Aalto University
School of Science

SeCo

# Web Ontology Language OWL
## Introduction to the Language

*Eero Hyvönen*
*Aalto University, Department of Computer Science*
*University of Helsinki, HELDIG-centre*
*Semantic Computing Research Group (SeCo), http://seco.cs.aalto.fi*
*eero.hyvonen@aalto.fi*

# Learning objective

- Learn Web Ontology Language OWL

# Basic Notions

## Axioms: statements/propositions

- Logical assertions about the domain of discourse (real world)
  - *E.g., "every eagle is a bird", "A chair has three or four legs"*
- Axioms are assumed to be always true (tautologies)

## Ontology

- An ontology is a set of axioms + data assertions (e.g., "Bob is an eagle")

## Consistency (inconsistency)

- There is (is not) a state of affairs that satisfies statements

# Main Components of an OWL Ontology

- **Classes** — Concepts (e.g. bird, eagle)
  - *Class definitions = "constructors"*
- **Properties** — Classes are defined using properties
  - ***Object properties*** — *(link individuals, e.g., married)*
  - ***Datatype properties*** — *(describe data values, e.g., name, age, date)*
  - ***Annotation properties*** — *(document ontology for human users)*
- **Individuals** — (e.g. Bob)

# Examples on the Next Slides are Taken from This OWL 2 Primer

W3C Recommendation

W3C®

## OWL 2 Web Ontology Language Primer (Second Edition)

### W3C Recommendation 11 December 2012

**This version:**
http://www.w3.org/TR/2012/REC-owl2-primer-20121211/
**Latest version (series 2):**
http://www.w3.org/TR/owl2-primer/
**Latest Recommendation:**
http://www.w3.org/TR/owl-primer
**Previous version:**
http://www.w3.org/TR/2012/PER-owl2-primer-20121018/
**Editors:**
Pascal Hitzler, Wright State University
Markus Krötzsch, University of Oxford
Bijan Parsia, University of Manchester
Peter F. Patel-Schneider, Nuance Communications
Sebastian Rudolph, FZI Research Center for Information Technology

Please refer to the **errata** for this document, which may include some normative corrections.

A color-coded version of this document showing changes made since the previous version is also available.

This document is also available in these non-normative formats: PDF version.

See also translations.



---

## Abstract

The OWL 2 Web Ontology Language, informally OWL 2, is an ontology language for the Semantic Web with formally defined meaning. OWL 2 ontologies provide classes, properties, individuals, and data values and are stored as Semantic Web documents. OWL 2 ontologies can be used along with information written in RDF, and OWL 2 ontologies themselves are primarily exchanged as RDF documents. The OWL 2 Document Overview describes the overall state of OWL 2, and should be read before other OWL 2 documents.

This primer provides an approachable introduction to OWL 2, including orientation for those coming from other disciplines, a running example showing how OWL 2 can be used to represent first simple information and then more complex information, how OWL 2 manages ontologies, and finally the distinctions between the various sublanguages of OWL 2.

# OWL Syntaxes

- RDF(S)-based syntaxes
- Specific OWL/XML schema
- More user-friendly notations
  - ***Functional-style syntax (for specifications)***
  - *Manchester syntax (for non-logicians)*

# Classes, hierarchies, individuals (1)

- Classes and individuals

Functional-Style Syntax

```
ClassAssertion( :Woman :Mary )
```

- Subclass relations

Functional-Style Syntax

```
SubClassOf( :Woman :Person )
```

- Class equivalence and disjointness
  - *Necessary and sufficient conditions*

Functional-Style Syntax

```
EquivalentClasses( :Person :Human )
```

Functional-Style Syntax

```
DisjointClasses( :Woman :Man )
```

- Individual equivalence and disjointness

Functional-Style Syntax

```
SameIndividual( :James :Jim )
```

Functional-Style Syntax

```
DifferentIndividuals( :John :Bill )
```

# Classes, hierarchies, individuals (2)

## Enumerations

- Defining a class by its members

**Functional-Style Syntax**

```
EquivalentClasses(
  :MyBirthdayGuests
  ObjectOneOf( :Bill :John :Mary)
)
```

## Complex classes

- Union

**Functional-Style Syntax**

```
EquivalentClasses(
  :Parent
  ObjectUnionOf( :Mother :Father )
)
```

- Intersection

**Functional-Style Syntax**

```
EquivalentClasses(
  :Mother
  ObjectIntersectionOf( :Woman :Parent )
)
```

- Complement
  - *Jack is a person but not a parent*

**Functional-Style Syntax**

```
ClassAssertion(
  ObjectIntersectionOf(
    :Person
    ObjectComplementOf( :Parent )
  )
  :Jack
)
```

