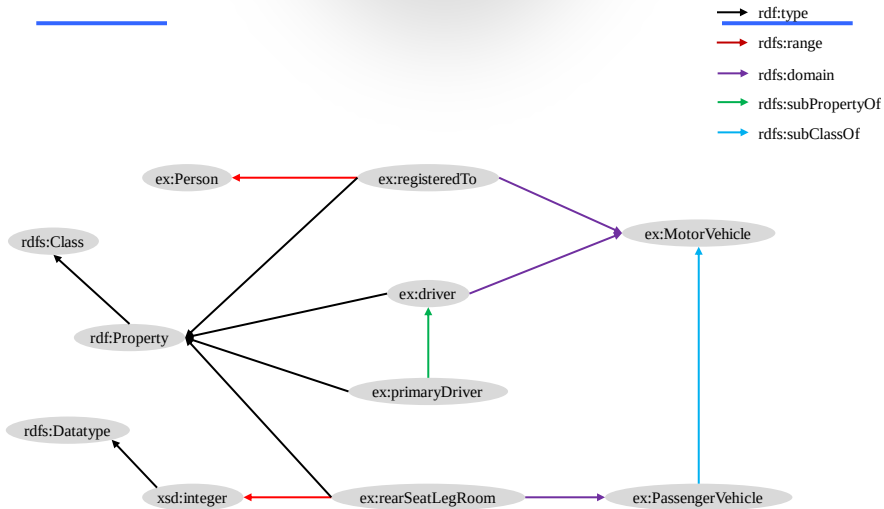
15

Domain and Range

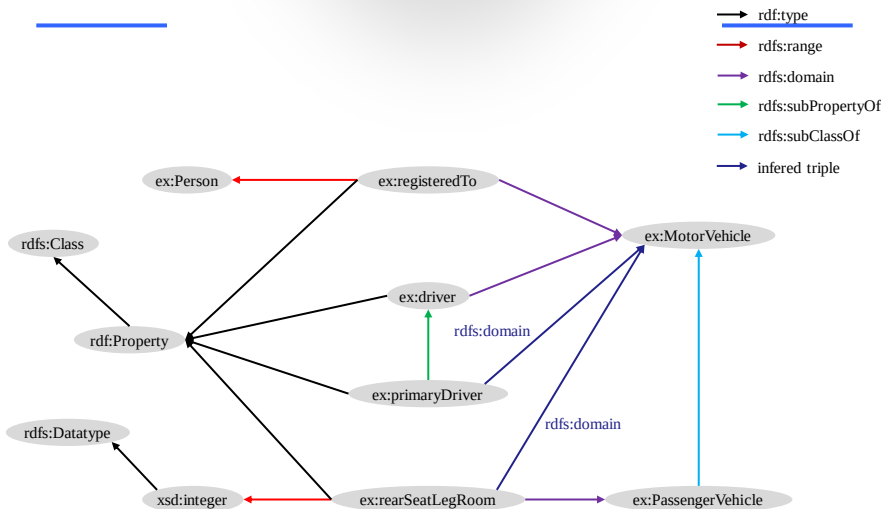


```
A rdf:type rdfs:Class.
B rdf:type rdfs:Class.
property1 rdf:type rdf:Property .
property1 rdfs:domain A .
property1 rdfs:range B .
```

Properties of Vehicle Schema



Properties of Vehicle Schema



TBox and ABox

TBox

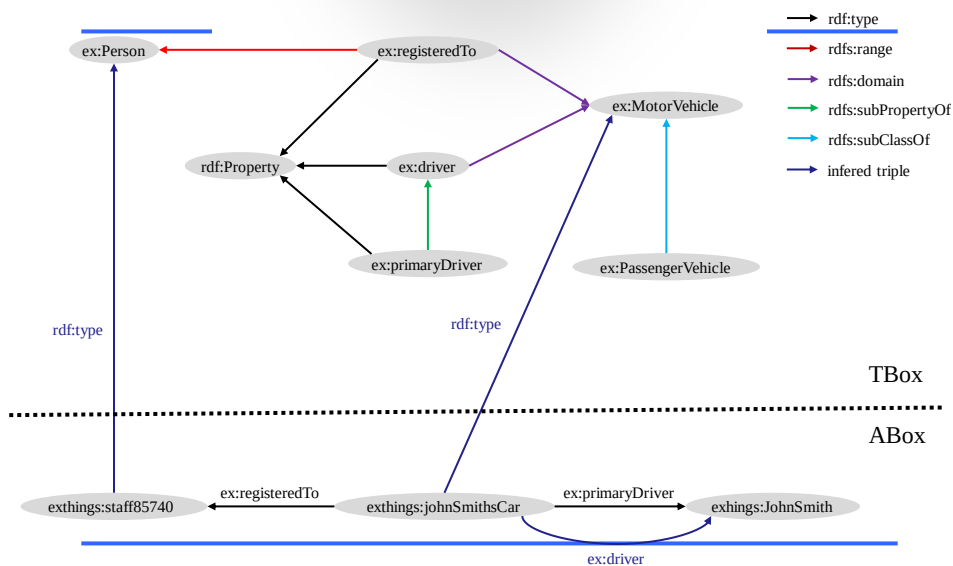
- **"terminological component"**
- a conceptualization associated with a set of facts

ABox

- **"assertion component"**
- a fact associated with a terminological vocabulary within a knowledge base

