



OWL Tutorial

LD4P RareMat / ARTFrame Meeting
Columbia University
January 11-12, 2018



Outline

- Goals
- RDF, RDFS, and OWL
- Inferencing
- OWL serializations
- OWL validation
- Demo: Building an OWL ontology in Protégé
- Brief consideration of hand-coding as an alternative to Protégé
- Exercise in building an OWL file
- Discussion and review



Goals

- Principal goal: Given an ontology (diagram and list of terms), create an OWL file.
 - Modeling tutorial focused on conceptual/semantic part of building a model.
 - Here we focus on formal specification of the model in OWL.
- Understand
 - Basic structure of RDF, RDFS, and OWL
 - Constraints and axioms
 - Inferencing
 - Serializations
- Develop familiarity with tools for writing and validating OWL files.



From Ontology to OWL

- **Data model**
 - An abstract conceptual model that identifies things in the real world, their classifications, properties, and relationships to one another.
 - Implementation-neutral: i.e., the model will be the same regardless of whether it is to be implemented as an ontology, a relational database, etc.
 - Most clearly represented as a diagram.
- **Ontology**
 - One type of formal specification of the concepts (terms) in the data model, including names, definitions, and their interrelationships, at an abstract level.
- **RDFS and OWL languages for encoding an ontology (OWL tutorial).**
- **RDFS and OWL language serializations (OWL tutorial).**



From Ontology to OWL

- **Data model**
 - An abstract conceptual model that identifies things in the real world, their classifications, properties, and relationships to one another.
 - Implementation-neutral: i.e., the model will be the same regardless of whether it is to be implemented as an ontology, a relational database, etc.
 - Most clearly represented as a diagram.
- **Ontology**
 - One type of formal specification of the concepts (terms) in the data model, including names, definitions, and their interrelationships, at an abstract level.
- **RDFS and OWL languages: implementations of an ontology**
- **RDFS and OWL language syntaxes (OWL tutorial)**



From Ontology to OWL

- Boundaries are fuzzy in practice - though not in theory.
- The purest way to do this would be to develop the ontology and let it dictate the most suitable implementation language.
- Conversely, in our case, we've made the prior choice of an OWL implementation, which determines the types of concepts we include in our ontology.
- We'll see examples in the following.
- This mixing is natural because it's not always easy to maintain strict boundaries and sequencing.



From RDF to RDFS to OWL

- Definition: languages for encoding the semantic relationships between items of data so that these relationships can be interpreted computationally.
- RDFS is built on RDF.
- OWL is built on RDFS.
- Progression of expressivity and power.
- The more expressive, the more powerful in terms of inferencing (entailments).
- Inferencing allows machines (reasoners) to compute new information from existing data.
- An exciting possibility!



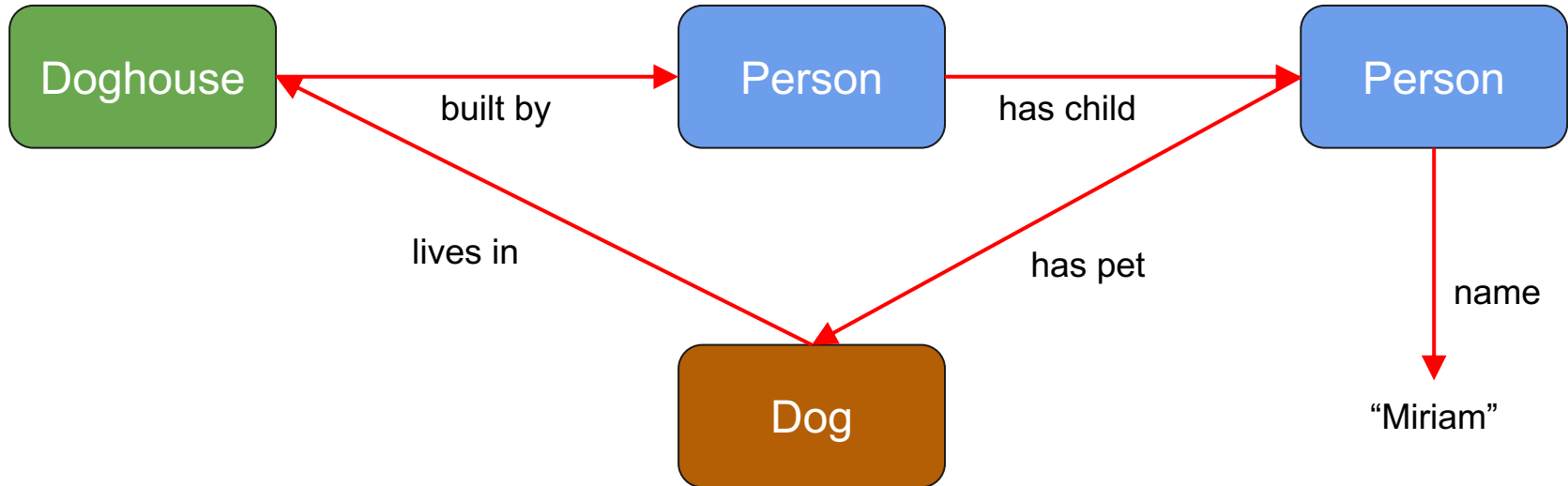
RDF



What is RDF?

