

INFO216: Advanced Modelling

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

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



Session S11 OWL

- Themes:
 - restriction classes
 - anatomy of OWL
 - more examples of *Turtle* (+ *Manchester Syntax*)
 - builds on *S06: RDFS-Plus*
 - *what and why?*
 - *basic OWL constructs*
 - *complex classes*
- Themes for S12:
 - rules, description logic, decision problems
 - perhaps the OWL API and reasoners



Readings

- Allemang & Hendler (2011):
Semantic Web for the Working Ontologist
 - chapter 11 (“Basic OWL”) and 12 (even more OWL!)
- 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... (S06)

- 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 players, a VolleyballTeam only 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!*



What does OWL offer? (S06)

- 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)
 - certain OWL ontologies are also logical systems
 - *description logic (DL)*
 - *OWL DL* has good computational behaviours
 - (*appearance of*) more powerful *entailments*



The Core OWL Concepts



Classes, properties, and individuals

- Web Ontology Language (OWL):
 - builds on RDF and RDFS (but not SKOS)
 - 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*
 - *in OWL DL, this is a requirement...*



Building blocks

- OWL 2 has three building blocks:
 - *entities*:
 - refer to real-world objects using IRIs
 - owl:NamedClass, owl:NamedIndividual
 - owl:ObjectProperty, owl:DatatypeProperty, owl:AnnotationProperty, owl:ObjectProperty
 - *axioms*:
 - basic statements the OWL ontology expresses
 - every triple in the RDF graph is an axiom
 - *expressions*:
 - combining simpler entities (classes, individuals, or properties) to define more complex ones
 - based on *constructors*



Building blocks

- OWL 2 has three building blocks:
 - *entities:*
 - refer to real-world objects using IRIs
 - owl:NamedClass, owl:NamedIndividual
 - owl:ObjectProperty, owl:DatatypeProperty, owl:AnnotationProperty, owl:ObjectProperty
 - *axioms:* ← *can be true or false!*
 - basic statements the OWL ontology expresses
 - every triple in the RDF graph is an axiom
 - *expressions:*
 - combining simpler entities (classes, individuals, or properties) to define more complex ones
 - based on *constructors*



Things and named individuals

- **owl:Thing**:
 - is equivalent to *rdfs:Resource*
- **owl:Nothing**
 - is the empty set
 - no resource has it as its *rdf:type*
- **owl:NamedIndividual**
 - is an *owl:Thing* with an IRI
 - defined in OWL2 DL



Named and constructed classes

- **owl:NamedClass** (with an **IRI**):
 - semantics are given by:
 - IRI-s, labels and other annotations
 - domain, range, subClassOf and other relationships
- **Constructed** (or **complex**) **owl:Class**:
 - built from existing classes, properties, individuals
 - which can be named *or anonymous*
 - constructed classes are *anonymous upon declaration*,
 - but can be *named* later
 - *unions, intersections* and *negations* of existing classes
 - *restrictions* on existing properties
 - *enumeration* of existing individuals



Object and datatype properties

- RDF triples: object is either a resource or a literal
 - OWL has two corresponding types of predicates
- **owl:ObjectProperty:**
 - rdfs:range (“verdiområde”) is an OWL-class of individuals
 - corresponds to RDF triples with a *resource* object
- **owl:DatatypeProperty:**
 - rdfs:range is an RDFS-datatype
 - corresponds to RDF triples with a *literal* object
- rdfs:domain (“definisjonsmengden”) for OWL properties is always an OWL-class of individuals



Annotation and ontology properties

- Annotation properties are used to annotate
 - *ontologies* (e.g., version)
 - *entities* (*classes, individuals, properties*) in the ont.
 - *axioms* (*triples*) in the ontology
 - for example: *rdfs:comment...*
- Ontology properties are used to manage ontologies
 - for example: *owl:imports...*
- They have *RDFS-semantics*
 - but no specific *description logic (DL)* semantics
 - often not “counted” alongside object and datatype properties



Summary: basic OWL types

- owl:Thing, owl:Nothing, owl:NamedIndividual
- owl:NamedClass, owl:Class
- owl:ObjectProperty, owl:DatatypeProperty
- owl:AnnotationProperty, owl:OntologyProperty