# Quantification

## Class constructors based on quantified property values

- **Universal** restrictions
  *All children of a happy person are happy (may have no children, too)*
- **Existential** restrictions
  *A parent has at least one child*
- 
  *Using several conditions*

**Functional-Style Syntax**

```
EquivalentClasses(
  :HappyPerson
  ObjectAllValuesFrom( :hasChild :HappyPerson )
)
```

**Functional-Style Syntax**

```
EquivalentClasses(
  :Parent
  ObjectSomeValuesFrom( :hasChild :Person )
)
```

**Functional-Style Syntax**

```
EquivalentClasses(
  :HappyPerson
  ObjectIntersectionOf(
    ObjectAllValuesFrom( :hasChild :HappyPerson )
    ObjectSomeValuesFrom( :hasChild :HappyPerson )
  )
)
```

# Property Cardinality Restrictions

- Max cardinality
  *John has at most 4 children*

**Functional-Style Syntax**

```
ClassAssertion(
  ObjectMaxCardinality( 4 :hasChild :Parent )
  :John
)
```

- Min cardinality
  *John has at least 2 children*

**Functional-Style Syntax**

```
ClassAssertion(
  ObjectMinCardinality( 2 :hasChild :Parent )
  :John
)
```

- Exact cardinality
  *John has 3 children*

**Functional-Style Syntax**

```
ClassAssertion(
  ObjectExactCardinality( 3 :hasChild :Parent )
  :John
)
```

# More Property Restrictions

- Value restrictions

**Functional-Style Syntax**

```
EquivalentClasses(
  :JohnsChildren
  ObjectHasValue( :hasParent :John )
)
```

- Self restrictions

**Functional-Style Syntax**

```
EquivalentClasses(
  :NarcisticPerson
  ObjectHasSelf( :loves )
)
```

# Property Types

**Object properties**

- Relate individuals to other individuals
- 
```
:rents rdf:type owl:ObjectProperty ;
        rdfs:domain :Person ;
        rdfs:range :Apartment ;
        rdfs:subPropertyOf :livesIn .
```

**Datatype properties**

- Relate individuals to literals of certain datatypes
- E.g., :age, :name of an individual of class Person

**Annotation properties**

- For labeling, commenting, etc. for human consumption
- No logical meaning for the machine!

# Property Characteristics (1)

- Inverse properties

**Functional-Style Syntax**

```
InverseObjectProperties( :hasParent :hasChild )
```

- Symmetric and asymmetric properties

**Functional-Style Syntax**

```
SymmetricObjectProperty( :hasSpouse )
```

**Functional-Style Syntax**

```
AsymmetricObjectProperty( :hasChild )
```

- Disjointness

**Functional-Style Syntax**

```
DisjointObjectProperties( :hasParent :hasSpouse )
```

- Reflexive (self-relating) and irreflexive properties

**Functional-Style Syntax**

```
ReflexiveObjectProperty( :hasRelative )
```

**Functional-Style Syntax**

```
IrreflexiveObjectProperty( :parentOf )
```

# Property characteristics (2)

- Transitive properties
  - *:isPartOf*
- Functional properties
  - *:hasNumberOfRooms*
- Inverse-functional properties
  - *:hasSocialSecurityID*
- Subproperty relations and property chains
- Keys
  - *Identify uniquely individuals by values of key properties*

**Functional-Style Syntax**

```
TransitiveObjectProperty( :hasAncestor )
```

**Functional-Style Syntax**

```
FunctionalObjectProperty( :hasHusband )
```

**Functional-Style Syntax**

```
InverseFunctionalObjectProperty( :hasHusband )
```

**Functional-Style Syntax**

```
SubObjectPropertyOf(
  ObjectPropertyChain( :hasParent :hasParent )
  :hasGrandparent
)
```

```
HasKey( :RegisteredPatient :hasWaitingListN )


ClassAssertion
( :RegisteredPatient :ThisPatient )
DataPropertyAssertion
( :hasWaitingListN :ThisPatient "123-45-6789" )
```

# Individual Facts for Populating an Ontoogy

**Class and property assertions**

- As in RDF

**Negative assertions**

- Asserting that a relation does *not* hold

**Identity assertions**

- owl:sameAs, owl:differentFrom, owl:allDifferent

# Setting Property Values

- Object property values

**Functional-Style Syntax**

```
ObjectPropertyAssertion( :hasWife :John :Mary )
```

**Functional-Style Syntax**

```
NegativeObjectPropertyAssertion( :hasWife :Bill :Mary )
```

  - *Domain/range restrictions*

**Functional-Style Syntax**

```
ObjectPropertyDomain( :hasWife :Man )
ObjectPropertyRange( :hasWife :Woman )
```

- Datatype properties

**Functional-Style Syntax**

```
DataPropertyAssertion( :hasAge :John "51"^^xsd:integer )
```

**Functional-Style Syntax**

```
DataPropertyDomain( :hasAge :Person )
DataPropertyRange( :hasAge xsd:nonNegativeInteger )
```

# Examples: OWL syntaxes

https://www.w3.org/TR/owl2-primer/

You can see and learn different syntaxes on the Primer!

