

# INFO216: **Advanced Modelling**

Theme, spring 2018:  
**Modelling and Programming  
the Web of Data**

Andreas L. Opdahl  
<Andreas.Opdahl@uib.no>



# Session S06: RDFS Plus

- Themes:
  - what and why?
  - basic OWL constructs (“RDFS-Plus”)
- Programming:
  - Jena *OntModel* class
  - *OntClass*, *Individual*, *ObjectProperty*, *DatatypeProperty*
  - OWL class that defines OWL-terms / IRIs
  - *OntModelSpec* class that defines reasoner types



# Readings

- Allemang & Hendler (2011):  
Semantic Web for the Working Ontologist
  - chapter 8 (“RDFS Plus”)
- Forum links (cursory):
  - OWL 2 Overview:  
*<http://www.w3.org/TR/owl-overview/>*
  - OWL 2 Primer:  
*<http://www.w3.org/TR/owl-primer/>*
    - **show**: Turtle and Manchester syntax
    - **hide**: other syntaxes



# Web Ontology Language (OWL)



# RDFS is a useful starting point...

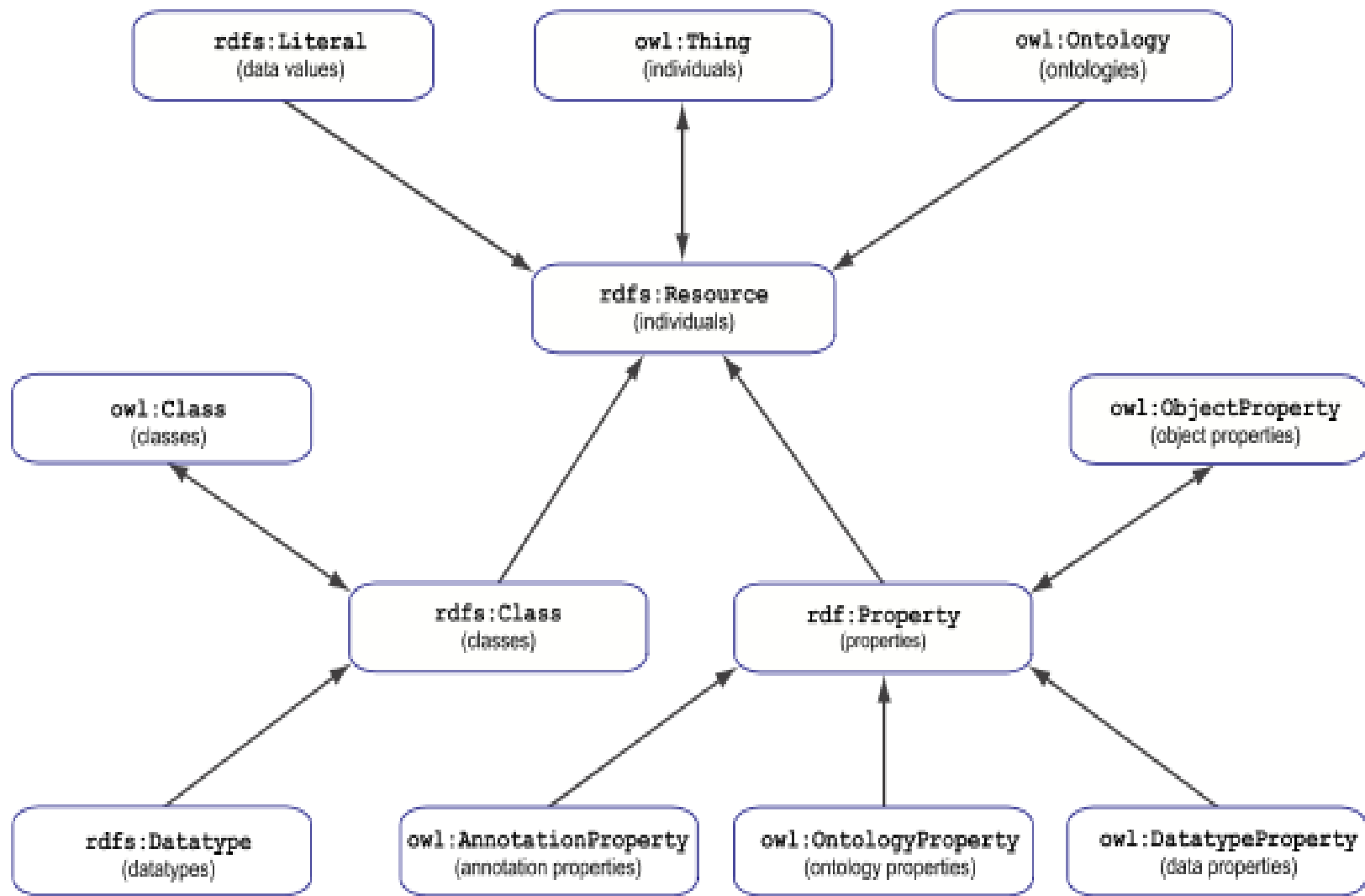
- But there's lots of simple stuff it cannot express, e.g.:
  - “every ancestor of an ancestor is an ancestor too”
  - “the BirthNumber of a Person is unique”
  - “a Republic has exactly one President”
  - “a FootballTeam has 11 activePlayers, a VolleyballTeam 6”
  - “a StringQuartet has two violins but only one viola and one cello”
  - “classes with different IRIs actually represent the same class”
  - “resources with different IRIs represent the same resource”
  - “properties with different IRIs are actually the same”
  - “two individuals are different”, “two classes are disjoint”
  - “a class is a union (or intersection) of other classes”
  - “a class is a negation of another class”
- *OWL expresses all this and more!*