More precise properties in “RDFS Plus”

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



Sameness and difference in “RDFS Plus”

- 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](#)



Complex OWL classes



Enumeration classes

- An *enumeration class* is defined by exhaustively listing all its member individuals, e.g.:
 - cal:Season
a owl:Class ;
owl:oneOf (cal:Spring ... cal:Winter) .
- An enumeration class is *closed*
 - there are no other member individuals
 - ensured by using *RDF Collections*:
 - rdf:List, rdf:first, rdf:rest, rdf:nil
- **Does not imply** that the individuals are **distinct**
 - this must be stated explicitly



Enumeration classes

- An *enumeration class* is defined by exhaustively listing all its member individuals, e.g.:
 - cal:Season
 - a owl:Class ;*
 - owl:oneOf (cal:Spring ... cal:Winter) .*
- An enumeration class is *closed*
 - there are no other member individuals
 - ensured by using *RDF Collections*:
 - *rdf:List, rdf:first, rdf:rest, rdf:nil*
- **Does *not* imply** that the individuals are **distinct**
 - this must be stated explicitly



Other ways to write complex classes

- Why can also write:

```
cal:Season owl:equivalentClass [  
  a owl:Class ;  
  owl:oneOf ( cal:Spring ... cal:Winter ) ] .
```

or (a weaker claim):

```
cal:Season owl:subClassOf [  
  a owl:Class ;  
  owl:oneOf ( cal:Spring ... cal:Winter ) ] .
```

- Reason:
 - sometimes we just need *rdfs:subClassOf*
 - and it can be computationally more efficient
 - *owl:equivalentClass* entails two-way *rdfs:subClassOf*



Union classes

- A union class contains all the individuals in *either of two or more other classes*, e.g.,
 - fam:Parent
a owl:Class;
owl:unionOf (fam:Father fam:Mother) .
- Entailment rule:
 - if *C owl:equivalentClass owl:unionOf (C1... Cn)* then
 - *C1 rdfs:subClassOf C Cn rdfs:subClassOf C .*
- why not say just, e.g.,:
 - fam:Father rdfs:subClassOf fam:Parent .
 - fam:Mother rdfs:subClassOf fam:Parent .

?



Intersection classes

- An intersection class contains all the individuals in *all of* two or more other classes, e.g.
 - `uib:StudentAssistant`
 a `owl:Class`;
 `owl:intersectionOf (uib:Student uib:Teacher) .`
- Entailment rule:
 - if *C owl:equivalentClass owl:intersectionOf (C1... Cn)* then
 - *C rdfs:subClassOf C1 C rdfs:subClassOf Cn .*
- why not say, e.g.:
 - `uib:StudentAssistant rdfs:subClassOf uib:Student .`
 - `uib:StudentAssistant rdfs:subClassOf uib:Teacher .`



Complement classes

- A complement class contains all the individuals *that are not* in another class:
 - fam:Father
owl:intersectionOf (
fam:Parent
[owl:complementOf fam:Mother]
).



Closed World Assumption (CWA)

- Whenever something is not explicitly stated in the ontology, can we assume that the opposite is the case?
 - DBpedia only lists three James Dean movies – can we thus assume that he only played in three?
- Classical logic and many ICT languages assume so:
 - this is the “*Closed World Assumption*” (CWA)
- *In RDF and OWL, we do not assume that something is false just because it is not stated*
 - this is the “*Open World Assumption*” (OWA)



Negated properties (OWL 2)

- A negated property states that a triple with a particular subject, predicate and object would not correspond to a fact, e.g.,
 - [] `rdf:type owl:NegativePropertyAssertion ;`
 `owl:sourceIndividual :Bill ;`
 `owl:assertionProperty :hasWife ;`
 `owl:targetIndividual :Mary .`
 - means that it is not correct that “Bill has Mary as his wife”
 - an ontology with such a triple and its negation is inconsistent



Negated properties (OWL 2)