**Functional-Style Syntax**

```
ClassAssertion( :Person :Mary )
```

**RDF/XML Syntax**

```
<Person rdf:about="Mary"/>
```

**Turtle Syntax**

```
:Mary rdf:type :Person .
```

**Manchester Syntax**

```
Individual: Mary
  Types: Person
```

**OWL/XML Syntax**

```
<ClassAssertion>
   <Class IRI="Person"/>
   <NamedIndividual IRI="Mary"/>
</ClassAssertion>
```

**Functional-Style Syntax**

```
EquivalentClasses(
  :Mother
  ObjectIntersectionOf( :Woman :Parent )
)
```

**RDF/XML Syntax**

```
<owl:Class rdf:about="Mother">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="Woman"/>
        <owl:Class rdf:about="Parent"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

**Turtle Syntax**

```
:Mother  owl:equivalentClass  [
  rdf:type             owl:Class ;
  owl:intersectionOf   ( :Woman :Parent )
] .
```

**Manchester Syntax**

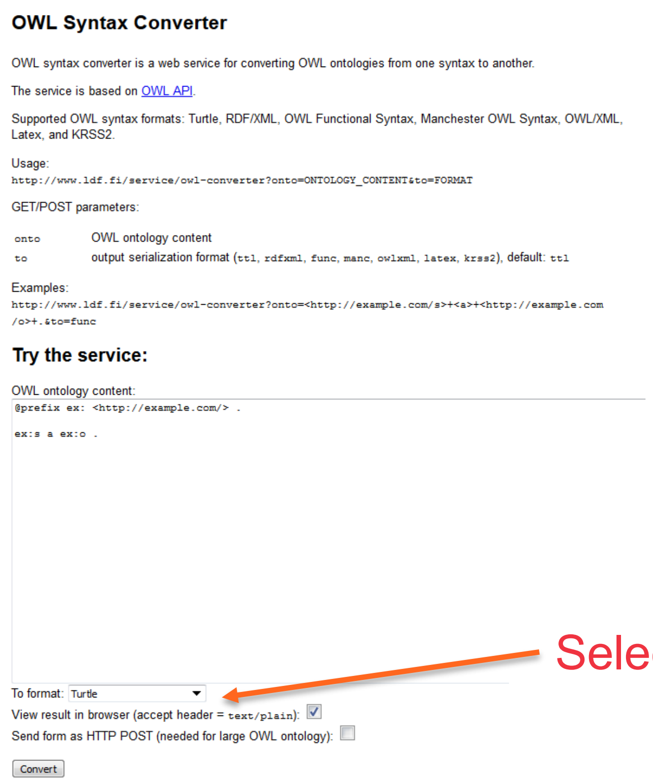
```
Class: Mother
  EquivalentTo: Woman and Parent
```

**OWL/XML Syntax**

```
<EquivalentClasses>
  <Class IRI="Mother"/>
  <ObjectIntersectionOf>
    <Class IRI="Woman"/>
    <Class IRI="Parent"/>
  </ObjectIntersectionOf>
</EquivalentClasses>
```

# OWL Syntax Converter

http://www.ldf.fi/service/owl-converter/



## OWL Syntax Converter

OWL syntax converter is a web service for converting OWL ontologies from one syntax to another.

The service is based on OWL API.

Supported OWL syntax formats: Turtle, RDF/XML, OWL Functional Syntax, Manchester OWL Syntax, OWL/XML, Latex, and KRSS2.

Usage:

```
http://www.ldf.fi/service/owl-converter?onto=ONTOLOGY_CONTENT&to=FORMAT
```

GET/POST parameters:

| | |
|---|---|
| `onto` | OWL ontology content |
| `to` | output serialization format (`ttl`, `rdfxml`, `func`, `manc`, `owlxml`, `latex`, `krss2`), default: `ttl` |

Examples:

```
http://www.ldf.fi/service/owl-converter?onto=<http://example.com/s>+<a>+<http://example.com
/o>+.&to=func
```

### Try the service:

OWL ontology content:

```
@prefix ex: <http://example.com/> .

ex:s a ex:o .
```

To format: Turtle

View result in browser (accept header = `text/plain`): ☑

Send form as HTTP POST (needed for large OWL ontology): ☐

Convert

Select format here

# Try the service:

OWL ontology content:

```
Prefix(:=<http://example.com/owl/families/>)
Ontology(<http://example.com/owl/families>
   SubClassOf(
     :ChildlessPerson
     ObjectIntersectionOf(
       :Person
       ObjectComplementOf(
         ObjectSomeValuesFrom(
           ObjectInverseOf( :hasParent )
           owl:Thing
         )
       )
     )
   )
)
```

To format: Turtle

View result in browser (accept header = `text/plain`): ☑

Send form as HTTP POST (needed for large OWL ontology): ☐

Convert

```
@prefix : <http://example.com/owl/families#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@base <http://example.com/owl/families> .