# Basic idea

- Web Ontology Language (OWL):
  - builds on RDF and RDFS
  - uses classes and properties from RDF and RDFS
  - adds precision and formality
- Basic OWL-concepts:
  - `owl:Class` `rdfs:subClassOf` `rdfs:Class` .
  - “`owl:Property`” `rdfs:subClassOf` `rdf:Property` .
  - “`owl:Individual`” `rdfs:subClassOf` `rdfs:Resource` .  
good practice: keep these three *disjoint*, i.e., no resource has more than one of them as *rdf:type*
    - “*owl:Individual*” is actually called *owl:Thing*





# What does OWL offer?

- Extensions of RDFS, e.g.:
  - more *specific types* of properties
  - *identical and different* classes, properties, individuals
  - *defining new classes*:
    - complex classes (union, intersection, complement)
    - property restrictions, enumeration of individuals
  - *defining new properties* based on existing ones
  - *mathematical formality* (for large parts of OWL)
    - (more on this later)





# Reuses or specialises RDFS

- *Reused* in OWL:
  - `rdf:type`, `rdf:Property`, `rdfs:subClassOf`,  
`rdfs:subPropertyOf`, `rdfs:domain`, `rdfs:range`
  - ...and lots of other stuff...
- *Renamed* by OWL:
  - `owl:Thing owl:sameAs rdfs:Resource` .
- *Specialised* by OWL:
  - everything else in OWL *specialises* something in RDFS



# Basic OWL ("RDFS-Plus")



# Inverse properties

- Properties can be each other's reverses (with subject and object swapped), e.g.,
  - `rex:HaakonMagnus fam:hasParent rex:Harald .`
  - `rex:Harald fam:hasChild rex:HaakonMagnus .`
- `P1 owl:inverseOf P2`:
  - `fam:hasParent owl:inverseOf fam:hasChild .`
  - `owl:inverseOf owl:inverseOf owl:inverseOf .`
  - `owl:inverseOf` a `owl:ObjectProperty` .
- “Entailment rules”:
  - if *`P1 owl:inverseOf P2`* then
    - *`P2 owl:inverseOf P1 .`*
  - if *`S P1 O . P1 owl:inverseOf P2`* then
    - *`O P2 S .`*