- A negated property states that a triple with a particular subject, predicate and object would not correspond to a fact, e.g.,
- ```
[rdf:type owl:NegativePropertyAssertion ;
 owl:sourceIndividual :Bill ;
 owl:assertionProperty :hasWife ;
 owl:targetIndividual :Mary .
```
- ```
[ rdf:type owl:NegativePropertyAssertion ;  
  owl:sourceIndividual      :Bill ;  
  owl:assertionProperty     :hasWife ;  
  owl:targetIndividual      :Mary      ] .
```
- The structure is similar to *triple reification*



Summary: complex classes

- owl:oneOf
- owl:unionOf
- owl:intersectionOf
- owl:complementOf (and the CWA)
- owl:NegativePropertyAssertion, owl:sourceIndividual, owl:assertionProperty, owl:targetIndividual



OWL restriction classes



Property value restrictions

- Defining a class by a particular value on one of its properties, e.g.:
 - ~~fam:Woman~~
 - ~~a owl:Restriction ;~~
 - ~~owl:onProperty fam:hasGender ;~~
 - ~~owl:hasValue fam:Female .~~
 - fam:Woman **owl:intersectionOf** (
 - fam:Person
 - [a owl:Restriction ;
 - owl:onProperty fam:hasGender ;
 - owl:hasValue fam:Female]) .



Existential property restrictions

- Defining a class by the existence of a relation (object property) to an individual in (another or the same) class, e.g.:
 - fam:Brother **owl:intersectionOf** (
 - fam:Male
 - [owl:Restriction ;
 - owl:onProperty fam:hasSibling ;
 - owl:someValuesFrom owl:Thing]) .
- **owl:someValuesFrom**: each individual in the defined class has *at least one* object property (given by owl:onProperty) to an individual in the other class (given by owl:someValuesFrom)



Existential property restrictions

- Defining a class by the existence of a relation (object property) to an individual in (another or the same) class, e.g.:
 - fam:Uncle **owl:intersectionOf** (
 - fam:Male
 - [a owl:Restriction ;
owl:onProperty fam:hasSibling ;
owl:someValuesFrom fam:Parent]) .
- **owl:someValuesFrom**: each individual in the defined class has *at least one* object property (given by owl:onProperty) to an individual in the other class (given by owl:someValuesFrom)



Universal property restrictions

- Defining a class by the necessity of a relation (object property) only to individuals in (another or the same) class, e.g.:

```
– fam:HappyFather owl:intersectionOf (  
    fam:Male  
    [ a owl:Restriction ;  
      owl:onProperty fam:hasChild ;  
      owl:allValuesFrom fam:HappyPerson ]  
    [ a owl:Restriction ;  
      owl:onProperty fam:hasChild ;  
      owl:someValuesFrom fam:HappyPerson ]  
  ) .
```



Property value restriction

- Using an anonymous property, e.g.:
 - fam:Orphan **owl:intersectionOf** (
 - fam:Person
 - [rdf:type owl:Restriction ;
 - owl:onProperty [owl:inverseOf :hasChild] ;
 - owl:allValuesFrom fam:Dead
 -]) .



Property self-reflexion (OWL2)

- Defining a class by a *Self* value on one of its properties, e.g.:

- fam:NarcissisticPerson

- rdf:type owl:Restriction ;

- owl:onProperty fam:loves ;

- owl:hasSelf "true"^^xsd:boolean .



Property value restriction

- Restrictions on data range, e.g.:
 - fam:personAge owl:equivalentClass
 - [rdf:type rdfs:Datatype;
 - owl:onDatatype xsd:integer;
 - owl:withRestrictions (
 - [xsd:minInclusive "0"^^xsd:integer]
 - [xsd:maxInclusive "150"^^xsd:integer])
 -].
 - :toddlerAge owl:equivalentClass
 - [rdf:type rdfs:Datatype;
 - owl:oneOf ("1"^^xsd:integer "2"^^xsd:integer)
 -].