<http://example.com/owl/families> rdf:type owl:Ontology .


#################################################################
#
#    Object Properties
#
#################################################################


###  http://example.com/owl/families/hasParent

<http://example.com/owl/families/hasParent> rdf:type owl:ObjectProperty .





#################################################################
#
#    Classes
#
#################################################################


###  http://example.com/owl/families/ChildlessPerson

<http://example.com/owl/families/ChildlessPerson> rdf:type owl:Class ;

                                                  rdfs:subClassOf [ owl:intersectionOf ( <http://example.com/owl/families/Person>
                                                                                         [ rdf:type owl:Class ;
                                                                                           owl:complementOf [ rdf:type owl:Restriction ;
                                                                                                              owl:onProperty [ owl:inverseOf <http://example.com/owl/families/hasParent>
                                                                                                                             ] ;
                                                                                                              owl:someValuesFrom owl:Thing
                                                                                                            ]
                                                                                         ]
                                                                                       ) ;
                                                                    rdf:type owl:Class
                                                                  ] .
```

# Ontology Document Parts

**Like RDF documents**

- Namespace declarations
- Name (IRI) of the ontology
- Ontology-level metadata (versioning, comments, etc.)
- Importing other OWL documents
- **Definition of classes**
- **Definition of properties**
- **Definition of individuals**

See a full example in: https://www.w3.org/TR/owl2-primer/

# Example OWL ontology from OWL 2 Primer

```
Prefix(:=<http://example.com/owl/families/>)
Prefix(otherOnt:=<http://example.org/otherOntologies/families/>)
Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>)
Prefix(owl:=<http://www.w3.org/2002/07/owl#>)
Ontology(<http://example.com/owl/families>
  Import( <http://example.org/otherOntologies/families.owl> )

  Declaration( NamedIndividual( :John ) )
  Declaration( NamedIndividual( :Mary ) )
  Declaration( NamedIndividual( :Jim ) )
  Declaration( NamedIndividual( :James ) )
  Declaration( NamedIndividual( :Jack ) )
  Declaration( NamedIndividual( :Bill ) )
  Declaration( NamedIndividual( :Susan ) )
  Declaration( Class( :Person ) )
  AnnotationAssertion( rdfs:comment :Person "Represents the set of all people." )
  Declaration( Class( :Woman ) )
  Declaration( Class( :Parent ) )
  Declaration( Class( :Father ) )
  Declaration( Class( :Mother ) )
  Declaration( Class( :SocialRole ) )
  Declaration( Class( :Man ) )
  Declaration( Class( :Teenager ) )
  Declaration( Class( :ChildlessPerson ) )
  Declaration( Class( :Human ) )
  Declaration( Class( :Female ) )
  Declaration( Class( :HappyPerson ) )
  Declaration( Class( :JohnsChildren ) )
  Declaration( Class( :NarcisticPerson ) )
  Declaration( Class( :MyBirthdayGuests ) )
  Declaration( Class( :Dead ) )
  Declaration( Class( :Orphan ) )
  Declaration( Class( :Adult ) )
  Declaration( Class( :YoungChild ) )
```

```
Declaration( ObjectProperty( :hasWife ) )
Declaration( ObjectProperty( :hasChild ) )
Declaration( ObjectProperty( :hasDaughter ) )
Declaration( ObjectProperty( :loves ) )
Declaration( ObjectProperty( :hasSpouse ) )
Declaration( ObjectProperty( :hasGrandparent ) )
Declaration( ObjectProperty( :hasParent ) )
Declaration( ObjectProperty( :hasBrother ) )
Declaration( ObjectProperty( :hasUncle ) )
Declaration( ObjectProperty( :hasSon ) )
Declaration( ObjectProperty( :hasAncestor ) )
Declaration( ObjectProperty( :hasHusband ) )
Declaration( DataProperty( :hasAge ) )
Declaration( DataProperty( :hasSSN ) )
Declaration( Datatype( :personAge ) )
Declaration( Datatype( :majorAge ) )
Declaration( Datatype( :toddlerAge ) )