- RDF = Resource Description Framework
- A standard model for data interchange on the web.
- Types of resources: individuals, properties, literals.
 - Literals may be typed: string, date, integer, decimal.
 - String literals may have a designated language.
- Defines the basic data structures on which RDFS and OWL are built.
- The fundamental RDF data structure is a **triple**.
 - Subject - predicate - object.
 - Two *nodes* connected by an *edge*.
- A set of one or more triples forms a *directed labeled graph*.

Directed Labeled Graph





Directed Labeled Graphs

- **Nodes:** Subject and object
 - Includes both resources (things) and literals.
- **Edges:** Properties
 - Connect two nodes
 - Labeled
 - Single direction (directed)
- Subjects must be things, not literals.
- Predicates (properties) are also things, not literals.
- Objects can be either things or literals.
- Triples are linked together into graphs.
- *Because literals cannot be subjects, they are dead ends in the graph, which is yet another reason not to like them.*



A Note on Blank Nodes

- Blank nodes are nodes with no URI.
- Represented with an identifier *scoped to the data set or document*.
- I.e., the same identifier doesn't identify the same node across data sets.
- Thus identifiers have no reuse potential and cannot link data from different data sets.
- Blank nodes can cause trouble in applications, so we avoid them in our data, and create a URI for everything.
- These are *application-level considerations*.
 - An *ontology* does not specify the use or non-use of blank nodes.



Familiar RDF Terms

- `rdf:Property`
- `rdf:type`
- `rdf:value`
- Others: ignore for now
- Note that Class is **not** defined in RDF.
 - The concept of Property is more fundamental than Class.
 - A graph can exist without classes: individuals do not have to be assigned to classes.
 - But it cannot exist without properties (edges).



RDFS



What is RDFS?

- RDFS = RDF Schema
- Extends RDF with additional vocabulary.
- Types of resources: individuals, classes, properties, literals.



Most Common RDFS Terms

- Classes
 - Resource
 - Literal
 - Class
 - NamedIndividual
- Properties
 - domain
 - range
 - subClassOf
 - subPropertyOf
- Human-readable properties (not computationally significant)
 - label
 - comment
 - seeAlso



RDFS Constraints and Entailments

- Domain constraints
 - Property P1 has domain of class C1
 - thing1 P thing2
 - Inference: thing1 has type C1
 - Example: property speaksLanguage has domain Person
- Range constraints
 - Property P2 has range of class C2
 - thing1 P2 thing2
 - Inference: thing2 has type C2
 - Example: property speaksLanguage has range Language



RDFS Entailments, continued

- subClassOf
 - Class C1 is a subclass of class C2
 - thing1 has type C1
 - Inference: thing1 has type C2
- subPropertyOf
 - Property P1 is a subproperty of property P2
 - thing1 P1 thing2
 - Inference: thing1 P2 thing2



Effect of Entailments

- In logic, inferences can lead to contradictions:
 - Socrates is a man.
 - All men are mortal.
 - Inference: Socrates is mortal.
 - **Socrates is not mortal. CONTRADICTION!**
- Not in RDFS, because negations cannot be expressed.
- Entailments are even more interesting in OWL because there are many more.



OWL



What is OWL?

- OWL = Web Ontology Language
- Built on RDFS.
- Types of resources: individuals, classes, properties, literals.
- Different flavors of OWL; we will not deal with these.
- Uses all the RDFS constraints: domain, range, subclassOf, subPropertyOf.
- Significantly more expressive than RDFS due to many additional properties and classes that enable complex inferencing.