19

TBox and ABox



Annotation & Utility Properties

These are semantics-free properties => cannot define sub-property, inverse or domain/range for them

Primarily used by tools to **interact with humans**

Annotation properties:

- `rdfs:label` – used to display a human readable label
- `rdfs:comment` – used to include a human-readable comment in the model

Utility properties:

- `rdfs:seeAlso` – used to reference some related information/URI (weak semantics)
- `rdfs:isDefinedBy` – specifies that another resource (URI) defines the subject resource

Annotation Example

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:kdc="http://data.kdata.kr/class/"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

  <kdc:StateDesignatedHeritage
    rdf:about="http://data.kdata.kr/resource/Namdaemun">
    <rdfs:label xml:lang="ko">Korean</rdfs:label>
    <rdfs:label xml:lang="en">Namdaemun</rdfs:label>
    <rdfs:seeAlso
      rdf:resource="http://dbpedia.org/resource/Namdaemun"/>
    <rdfs:comment>Information about Namdaemun</rdfs:comment>
  </kdc:StateDesignatedHeritage>

</rdf:RDF>
```


Entailment Rules

What is Entailment Rules?

- some **inference patterns** which capture some of the various forms of vocabulary entailment used as a guide for the design of software **to check RDF graphs for RDF and RDFS entailment**
- to add a **consequent triple to a graph when it contains triples conforming to a pattern**
 - *a graph entails any larger graph that is obtained by applying the rules to the original graph*

Conventions

- **aaa, bbb**, etc., stand for any URI reference
- **uuu, vvv**, etc. for any URI reference or blank node identifier
- **xxx, yyy** etc. for any URI reference, blank node identifier or literal
- **lll** for any literal
- **_:nnn**, etc., for blank node identifiers

RDF Entailment Rules

Rule Name	if E contains	then add
rdf1	uuu aaa yyy .	aaa rdf:type rdf:Property .
rdf2	uuu aaa lll . where lll is a well-typed XML literal .	_:nnn rdf:type rdf:XMLLiteral . where _:nnn identifies a blank node allocated to lll by rule lg.

RDFS Entailment Rules

Rule Name	if E contains	then add
rdfs1	uuu aaa lll. where lll is a plain literal (with or without a language tag).	_:nnn rdf:type rdfs:Literal . where _:nnn identifies a blank node allocated to lll by rule lg.
rdfs2	aaa rdfs:domain xxx . uuu aaa yyy .	uuu rdf:type xxx .
rdfs3	aaa rdfs:range xxx . uuu aaa vv .	vv rdf:type xxx .
rdfs4a	uuu aaa xxx .	uuu rdf:type rdfs:Resource .
rdfs4b	uuu aaa vv .	vv rdf:type rdfs:Resource .
rdfs5	uuu rdfs:subPropertyOf vv . vv rdfs:subPropertyOf xxx .	uuu rdfs:subPropertyOf xxx .
rdfs6	uuu rdf:type rdf:Property .	uuu rdfs:subPropertyOf uu .
rdfs7	aaa rdfs:subPropertyOf bbb . uuu aaa yyy .	uuu bbb yyy .
rdfs8	uuu rdf:type rdfs:Class .	uuu rdfs:subClassOf rdfs:Resource .
rdfs9	uuu rdfs:subClassOf xxx . vv rdf:type uu .	vv rdf:type xxx .
rdfs10	uuu rdf:type rdfs:Class .	uuu rdfs:subClassOf uu .
rdfs11	uuu rdfs:subClassOf vv . vv rdfs:subClassOf xxx .	uuu rdfs:subClassOf xxx .
rdfs12	uuu rdf:type rdfs:ContainerMembershipProperty .	uuu rdfs:subPropertyOf rdfs:member .
rdfs13	uuu rdf:type rdfs:Datatype .	uuu rdfs:subClassOf rdfs:Literal .

Extensional Entailment Rules

Rule Name	if E contains	then add
ext1	uuu rdfs:domain vv . vv rdfs:subClassOf zz .	uuu rdfs:domain zz .
ext2	uuu rdfs:range vv . vv rdfs:subClassOf zz .	uuu rdfs:range zz .
ext3	uuu rdfs:domain vv . www rdfs:subPropertyOf uu .	www rdfs:domain vv .
ext4	uuu rdfs:range vv . www rdfs:subPropertyOf uu .	www rdfs:range vv .
ext5	rdf:type rdfs:subPropertyOf www . www rdfs:domain vv .	rdfs:Resource rdfs:subClassOf vv .
ext6	rdfs:subClassOf rdfs:subPropertyOf www . www rdfs:domain vv .	rdfs:Class rdfs:subClassOf vv .
ext7	rdfs:subPropertyOf rdfs:subPropertyOf www . www rdfs:domain vv .	rdf:Property rdfs:subClassOf vv .
ext8	rdfs:subClassOf rdfs:subPropertyOf www . www rdfs:range vv .	rdfs:Class rdfs:subClassOf vv .
ext9	rdfs:subPropertyOf rdfs:subPropertyOf www . www rdfs:range vv .	rdf:Property rdfs:subClassOf vv .

Reading

Read W3C's RDF Schema Specification

<http://www.w3.org/TR/rdf-schema/>

29

Structuring Knowledge and
enhancing semantics by adding more
standard vocabulary

Ontology Web Language (OWL)

RDF/RDFS 'Liberality'

No distinction between classes and instances (individuals)

```
Species a rdfs:Class.
Lion a Species.
Leo a Lion.
```

Properties can themselves have properties