SubObjectPropertyOf( :hasWife :hasSpouse )
SubObjectPropertyOf(
  ObjectPropertyChain( :hasParent :hasParent )
  :hasGrandparent
)
SubObjectPropertyOf(
  ObjectPropertyChain( :hasFather :hasBrother )
  :hasUncle
)
SubObjectPropertyOf(
  :hasFather
  :hasParent
)
```

```
EquivalentObjectProperties( :hasChild otherOnt:child )
InverseObjectProperties( :hasParent :hasChild )
EquivalentDataProperties( :hasAge otherOnt:age )
DisjointObjectProperties( :hasSon :hasDaughter )
ObjectPropertyDomain( :hasWife :Man )
ObjectPropertyRange( :hasWife :Woman )
DataPropertyDomain( :hasAge :Person )
DataPropertyRange( :hasAge xsd:nonNegativeInteger )

SymmetricObjectProperty( :hasSpouse )
AsymmetricObjectProperty( :hasChild )
DisjointObjectProperties( :hasParent :hasSpouse )
ReflexiveObjectProperty( :hasRelative )
IrreflexiveObjectProperty( :parentOf )
FunctionalObjectProperty( :hasHusband )
InverseFunctionalObjectProperty( :hasHusband )
TransitiveObjectProperty( :hasAncestor )
FunctionalDataProperty( :hasAge )

SubClassOf( :Woman :Person )
SubClassOf( :Mother :Woman )
SubClassOf(
  :Grandfather
  ObjectIntersectionOf( :Man :Parent )
)
SubClassOf(
  :Teenager
  DataSomeValuesFrom( :hasAge
    DatatypeRestriction( xsd:integer
      xsd:minExclusive "12"^^xsd:integer
      xsd:maxInclusive "19"^^xsd:integer
    )
  )
)
```

```
SubClassOf(
    Annotation( rdfs:comment "States that every man is a person." )
    :Man
    :Person
  )
  SubClassOf(
    :Father
    ObjectIntersectionOf( :Man :Parent )
  )
  SubClassOf(
    :ChildlessPerson
    ObjectIntersectionOf(
      :Person
      ObjectComplementOf(
        ObjectSomeValuesFrom(
          ObjectInverseOf( :hasParent )
          owl:Thing
        )
      )
    )
  )
  SubClassOf(
    ObjectIntersectionOf(
      ObjectOneOf( :Mary :Bill :Meg )
      :Female
    )
    ObjectIntersectionOf(
      :Parent
      ObjectMaxCardinality( 1 :hasChild )
      ObjectAllValuesFrom( :hasChild :Female )
    )
  )
```

```
EquivalentClasses( :Person :Human )
EquivalentClasses(
  :Mother
  ObjectIntersectionOf( :Woman :Parent )
)
EquivalentClasses(
  :Parent
  ObjectUnionOf( :Mother :Father )
)
EquivalentClasses(
  :ChildlessPerson
  ObjectIntersectionOf(
    :Person
    ObjectComplementOf( :Parent )
  )
)
EquivalentClasses(
  :Parent
  ObjectSomeValuesFrom( :hasChild :Person )
)
EquivalentClasses(
  :HappyPerson
  ObjectIntersectionOf(
     ObjectAllValuesFrom( :hasChild :HappyPerson )
     ObjectSomeValuesFrom( :hasChild :HappyPerson )
  )
)
EquivalentClasses(
  :JohnsChildren
  ObjectHasValue( :hasParent :John )
)
EquivalentClasses(
  :NarcisticPerson
  ObjectHasSelf( :loves )
)
```

```
ObjectPropertyAssertion( :hasWife :John :Mary )
   NegativeObjectPropertyAssertion( :hasWife :Bill :Mary )
   NegativeObjectPropertyAssertion(
     :hasDaughter
     :Bill
     :Susan
   )
   DataPropertyAssertion( :hasAge :John "51"^^xsd:integer )
   NegativeDataPropertyAssertion( :hasAge :Jack "53"^^xsd:integer )

   SameIndividual( :John :Jack )
   SameIndividual( :John otherOnt:JohnBrown )
   SameIndividual( :Mary otherOnt:MaryBrown )
   DifferentIndividuals( :John :Bill )
 )

EquivalentClasses(
     :MyBirthdayGuests
     ObjectOneOf( :Bill :John :Mary)
   )