Cardinality restriction

- Defining a class by the number of object values its individuals have for some property, e.g.:
 - music:Quartet owl:intersectionOf (
 music:Ensemble
 [a owl:Restriction ;
 owl:onProperty music:hasInstrument ;
 owl:cardinality 4]
)
).
- owl:cardinality gives the *exact cardinality*
owl:minCardinality gives the *least cardinality*
owl:maxCardinality gives the *greatest cardinality*



Qualified cardinality restriction (OWL2)

- Defining a class by the number of object values its individuals have *of a given class* for some property, e.g.:
 - `pol:Triumvirate owl:intersectionOf (`
 `pol:PoliticalLeadership`
 `[`
 `a owl:Restriction ;`
 `owl:onProperty pol:hasMember ;`
 `owl:qualifiedCardinality 3 ;`
 `owl:onClass pol:PoliticalLeader` `]`
 `) .`
- `owl:qualifiedCardinality` gives the *exact cardinality*
- `owl:minQualifiedCardinality` gives the *least cardinality*
- `owl:maxQualifiedCardinality` gives the *greatest cardinality*
- *Perhaps the most important addition in OWL2!*



Qualified cardinality restriction (OWL2)

- `music:StringQuartet owl:intersectionOf (`
 `music:MusicalQuartet`
 `[` `a owl:Class ;`
 `owl:onProperty music:hasInstrument ;`
 `owl:qualifiedCardinality "2" ;`
 `owl:onClass music:Violin]`
 `[` `a owl:Class ;`
 `owl:onProperty music:hasInstrument ;`
 `owl:qualifiedCardinality "1" ;`
 `owl:onClass music:Viola]`
 `[` `a owl:Class ;`
 `owl:onProperty music:hasInstrument ;`
 `owl:qualifiedCardinality "1" ;`
 `owl:onClass music:Cello]`
 `).`