```
hasDaughter subPropertyOf hasChild.
hasDaughter a familyProperty.
```

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

```
rdf:type rdfs:range rdfs:Class.
rdf:Property a rdfs:Class.
rdf:type rdfs:subPropertyOf rdfs:subClassOf.
```

Problems with RDFS

RDFS sometimes **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*
- ...

Only limited **reasoning support**

- More likely to have to embed semantics in the application code

Web Ontology Language Requirements

[Desirable features](#) identified for Web Ontology Language:

Extends existing Web standards

- Such as XML, RDF, RDFS

Easy to understand and use

Formally specified

Of 'adequate' expressive power

Possible to provide automated reasoning support

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 ('Description Logic 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
- OWL language now a W3C [Candidate Recommendation](#)

OWL Language

Three species of OWL version 1

- **OWL full** is union of OWL syntax and RDF
- **OWL DL** restricted to the description logic FOL fragment
- **OWL Lite** is 'easier to implement' subset of OWL DL

OWL version 2 profiles (all subsets 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)

Standard prefix declaration (turtle)

@prefix owl: <http://www.w3.org/2002/07/owl#> .

The Species of OWL

OWL Lite

- primarily **needing a classification hierarchy and simple constraint** features
- ex. only to permit cardinality values of 0 or 1

OWL DL

- the **maximum expressiveness** without losing computational **completeness and decidability** of reasoning systems

OWL Full

- the **maximum expressiveness** and the **syntactic freedom** of RDF with **no computational guarantees**

OWL Lite Synopsis

The Species of OWL

RDF Schema Features: • Class (Thing, Nothing) • rdfs:subClassOf • rdf:Property • rdfs:subPropertyOf • rdfs:domain • rdfs:range • Individual	(In)Equality: • equivalentClass • equivalentProperty • sameAs • differentFrom • AllDifferent • distinctMembers	Property Characteristics: • ObjectProperty • DatatypeProperty • inverseOf • TransitiveProperty • SymmetricProperty • FunctionalProperty • InverseFunctionalProperty
Property Restrictions: • Restriction • onProperty • allValuesFrom • someValuesFrom	Restricted Cardinality: • minCardinality (only 0 or 1) • maxCardinality (only 0 or 1) • cardinality (only 0 or 1)	Header Information: • Ontology • Imports
Class Intersection: • intersectionOf	Versioning: • versionInfo • priorVersion • backwardCompatibleWith • incompatibleWith • DeprecatedClass • DeprecatedProperty	Annotation Properties: • rdfs:label • rdfs:comment • rdfs:seeAlso • rdfs:isDefinedBy • AnnotationProperty • OntologyProperty
Datatypes • xsd:datatypes		

OWL DL and Full Synopsis

Class Axioms: • oneOf, dataRange • disjointWith • equivalentClass (applied to class expressions) • rdfs:subClassOf (applied to class expressions)	Boolean Combinations of Class Expressions: • unionOf • complementOf • intersectionOf
Arbitrary Cardinality: • minCardinality • maxCardinality • cardinality	Filler Information: • hasValue

Ontology Representation Language Approach: Ontology Web Language (OWL)

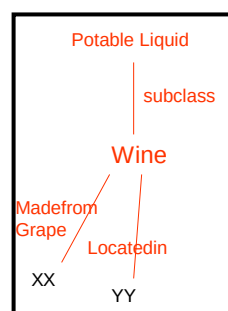
Defined by the W3C,

- <http://www.w3c.org/2001/sw/WebOnt/>

```

Wine      a      owl:Class ;
rdfs:subClassOf PotableLiquid ;
rdfs:subClassOf [
    rdf:type owl:Restriction;
    owl:onProperty madeFromGrape ;
    owl:minCardinality 1
] ;
rdfs:subClassOf [
    rdf:type owl:Restriction ;
    owl:onProperty locatedIn;
    owl:minCardinality 1
] .

```



Namespaces for Ontology

Namespace

- a precise indication of **what specific vocabularies are being used**
- to include a set of XML namespace declarations enclosed in an opening `rdf:RDF` tag

```
<rdf:RDF
  xmlns      = "http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#"
  xmlns:vin  = "http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#"
  xml:base   = "http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#"
  xmlns:food = "http://www.w3.org/TR/2004/REC-owl-guide-20040210/food#"
  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#"
  ...
</rdf:RDF>
```

39

Ontology Headers

Ontology Headers

- a collection of **assertions about the ontology** grouped under an `owl:Ontology` tag for comments, version control and inclusion of other ontologies

```
<owl:Ontology rdf:about="">
  <rdfs:comment>An example OWL ontology</rdfs:comment>
  <owl:priorVersion rdf:resource="http://www.w3.org/TR/2003/PR-owl-guide-20031215/wine"/>
  <owl:imports rdf:resource="http://www.w3.org/TR/2004/REC-owl-guide-20040210/food"/>
  <rdfs:label>Wine Ontology</rdfs:label>
  ...
</owl:Ontology>
```

Syntax

- `owl:Ontology` element
 - a place to collect much of the OWL meta-data for the document
- `owl:priorVersion` element
 - a standard tag intended to provide hooks for version control systems working with ontologies
- `owl:imports` element
 - an include-style mechanism

40

Building Blocks of OWL

Classes

- comparable with classes in RDFS

Individuals

- comparable with objects in RDFS

Properties

- comparable with properties in RDFS

41

Simple Named Classes

owl:Thing

- **Every individual** in the OWL world is **a member of the class owl:Thing**.
- **Each user-defined class** is implicitly **a subclass of owl:Thing**.

owl:Class

- to define **a group of individuals**
- a subclass of rdfs:Class

```
<owl:Class rdf:ID="Winery"/>
<owl:Class rdf:ID="Region"/>
<owl:Class rdf:ID="ConsumableThing"/>
```

rdfs:subClassOf

- the fundamental taxonomic constructor for classes
- If X is a subclass of Y, then every instance of X is also an instance of Y.
- The rdfs:subClassOf relation is transitive.
 - If X is a subclass of Y and Y a subclass of Z then X is a subclass of Z.