   ClassAssertion( :Person :Mary )
   ClassAssertion( :Woman :Mary )
   ClassAssertion(
     ObjectIntersectionOf(
       :Person
       ObjectComplementOf( :Parent )
     )
     :Jack
   )
   ClassAssertion(
     ObjectMaxCardinality( 4 :hasChild :Parent )
     :John
   )
   ClassAssertion(
     ObjectMinCardinality( 2 :hasChild :Parent )
     :John
   )
```

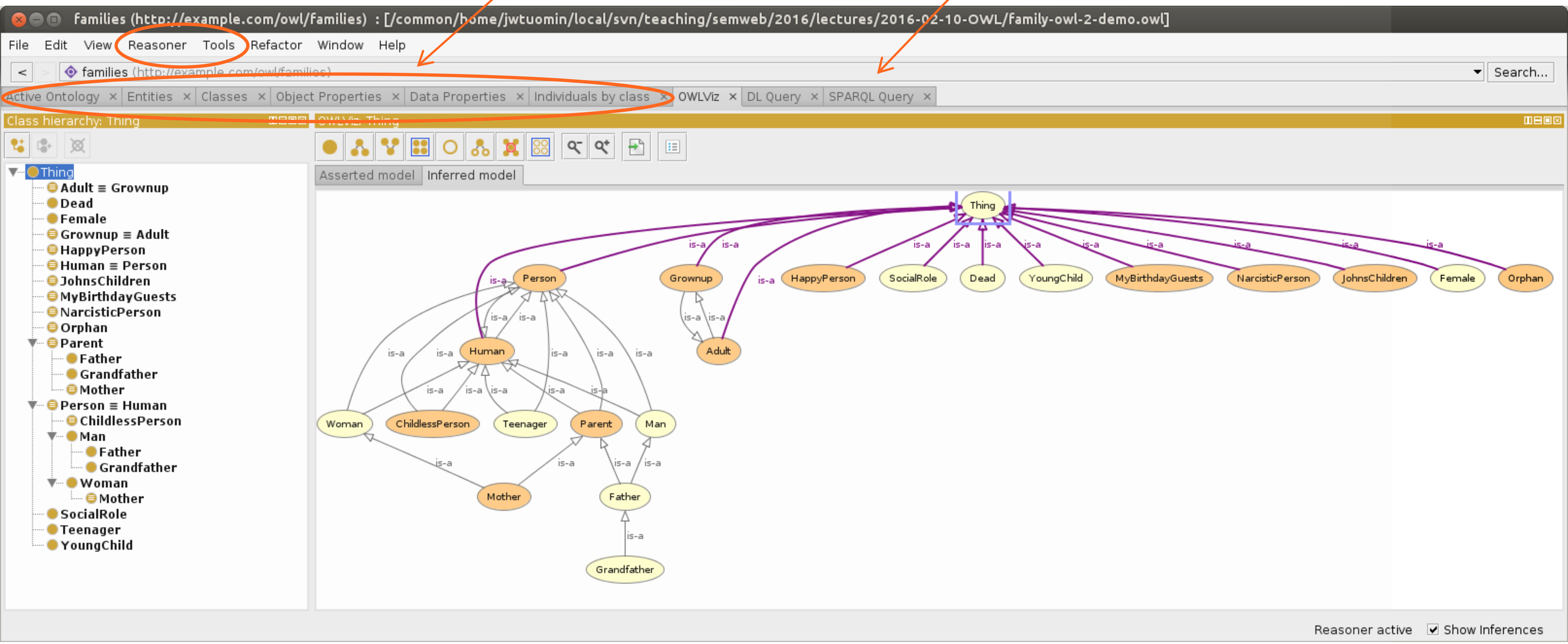
```
  ClassAssertion(
    ObjectExactCardinality( 3 :hasChild :Parent )
    :John
  )
  ClassAssertion(
    ObjectExactCardinality( 5 :hasChild )
    :John
  )
  ClassAssertion( :Father :John )
  ClassAssertion( :SocialRole :Father )
  ObjectPropertyAssertion( :hasWife :John :Mary )
  NegativeObjectPropertyAssertion( :hasWife :Bill :Mary )
  NegativeObjectPropertyAssertion(
    :hasDaughter
    :Bill
    :Susan
  )
  DataPropertyAssertion( :hasAge :John "51"^^xsd:integer )
  NegativeDataPropertyAssertion( :hasAge :Jack "53"^^xsd:integer )