OWL Individuals

- sameAs
 - Perhaps the most important OWL predicate
 - Critical in entity reconciliation
- To block reconciliation of individuals
 - differentFrom
 - Semantically, though not formally, a negation of sameAs
 - allDifferent



OWL Classes

- OWL defines owl:Class than using rdfs:Class.
 - Subclass of rdfs:Class.
- You can say more things about OWL classes than RDFS classes.
 - Membership constraints
 - Set operations
 - Property restrictions: restrictions on a class *relative to* a property

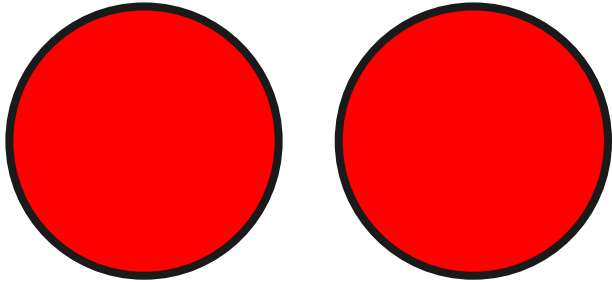


OWL Class Membership Constraints

- Define classes with a finite, enumerable set of members
 - E.g., the class `Continent` has 5 specific members
 - Useful in defining controlled vocabularies
- Membership constraints with set predicates
 - Disjoint classes
 - Equivalent classes



Disjointness



Vegetarians and meat-eaters are disjoint classes.

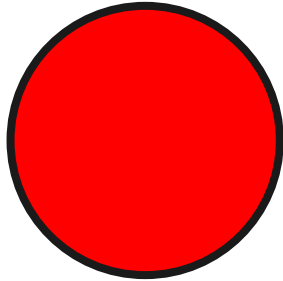
Useful to indirectly express non-membership.

Miriam is a vegetarian.

Inference: Miriam is not a meat-eater.



Equivalence



Two classes have the same members.

Useful to align classes from different ontologies.

E.g., foaf:Person is equivalent to schema:Person.

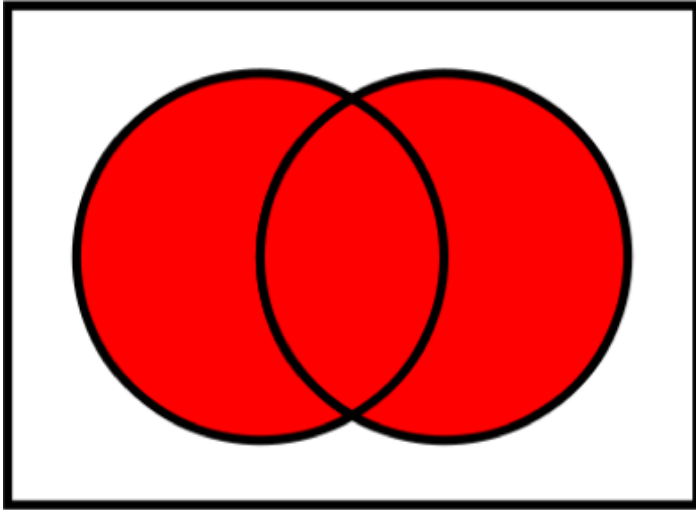


OWL Set Operations

- Define classes
- Anonymous (no URI)
- Ad hoc: You want to refer to it once but not persist or reuse it.
- Predicates
 - `unionOf`
 - `intersectionOf`
 - `complementOf`



Union



- Define a class: elements that are in both sets.
- Use where there is no logical or persistent superclass.
- Example: the class of PTA members is the union of the class of parents and the class of teachers.
- Useful to define domains and ranges: the domain of property “attends PTA meetings”.

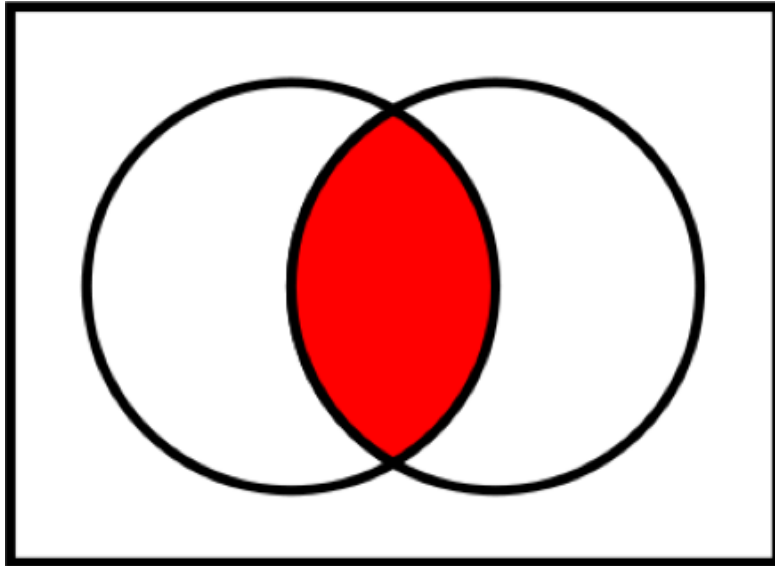


Union, Continued

- Could be achieved through subclassing.
- E.g., a Primate class is the superclass of humans, monkeys, and apes.
- But sometimes there is no reasonable superclass, or not one that I want to persist or reuse in other contexts.
- So I can use the union.
- RDFS can only express domains and ranges that are intersections of two classes.