```
<owl:Class rdf:ID="PotableLiquid">
  <rdfs:subClassOf rdf:resource="#ConsumableThing" />
  ...
</owl:Class>
```

42

Individuals

Individuals

- to describe members of classes using `rdf:type`

```
<Region rdf:ID="CentralCoastRegion" />
```

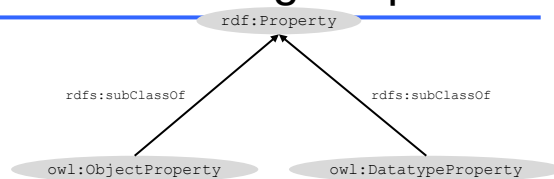
```
<owl:Thing rdf:ID="CentralCoastRegion" />
<owl:Thing rdf:about="#CentralCoastRegion">
  <rdf:type rdf:resource="#Region"/>
</owl:Thing>
```

43

Defining Properties

Property

- a binary relation



Two types of properties

- distinguish properties according to whether they relate individuals to individuals (object properties) or individuals to datatypes (datatype properties)
 - datatype property*
 - **relations between instances of classes and RDF literals and XML Schema datatypes**
 - object property*
 - **relations between instances** of two classes

44

Object Property

Declaration of Object Property

```

<owl:ObjectProperty rdf:ID="madeFromGrape">
  <rdfs:domain rdf:resource="#Wine" />
  <rdfs:range rdf:resource="#WineGrape" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="course">
  <rdfs:domain rdf:resource="#Meal" />
  <rdfs:range rdf:resource="#MealCourse" />
</owl:ObjectProperty>

<owl:Thing rdf:ID="LindemansBin65Chardonnay">
  <madeFromGrape rdf:resource="#ChardonnayGrape" />
</owl:Thing>

```

45

Properties and Datatypes

xsd:string	xsd:normalizedString	xsd:boolean
xsd:decimal	xsd:float	xsd:double
xsd:integer	xsd:nonNegativeInteger	xsd:positiveInteger
xsd:nonPositiveInteger	xsd:negativeInteger	xsd:long
xsd:int	xsd:short	xsd:byte
xsd:unsignedLong	xsd:unsignedInt	xsd:unsignedShort
xsd:unsignedByte	xsd:hexBinary	xsd:base64Binary
xsd:dateTime	xsd:time	xsd:date
xsd:gYearMonth	xsd:gYear	xsd:gMonthDay
xsd:gDay	xsd:gMonth	xsd:anyURI
xsd:token	xsd:language	xsd:NMTOKEN
xsd:Name	xsd:NCName	

```

<owl:Class rdf:ID="VintageYear" />

<owl:DatatypeProperty rdf:ID="yearValue">
  <rdfs:domain rdf:resource="#VintageYear" />
  <rdfs:range rdf:resource="xsd:positiveInteger"/>
</owl:DatatypeProperty>

```

```

<VintageYear rdf:ID="Year1998">
  <yearValue rdf:datatype="xsd:positiveInteger">1998</yearValue>
</VintageYear>

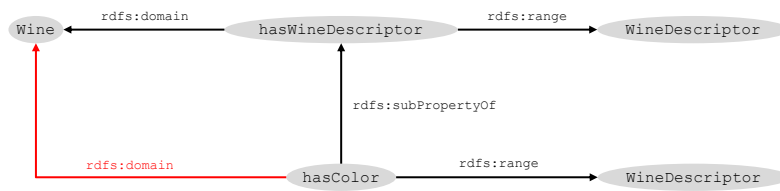
```

46

Properties in a Hierarchy

```
<owl:ObjectProperty rdf:ID="hasWineDescriptor">
  <rdfs:domain rdf:resource="#Wine" />
  <rdfs:range rdf:resource="#WineDescriptor" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasColor">
  <rdfs:subPropertyOf rdf:resource="#hasWineDescriptor" />
  <rdfs:range rdf:resource="#WineColor" />
  ...
</owl:ObjectProperty>
```



Property Characteristics

TransitiveProperty

SymmetricProperty

FunctionalProperty

inverseOf

InverseFunctionalProperty

TransitiveProperty

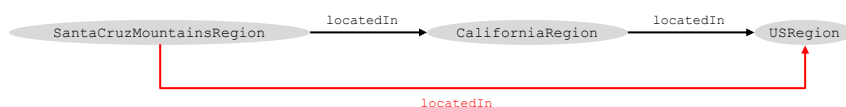
If a property, P , is specified as transitive then for any x , y , and z :

- $P(x, y)$ and $P(y, z)$ implies $P(x, z)$

```
<owl:ObjectProperty rdf:ID="locatedIn">
  <rdf:type rdf:resource="#owl:TransitiveProperty" />
  <rdfs:domain rdf:resource="#owl:Thing" />
  <rdfs:range rdf:resource="#Region" />
</owl:ObjectProperty>