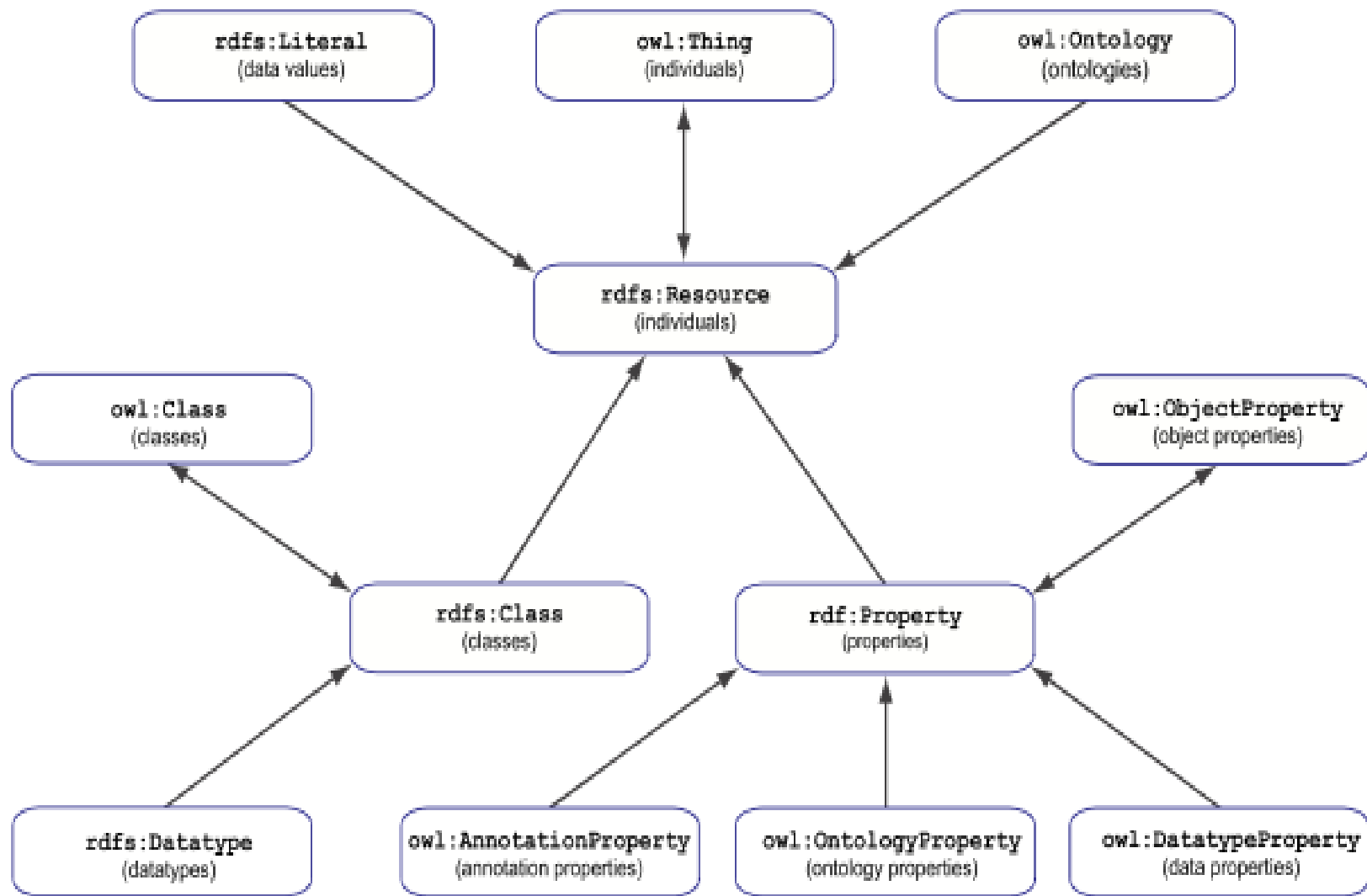
Summary: property restrictions

- owl:Restriction owl:onProperty
- owl:someValuesFrom, owl:allValuesFrom, owl:hasValue
- owl:cardinality, owl:minCardinality, owl:maxCardinality
- owl:onClass, owl:qualifiedCardinality,
owl:minQualifiedCardinality, owl:maxQualifiedCardinality