# Symmetric properties

- Some properties are their own inverse, e.g.,
  - `rex:Harald fam:marriedTo rex:Sonja .`
  - `rex:Sonja fam:marriedTo rex:Harald .`
- `P rdf:type owl:SymmetricProperty`:
  - `fam:marriedTo a owl:SymmetricProperty .`
  - `owl:inverseOf a owl:SymmetricProperty .`
  - `owl:SymmetricProperty rdfs:subClassOf owl:ObjectProperty .`
- Entailment rules:
  - if  *$P$  a `owl:SymmetricProperty`* then
    - *$P$  `owl:inverseOf`  $P$  .*
  - if  *$S P O . P$  a `owl:SymmetricProperty`* then
    - *$O P S .$*



# Asymmetric, reflexive, irreflexive properties

- New in OWL2:
  - both *reflexive* and *irreflexive* properties:
    - owl:sameAs a owl:ReflexiveProperty .
      - “every resource is owl:sameAs itself”
    - fam:hasChild a owl:IrreflexiveProperty .
      - “no resource can be fam:hasChild of itself”
    - *many properties are neither!*
  - both *symmetric* and *asymmetric* properties:
    - fam:marriedTo a owl:SymmetricProperty .
      - “fam:marriedTo is always mutual (two-way)”
    - fam:hasChild a owl:AsymmetricProperty .
      - “no resources can be fam:hasChild of each other”
    - *many properties are neither!*



# Transitive properties

- Some properties can form chains so that the result is the property itself, e.g.:
  - `rex:HaakonMagnus fam:hasAncestor rex:Harald .`
  - `rex:Harald fam:hasAncestor rex:Olav .`
  - `rex:HaakonMagnus fam:hasAncestor rex:Olav .`
- `P a owl:TransitiveProperty`:
  - `fam:hasAncestor a owl:TransitiveProperty .`
  - `rdfs:subClassOf a owl:TransitiveProperty .`
  - `rdfs:subPropertyOf a owl:TransitiveProperty .`
- Entailment rules:
  - “if *`S P X . X P O . P a owl:TransitiveProperty`* then
    - *`S P O .`*”



# Functional properties

- Each subject *can only have one* object value for the functional property, e.g.,
  - fam:mother a owl:FunctionalProperty .
  - fam:birthdate a owl:FunctionalProperty .
  - owl:FunctionalProperty rdfs:subClassOf “owl:Property” .
- “Entailment rule”:
  - if  $S P O1 . S P O2 . P$  a *owl:FunctionalProperty* then
    - $O1$  *owl:sameAs*  $O2$  .
  - This rule is for *owl:ObjectProperties*
  - There is a corresponding rule for *owl:DatatypeProperties*
    - but if two different literals become asserted as *owl:sameAs* one another, the ontology is inconsistent



# Inverse functional properties

- Two different subjects cannot have the same object for an inverse functional property, i.e.,
  - fam:persNum a owl:InverseFunctionalObjectProperty .
  - fam:persNum a owl:FunctionalProperty .
  - owl:FunctionalProperty  
owl:inverseOf owl:InverseFunctionalObjectProperty .
- Inverse functional properties are *unique* for each individual
  - used for *identifiers* in OWL 1
  - OWL 2 has a built-in *owl:hasKey* property for identifiers:
    - similar to inverse functional object properties
    - can only be used with OWL 2's *owl:NamedIndividuals*
    - ...not for anonymous *owl:Individuals*





# Summary: more specific properties

- owl:inverseOf
- owl:SymmetricProperty, owl:AsymmetricProperty
- owl:ReflexiveProperty, owl:IrreflexiveProperty
- owl:TransitiveProperty
- owl:FunctionalProperty, owl:InverseFunctionalProperty
- owl:hasKey
- Also:
  - negated properties (later)
  - chained properties, e.g.:  
fam:hasGrandparent  
owl:propertyChainAxiom ( :hasParent :hasParent ) .