<Region rdf:ID="SantaCruzMountainsRegion">
  <locatedIn rdf:resource="#CaliforniaRegion" />
</Region>

<Region rdf:ID="CaliforniaRegion">
  <locatedIn rdf:resource="#USRegion" />
</Region>
```



49

[Myungjin Lee]

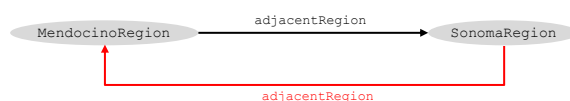
SymmetricProperty

If a property, P , is tagged as symmetric then for any x and y :

- $P(x, y)$ iff $P(y, x)$

```
<owl:ObjectProperty rdf:ID="adjacentRegion">
  <rdf:type rdf:resource="#owl:SymmetricProperty" />
  <rdfs:domain rdf:resource="#Region" />
  <rdfs:range rdf:resource="#Region" />
</owl:ObjectProperty>

<Region rdf:ID="MendocinoRegion">
  <locatedIn rdf:resource="#CaliforniaRegion" />
  <adjacentRegion rdf:resource="#SonomaRegion" />
</Region>
```



50

[Myungjin Lee]

FunctionalProperty

If a property, P , is tagged as functional then for all x , y , and z :

- $P(x, y)$ and $P(x, z)$ implies $y = z$

```
<owl:Class rdf:ID="VintageYear" />
<owl:ObjectProperty rdf:ID="hasVintageYear">
  <rdf:type rdf:resource="#owl:FunctionalProperty" />
  <rdfs:domain rdf:resource="#Vintage" />
  <rdfs:range rdf:resource="#VintageYear" />
</owl:ObjectProperty>
```



51

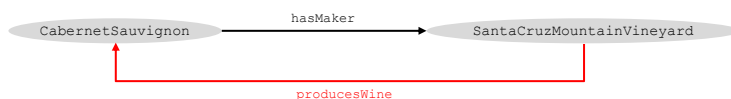
[Myungjin Lee]

inverseOf

If a property, P_1 , is tagged as the `owl:inverseOf` P_2 , then for all x and y :

- $P_1(x, y)$ iff $P_2(y, x)$

```
<owl:ObjectProperty rdf:ID="hasMaker">
  <rdf:type rdf:resource="#owl:FunctionalProperty" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="producesWine">
  <owl:inverseOf rdf:resource="#hasMaker" />
</owl:ObjectProperty>
```



52

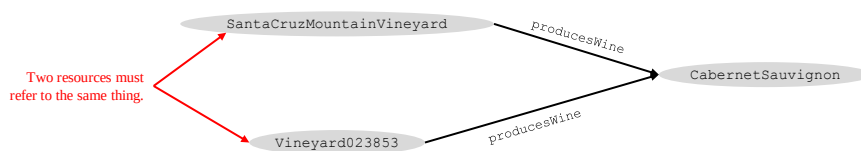
[Myungjin Lee]

InverseFunctionalProperty

If a property, P, is tagged as InverseFunctional then for all x, y and z:

- $P(y, x)$ and $P(z, x)$ implies $y = z$

```
<owl:ObjectProperty rdf:ID="hasMaker" />
<owl:ObjectProperty rdf:ID="producesWine">
  <rdf:type rdf:resource="#owl:InverseFunctionalProperty" />
  <owl:inverseOf rdf:resource="#hasMaker" />
</owl:ObjectProperty>
```



53

[Myungjin Lee]

Equivalence between Classes and Properties

`owl:equivalentClass`

- to indicate that two classes have precisely the same instances

```
<owl:Class rdf:ID="Wine">
  <owl:equivalentClass rdf:resource="#vin;Wine"/>
</owl:Class>
```

`owl:equivalentProperty`

- to tie together properties

```
<owl:DatatypeProperty rdf:ID="name">
  <owl:equivalentProperty rdf:resource="http://purl.org/metadata/dublin-core#Title"/>
</owl:DatatypeProperty>
```

54

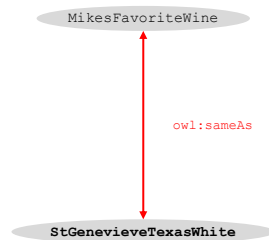
[Myungjin Lee]

Identity between Individuals

owl:sameAs

- to declare two individuals to be identical

```
<Wine rdf:ID="MikesFavoriteWine">
  <owl:sameAs rdf:resource="#StGenevieveTexasWhite" />
</Wine>
```



55

[Myungjin Lee]

Different Individuals

owl:differentFrom

- to provide the opposite effect from sameAs

```
<WineSugar rdf:ID="Dry" />

<WineSugar rdf:ID="Sweet">
  <owl:differentFrom rdf:resource="#Dry"/>
</WineSugar>

<WineSugar rdf:ID="OffDry">
  <owl:differentFrom rdf:resource="#Dry"/>
  <owl:differentFrom rdf:resource="#Sweet"/>
</WineSugar>
```

owl:AllDifferent

- to define a set of mutually distinct individuals