  SameIndividual( :John :Jack )
  SameIndividual( :John otherOnt:JohnBrown )
  SameIndividual( :Mary otherOnt:MaryBrown )
  DifferentIndividuals( :John :Bill )
)
```

# Example ontology in Protégé editor

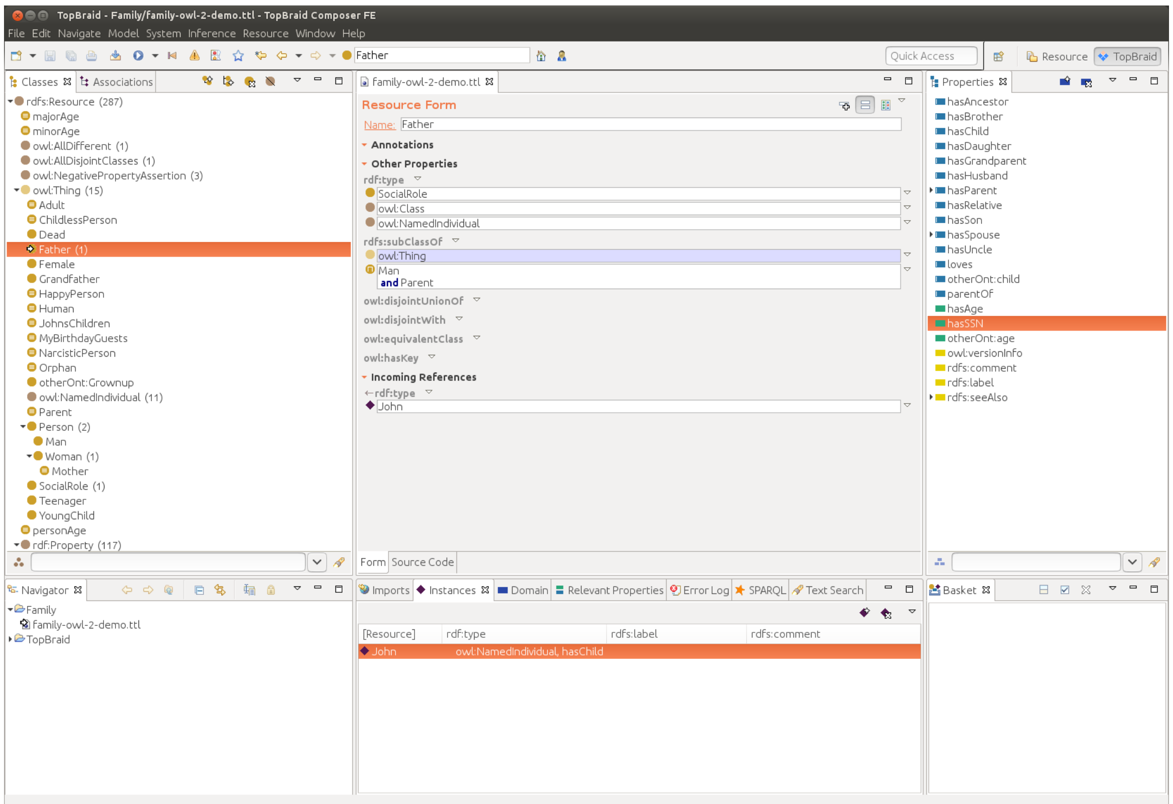
OWL constructs

Plugins



# ...and in TopBraid Composer

- Commercial product with a free edition option
- SPIN rules for reasoning, e.g., OWL RL support available
- Includes possibility for SPARQL querying



# Two Logical Assumptions of OWL

# Closed World Assumption (CWA)

**Closed-world assumption: what is not known to be true is false**

- Assumes that everything is known, and data not stated is assumed to be false
- Very powerful and often useful assumption
  - *In use in, e.g., databases*
- The notion of defaults leads to nonmonotonic logics

**OWL adopts the open-world assumption:**

- CWA is not made
- On the huge and only partially knowable WWW, this is a correct assumption
- Lots of additional assertions may be needed for closing data
  - *For stating what facts are not true in addition to what is true*

# Unique Names Assumption (UNA)

- Typical database applications assume that individuals with same/different names are indeed same/different individuals
- OWL follows the usual logical paradigm where this is **not** the case
  - *Plausible on the WWW where multiple IDs exist*
- One may want to indicate portions of the ontology for which the assumption does or does not hold
  - *In many cases UNA is useful*
  - *Lots of additional assertions may then be needed for stating what objects are different and what are the same*

# OWL Profiles:
## Trade-off between Expressive Power and Efficient Reasoning

Aalto University
School of Science

Department of
Computer Science

SeCo

# Compatibility of OWL with RDF(S): OWL Full

## OWL 2 Full

- All OWL features added on top of RDF(S)
  - *Allows, e.g., redefining the meaning of RDF(S) and OWL primitives*
- Advantages
  - *Fully upward compatible with RDF*
    - Any RDF document is an OWL 2 Full document
    - Any RDF(S) conclusion is an OWL 2 Full conclusion
  - *RDF-based semantics*
- Disadvantages
  - *Undecidable, as RDFS already has some very powerful modeling primitives*
  - *Complete and efficient reasoning not possible*

# Compatibility of OWL 2 with RDF(S): OWL DL

**OWL 2 DL (Description Logic)**

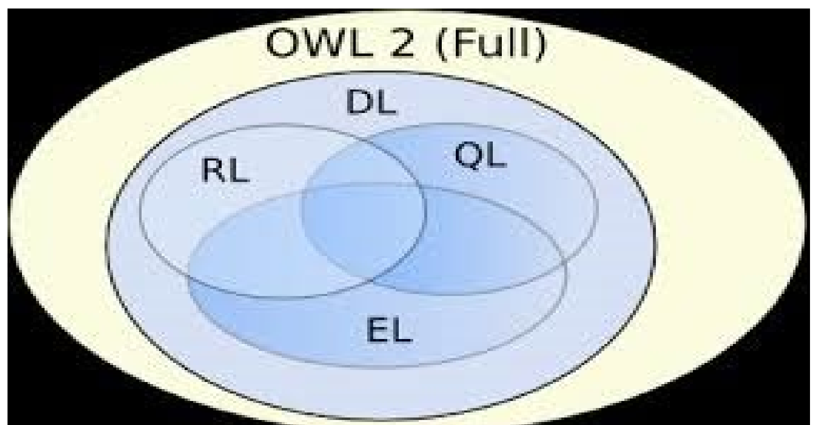
- Restricted form of OWL Full for which decidable, efficient support for reasoning is possible
  - *OWL 2 primitives cannot be applied to themselves*
  - *Only classes of non-literal resources considered*
  - *Strict separation between datatype and object properties*
  - *Strict separation between an individual, a class, or a property*
    - Using ”punning” the same name may be used for different purposes, but treated as different views on the same IRI, interpreted semantically as if they were distinct
- Direct semantics, based on Description logics (Terminology logics)
  - *Subsets of predicate logic*
  - *But also RDF-based semantics can be applied to OWL 2 DL ontologies*