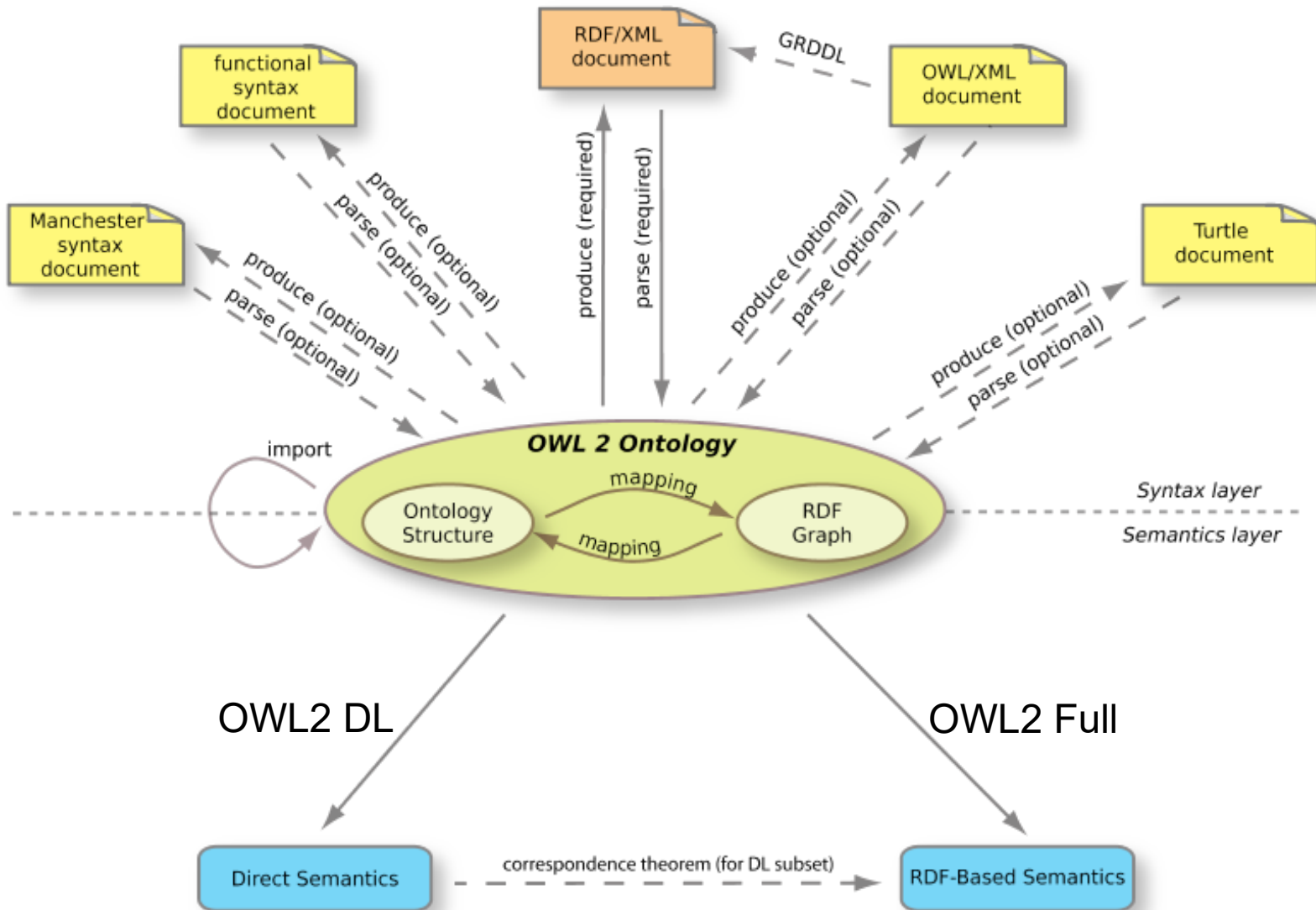
Anatomy of OWL





OWL versions

- **OWL “1”** (2002):
 - OWL Full – anything goes
 - OWL DL – fragment of OWL Full,
 - formal semantics through *description logic*
 - OWL Lite – simple fragment of OWL DL, not much used
- **OWL 2** (2008):
 - *backwards compatible with OWL “1”!*
 - OWL2 DL – fragment of OWL2 full, extension of OWL DL
 - **OWL2 DL** – has three further fragments:
 - OWL2 EL – quick reasoning, fragment of OWL2 DL
 - OWL2 RL – rule language, fragment of OWL2 DL
 - OWL LD – for Linked Data
 - OWL2 QL – query language, fragment of OWL2 DL



Summary of OWL terms

- owl:Ontology owl:Class owl:DatatypeProperty owl:ObjectProperty owl:NamedIndividual
- owl:Thing owl:Nothing owl:topObjectProperty owl:bottomObjectProperty owl:topDataProperty owl:bottomDataProperty
- owl:inverseOf owl:FunctionalProperty owl:InverseFunctionalProperty owl:TransitiveProperty owl:ReflexiveProperty owl:IrreflexiveProperty owl:SymmetricProperty owl:AsymmetricProperty owl:propertyChainAxiom
- owl:equivalentClass owl:disjointWith owl:equivalentProperty owl:propertyDisjointWith owl:sameAs owl:differentFrom owl:AllDifferent owl:AllDisjointClasses owl:AllDisjointProperties owl:members owl:distinctMembers owl:disjointUnionOf owl:NegativePropertyAssertion owl:assertionProperty owl:sourceIndividual owl:targetIndividual owl:targetValue
- owl:complementOf owl:intersectionOf owl:unionOf owl:oneOf owl:datatypeComplementOf owl:onDatatype owl:withRestrictions
- owl:Restriction owl:onProperty owl:onProperties owl:allValuesFrom owl:someValuesFrom owl:onDataRange owl:hasValue owl:hasSelf owl:cardinality owl:qualifiedCardinality owl:minCardinality owl:maxCardinality owl:onClass owl:minQualifiedCardinality owl:maxQualifiedCardinality
- owl:hasKey
- owl:annotatedProperty owl:annotatedSource owl:annotatedTarget owl:Annotation owl:AnnotationProperty owl:Axiom owl:imports owl:versionInfo owl:versionIRI owl:priorVersion owl:backwardCompatibleWith owl:OntologyProperty owl:incompatibleWith owl:deprecated owl:DeprecatedClass owl:DeprecatedProperty
- deprecated: owl:DataRange