```
<owl:AllDifferent>
  <owl:distinctMembers rdf:parseType="Collection">
    <vin:WineColor rdf:about="#Red" />
    <vin:WineColor rdf:about="#White" />
    <vin:WineColor rdf:about="#Rose" />
  </owl:distinctMembers>
</owl:AllDifferent>
```

56

[Myungjin Lee]

Property Restrictions

Property Restrictions

- to further constrain the range of a property in specific contexts
- to indicate the restricted property using the `owl:onProperty` element within the context of an `owl:Restriction`

the Various Forms of Restriction

- `allValuesFrom`, `someValuesFrom`
- Cardinality
- `hasValue`

57

[Myungjin Lee]

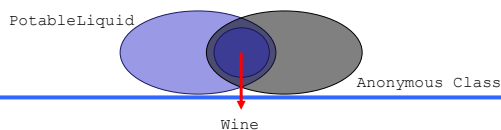
allValuesFrom

owl:allValuesFrom

- to require that for every instance of the class that has instances of the specified property
- the values of the property are all members of the class indicated by the `owl:allValuesFrom` clause

anonymous class

```
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="#food;PotableLiquid" />
  ...
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMaker" />
      <owl:allValuesFrom rdf:resource="#Winery" />
    </owl:Restriction>
  </rdfs:subClassOf>
  ...
</owl:Class>
```

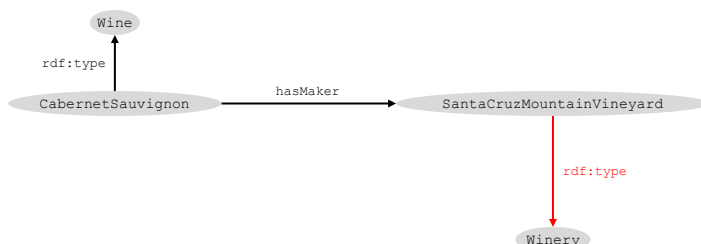


58

[Myungjin Lee]

allValuesFrom

```
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="&food;PotableLiquid" />
  ...
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMaker" />
      <owl:allValuesFrom rdf:resource="#Winery" />
    </owl:Restriction>
  </rdfs:subClassOf>
  ...
</owl:Class>
```

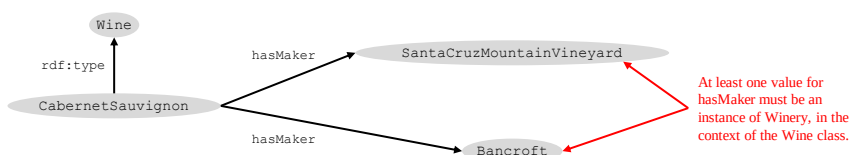


someValuesFrom

owl:someValuesFrom

- at least one of the `hasMaker` properties of a `Wine` must point to an individual that is a `Winery`

```
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="&food;PotableLiquid" />
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMaker" />
      <owl:someValuesFrom rdf:resource="#Winery" />
    </owl:Restriction>
  </rdfs:subClassOf>
  ...
</owl:Class>
```



allValuesFrom vs. someValuesFrom

Relation	Implications
allValuesFrom	For all wines, if they have makers, all the makers are wineries.
someValuesFrom	For all wines, they have at least one maker that is a winery.

owl:allValuesFrom

```
<owl:onProperty rdf:resource="#emptiesInto"/>
<owl:allValuesFrom rdf:resource="#Sea"/>
```

- Wherever there is an emptiesInto property, all its values must be instances of Sea. [There may be zero emptiesInto properties.]

owl:someValuesFrom

```
<owl:onProperty rdf:resource="#connectsTo"/>
<owl:someValuesFrom rdf:resource="#BodyOfWater"/>
```

- There must be at least one connectsTo property whose value is BodyOfWater. [There must be at least one connectsTo property.]

Cardinality

owl:cardinality

- the specification of exactly the number of elements in a relation

```
<owl:Class rdf:ID="Vintage">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasVintageYear"/>
      <owl:cardinality rdf:datatype="xsd:nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

every Vintage has exactly one VintageYear

Cardinality

owl:minCardinality

- to specify a *lower* bound

owl:maxCardinality

- to specify an *upper* bound

```
<owl:Class rdf:ID="Vintage">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#vintageOf"/>
      <owl:minCardinality rdf:datatype="xsd:nonNegativeInteger">1</owl:minCardinality>
      <owl:maxCardinality rdf:datatype="xsd:nonNegativeInteger">5</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

63

[Myungjin Lee]

hasValue

owl:hasValue

- to specify classes based on the existence of particular property values
- at least one of its property values is equal to the hasValue resource

```
<owl:Class rdf:ID="Burgundy">
  ...
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasSugar" />
      <owl:hasValue rdf:resource="#Dry" />
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

all *Burgundy* wines are *dry*.
their *hasSugar* property must have at least one value that is equal to *Dry*.

64

[Myungjin Lee]

Complex Classes

Set Operators

- intersectionOf, unionOf, complementOf

Enumerated Classes

- oneOf

Disjoint Classes

- disjointWith

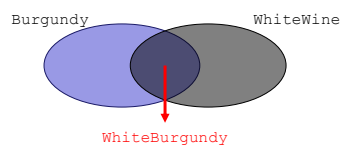
65

intersectionOf

```
<owl:Class rdf:ID="WhiteWine">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Wine" />
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasColor" />
      <owl:hasValue rdf:resource="#White" />
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>

<owl:Class rdf:about="#Burgundy">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Wine" />
    <owl:Restriction>
      <owl:onProperty rdf:resource="#locatedIn" />
      <owl:hasValue rdf:resource="#BourgogneRegion" />
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>

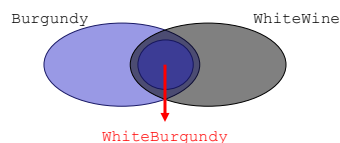
<owl:Class rdf:ID="WhiteBurgundy">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Burgundy" />
    <owl:Class rdf:about="#WhiteWine" />
  </owl:intersectionOf>
</owl:Class>
```



intersectionOf

Contrast with defining *Fleuve* using two `subClassOf` statements

```
<owl:Class rdf:ID="WhiteBurgundy">
  <rdfs:subClassOf rdf:about="#Burgundy" />
  <rdfs:subClassOf rdf:about="#WhiteWine" />
</owl:Class>
```