- Reasoning engines are available for DL
  - *Pellet, FaCT, RACER, HermiT*

# OWL 2 profiles

**OWL 2 DL includes three specific profiles for different use cases**

- OWL 2 EL
- OWL 2 QL
- OWL 2 RL

**Each profile includes a subset of OWL DL features**



# OWL 2 EL ("$\mathcal{EL}$ description logics")

- Good for ontologies with lots of classes and/or properties
- Polynomial complexity of standard inference types: satisfiability, classification, instance checking
- Used for large scale class ontologies, e.g., Snomed CT

- Limitations include:
  - *Negation and disjunction not supported*
  - *Universal quantification on properties*
    - *E.g., "all children of a rich person are rich" cannot be stated*
  - *All kinds of role inverses are not available*
    - *E.g., parentOf and childOf cannot be stated as inverses*

# OWL 2 QL (“query language”)

- Good for querying large numbers of individuals
- Relational Query Languages (conjunctive queries)
  - *Can be implemented efficiently using relational databases*

- Limitations include:
  - *Existential quantification of roles to a class expression*
    - *E.g., it can be stated that every person has a parent, but not that every person has a female parent*
  - *Property chain axioms and equality are not supported*

# OWL 2 RL ("rule language")

- Good for rule-based reasoning, database focus
- Can be implemented using logic programming
  - *First-order implications: IF certain triples exist THEN add additional triples*
    - See partial axiomatization of the OWL 2 RDF-based semantics as rules in the OWL 2 Profiles specification (Section 4.3)

- Limitations include:
  - *Statements where the existence of an individual enforces the existence of another individual*
    - E.g., the statement "every person has a parent" is not expressible
  - *Restricts class axioms asymmetrically*
    - *Constructs for a subclass cannot necessarily be used as a superclass*

- An implementation of OWL RL is used in the exercises for reasoning

# Summary

- OWL 2 extends RDF(S)
- Multiple corresponding syntaxes
  - *RDF notations (RDF/XML, Turtle, etc.), OWL/XML, Functional-style, Manchester syntax*
- Two corresponding semantics: Direct and RDF-based
- Two OWL versions
  - *OWL 2 DL: Direct semantics based on Description Logics*
    - *Decidable, efficient reasoning*
    - *Not fully upward compatible with RDF(S)*
  - *OWL 2 Full: Based on RDF semantics*
    - *Undecidable, partial reasoning possible*
    - *Upward compatible with RDF(S)*
- Three more efficient DL profiles for different purposes
  - *OWL 2 EL, OWL 2 QL, and OWL 2 RL*

# References and Further Information

**Namespace IRI of OWL contains the specification in RDF for 1) classes and 2) properties**

- http://www.w3.org/2002/07/owl#

**Theory**


- M. Krötzsch, F. Simančík, I. Horrocks: Description Logic Primer. 2013.
- P. Hitzler, M. Krötzsch, S. Rudolph: Foundations of Semantic Web Technologies. CRC Press, 2009.


**Reasoners**

- http://semanticweb.org/wiki/Category_Reasoner