# Individual equivalence

- Two individuals (with different IRI-s) may represent the same thing:
  - [http://dbpedia.org/resource/Amanda\\_Plummer](http://dbpedia.org/resource/Amanda_Plummer)
  - [http://yago-knowledge.org/resource/Amanda\\_Plummer](http://yago-knowledge.org/resource/Amanda_Plummer)
  - <http://data.linkedmdb.org/resource/actor/34880>
- I1 owl:sameAs I2:
  - owl:sameAs a owl:ReflexiveProperty .
  - owl:sameAs a owl:SymmetricProperty .
  - owl:sameAs a owl:TransitiveProperty .
- owl:sameAs is an *equivalence relation*:
  - because it is *reflexive*, *symmetric* and *transitive*



# Unique Name Assumption (UNA)

- If two resources have different names, do they necessarily represent different things?
- RDF and OWL does *not* assume this!
  - *in RDF and OWL, we do not know whether resources with different names represent different things or not*
- We can use
  - `owl:sameAs` – two resources represent the same thing!
  - `owl:differentFrom` – they represent different things!
- Some ICT-languages and technologies use UNA, others do not!



# Individual difference

- A *pair* of individuals with different names (IRI-s) must represent different things, e.g.,
  - cal:Spring owl:differentFrom cal:Summer .
- ...is *not* transitive



# Individual difference

- *A pair* of individuals with different names (IRI-s) must represent different things, e.g.,
  - `cal:Spring owl:differentFrom cal:Summer` .
- *A group* of individuals with different names (IRI-s) must represent different things, e.g.,
  - `[ a owl:AllDifferent ] owl:distinctMembers ( cal:Spring cal:Summer cal:Autumn cal:Winter )` .



# Individual difference

- *A pair* of individuals with different names (IRI-s) must represent different things, e.g.,
  - `cal:Spring owl:differentFrom cal:Summer` .
- *A group* of individuals with different names (IRI-s) must represent different things, e.g.,
  - `[ a owl:AllDifferent ] owl:distinctMembers ( cal:Spring cal:Summer cal:Autumn cal:Winter )` .
  - *owl:AllDifferent* and *owl:distinctMembers* are special constructs in OWL
    - they must always be used together
  - ...corresponds to pairwise *owl:differentFrom* between *all* individuals in the *owl:distinctMembers*-list



# Equivalent classes

- Two classes (with different IRI-s) represent the same class:
- C1 owl:equivalentClass C2:
  - owl:equivalentClass a owl:ReflexiveProperty .
  - owl:equivalentClass a owl:SymmetricProperty .
  - owl:equivalentClass a owl:TransitiveProperty .
- owl:equivalentClass is another *equivalence relation*:
  - it is *reflexive*, *symmetric* and *transitive*
- means the same as
  - C1 rdfs:subClassOf C2 and C2 rdfs:subClassOf C1



# Disjoint classes

- Some classes cannot have the same individual as a member,
  - fam:Male owl:disjointWith fam:Female .
  - owl:disjointWith a owl:SymmetricProperty .
    - ...but it is *not* transitive
- I.e., no individual can have both classes as its rdf:type
  - ...corresponds to owl:differentFrom between *all* pairs of individuals in fam:Male and fam:Female
- Preferred in *formal* versions of OWL (no “punning”):
  - owl:Class owl:disjointWith “owl:Property” .
  - owl:Class owl:disjointWith “owl:Individual” .
  - “owl:Property” owl:disjointWith owl:Individual .





# Equivalent properties

- Two properties (with different IRI-s) represent the same property:
- P1 owl:equivalentProperty P2:
  - owl:equivalentProperty a owl:ReflexiveProperty .
  - owl:equivalentProperty a owl:SymmetricProperty .
  - owl:equivalentProperty a owl:TransitiveProperty .
- owl:equivalentProperty is another *equivalence relation*:
  - it is *reflexive*, *symmetric* and *transitive*
- Also *disjoint* properties:
  - :hasParent owl:propertyDisjointWith :hasSpouse .



# Summary: sameness and difference

- Individuals:
  - pairwise: [owl:sameAs](#), [owl:differentFrom](#)
  - groupwise difference: [owl:AllDifferent](#)
- Classes:
  - pairwise: [owl:equivalentClass](#), [owl:disjointWith](#)
  - groupwise difference: [owl:AllDisjointClasses](#)
- Properties:
  - pairwise: [equivalentProperty](#), [propertyDisjointWith](#)
  - groupwise difference: [owl:AllDisjointProperties](#)
- Membership in the groups:
  - [owl:distinctMembers](#) (*preferred*) or [owl:members](#)