Contrast

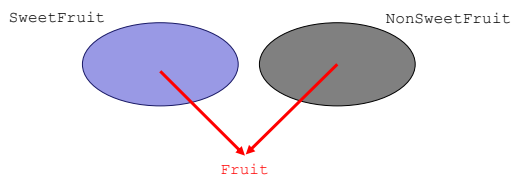
- Defining a *WhiteBurgundy* using two `subClassOf` elements: all instances of *WhiteBurgundy* must be a *Burgundy* and *WhiteWine*.
- Defining a *WhiteBurgundy* using `intersectionOf`: a *WhiteBurgundy* is the collection of all instances that is both a *Burgundy* and *WhiteWine*.
- Thus, the `subClassOf` form merely characterizes a *WhiteBurgundy*, whereas the `intersectionOf` form defines a *WhiteBurgundy*.

67

[Myungjin Lee]

unionOf

```
<owl:Class rdf:ID="Fruit">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#SweetFruit" />
    <owl:Class rdf:about="#NonSweetFruit" />
  </owl:unionOf>
</owl:Class>
```

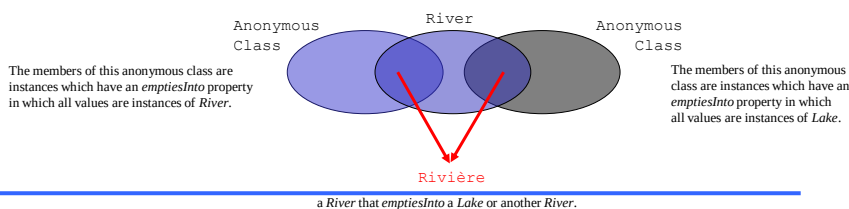


68

[Myungjin Lee]

Example of Intersection and Union

```
<owl:Class rdf:ID="Rivière">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#River"/>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:onProperty rdf:resource="#emptiesInto"/>
          <owl:allValuesFrom rdf:resource="#Lake"/>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty rdf:resource="#emptiesInto"/>
          <owl:allValuesFrom rdf:resource="#River"/>
        </owl:Restriction>
      </owl:unionOf>
    </owl:Class>
  </owl:intersectionOf>
</owl:Class>
```



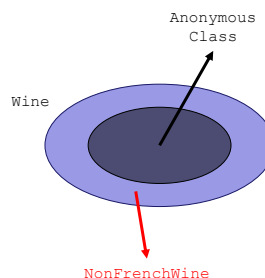
Complement

owl:complementOf

- to select all individuals from the domain of discourse that do not belong to a certain class

```
<owl:Class rdf:ID="ConsumableThing" />
<owl:Class rdf:ID="NonConsumableThing">
  <owl:complementOf rdf:resource="#ConsumableThing" />
</owl:Class>
```

```
<owl:Class rdf:ID="NonFrenchWine">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Wine"/>
    <owl:Class>
      <owl:complementOf>
        <owl:Restriction>
          <owl:onProperty rdf:resource="#locatedIn" />
          <owl:hasValue rdf:resource="#FrenchRegion" />
        </owl:Restriction>
      </owl:complementOf>
    </owl:Class>
  </owl:intersectionOf>
</owl:Class>
```



Enumerated Classes

`owl:oneOf`

- to specify a class via a direct enumeration of its members

```
<owl:Class rdf:ID="WineColor">
  <rdfs:subClassOf rdf:resource="#WineDescriptor"/>
  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="#White"/>
    <owl:Thing rdf:about="#Rose"/>
    <owl:Thing rdf:about="#Red"/>
  </owl:oneOf>
</owl:Class>
```

no other individuals can be a valid *WineColor* since the class has been defined by enumeration

71

[Myungjin Lee]

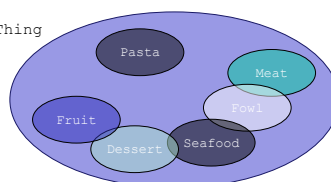
Disjoint Classes

`owl:disjointWith`

- an individual that is a member of one class cannot simultaneously be a member of another class

```
<owl:Class rdf:ID="Pasta">
  <rdfs:subClassOf rdf:resource="#EdibleThing"/>
  <owl:disjointWith rdf:resource="#Meat"/>
  <owl:disjointWith rdf:resource="#Fowl"/>
  <owl:disjointWith rdf:resource="#Seafood"/>
  <owl:disjointWith rdf:resource="#Dessert"/>
  <owl:disjointWith rdf:resource="#Fruit"/>
</owl:Class>
```

EdibleThing



The above class definition only states that there are no instances of *Pasta* which overlap with *Meat*, *Fowl*, *Seafood*, *Dessert*, or *Fruit*. It does not state that all four classes are disjoint.

72

[Myungjin Lee]

Disjoint Classes

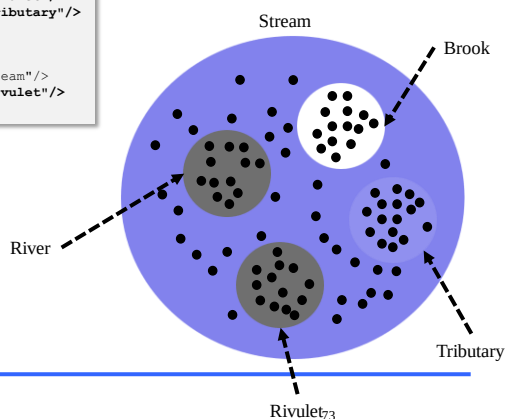
```

<owl:Class rdf:ID="River">
  <rdfs:subClassOf rdf:resource="#Stream"/>
  <owl:disjointWith rdf:resource="#Brook"/>
  <owl:disjointWith rdf:resource="#Rivulet"/>
  <owl:disjointWith rdf:resource="#Tributary"/>
</owl:Class>

<owl:Class rdf:ID="Brook">
  <rdfs:subClassOf rdf:resource="#Stream"/>
  <owl:disjointWith rdf:resource="#Rivulet"/>
  <owl:disjointWith rdf:resource="#Tributary"/>
</owl:Class>

<owl:Class rdf:ID="Tributary">
  <rdfs:subClassOf rdf:resource="#Stream"/>
  <owl:disjointWith rdf:resource="#Rivulet"/>
</owl:Class>

```

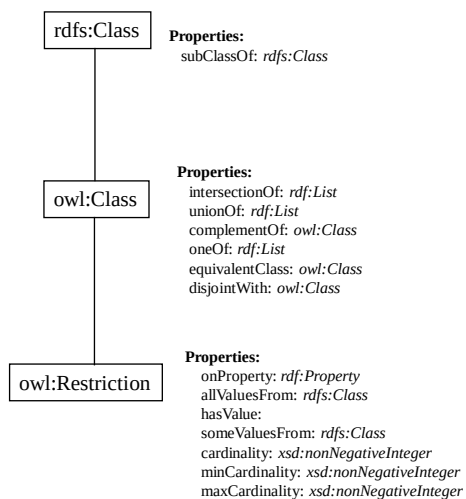


OWL as Description Language

Axiom	DL Syntax	Example
subClassOf	$C_1 \sqsubseteq C_2$	$\text{Human} \sqsubseteq \text{Animal} \sqcap \text{Biped}$
equivalentClass	$C_1 \equiv C_2$	$\text{Man} \equiv \text{Human} \sqcap \text{Male}$
disjointWith	$C_1 \sqsubseteq \neg C_2$	$\text{Male} \sqsubseteq \neg \text{Female}$
sameIndividualAs	$\{x_1\} \equiv \{x_2\}$	$\{\text{President_Bush}\} \equiv \{\text{G_W_Bush}\}$
differentFrom	$\{x_1\} \sqsubseteq \neg \{x_2\}$	$\{\text{john}\} \sqsubseteq \neg \{\text{peter}\}$
subPropertyOf	$P_1 \sqsubseteq P_2$	$\text{hasDaughter} \sqsubseteq \text{hasChild}$
equivalentProperty	$P_1 \equiv P_2$	$\text{cost} \equiv \text{price}$
inverseOf	$P_1 \equiv P_2^-$	$\text{hasChild} \equiv \text{hasParent}^-$
transitiveProperty	$P^+ \sqsubseteq P$	$\text{ancestor}^+ \sqsubseteq \text{ancestor}$
functionalProperty	$T \sqsubseteq \leq 1P$	$T \sqsubseteq \leq 1 \text{hasMother}$
inverseFunctionalProperty	$T \sqsubseteq \leq 1P^-$	$T \sqsubseteq \leq 1 \text{hasSSN}^-$

Constructor	DL Syntax	Example	Modal Syntax
intersectionOf	$C_1 \sqcap \dots \sqcap C_n$	$\text{Human} \sqcap \text{Male}$	$C_1 \wedge \dots \wedge C_n$
unionOf	$C_1 \sqcup \dots \sqcup C_n$	$\text{Doctor} \sqcup \text{Lawyer}$	$C_1 \vee \dots \vee C_n$
complementOf	$\neg C$	$\neg \text{Male}$	$\neg C$
oneOf	$\{x_1\} \sqcup \dots \sqcup \{x_n\}$	$\{\text{john}\} \sqcup \{\text{mary}\}$	$x_1 \vee \dots \vee x_n$
allValuesFrom	$\forall P.C$	$\forall \text{hasChild}.\text{Doctor}$	$[P]C$
someValuesFrom	$\exists P.C$	$\exists \text{hasChild}.\text{Lawyer}$	$\langle P \rangle C$
maxCardinality	$\leq nP$	$\leq 1 \text{hasChild}$	$[P]_{n+1}$
minCardinality	$\geq nP$	$\geq 2 \text{hasChild}$	$\langle P \rangle_n$

Summary of Class Properties



RDF,RDFS,OWL,SWRL

© Declan O'Sullivan

75

Reading

OWL (2004 Spec) <http://www.w3.org/TR/2004/REC-owl-guide-20040210/>

OWL 2 (2012 Spec) <http://www.w3.org/TR/owl2-overview/>

76

Next Steps

Download Protégé Ontology Editor

We will create an ontology and populate with instances