Intersection

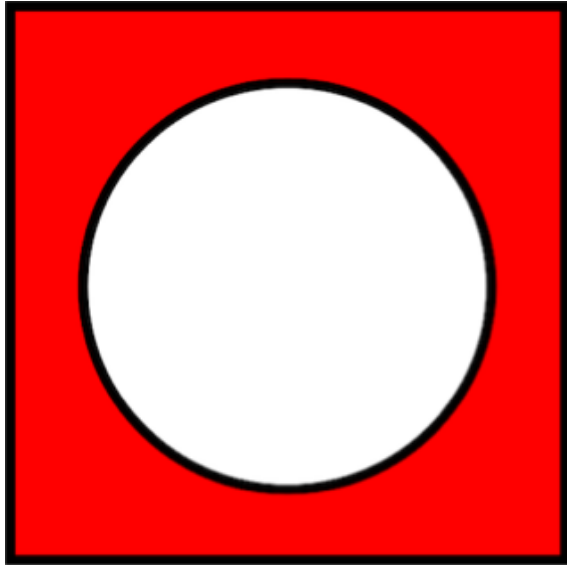


Define a class: elements that are members of both sets.

Example: the class of people who speak both French and English.



Absolute Complement



Define a class: all elements that are not members of a specific set.

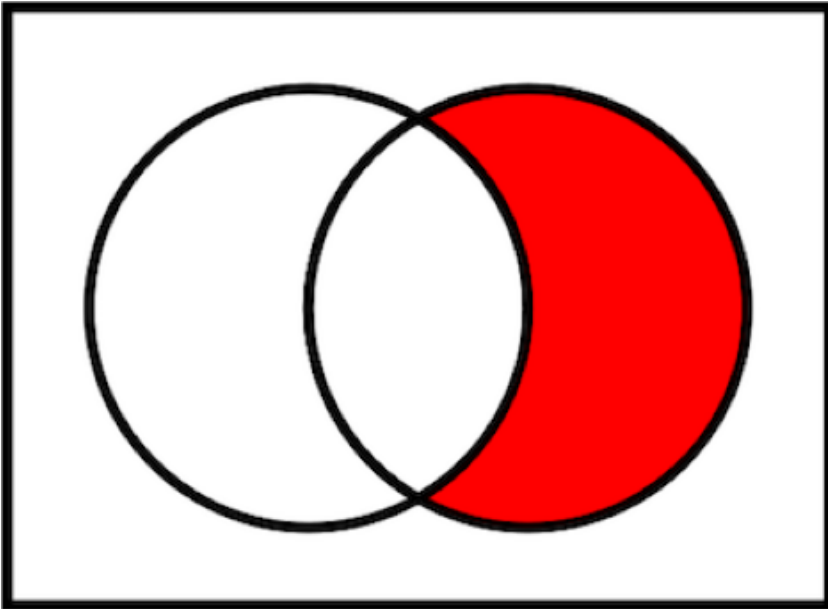
The area in red is the complement of the area in white.

Example: White circle: meat. Red area: everything that is not meat, including chairs.

Relative complement (next slide) more useful than *absolute* complement.



Relative Complement



The area in red is the complement of the white circle *relative to* the enclosing rectangle.

Example: Enclosing rectangle = edible things. White circle = meat. Red = edible things that are not meat.

The class of things vegetarians eat is the area in red.



Property Restrictions: Values

- RDFS range is absolute.
 - I.e., property P has range C regardless of the type of the subject..
- An OWL property restriction allows us to say: for members of Class C1, all the values of Property P are members of Class C2.
 - Example: For class Person, all the values of property hasParent are members of the class Person.
 - Whereas we cannot say the *range* of hasParent is Person, because animals also have parents.



Property Restrictions: Cardinality

- For members of class C, the property P has
 - Exactly n values, or
 - A minimum of n values, or
 - A maximum of n values
- Example: For the class of homeowners, the property “owns home” has a minimum cardinality of 1.
- Whereas for the predicate itself, there is no minimum number of values.



OWL Properties

- RDFS allows properties that take both things and literals as objects.
- In OWL, all properties must be one of the following types:
 - ObjectProperty - objects are things
 - hasChild
 - DatatypeProperty - objects are literals
 - name
 - AnnotationProperty - human-readable only, machines ignore
 - rdfs:label, rdfs:comment, rdfs:seeAlso
 - OntologyProperty - subjects and objects are ontologies
- If Thing1 hasChild Thing2, then all the values of hasChild are things rather than literals.
- Additional property types which are extremely useful for inferencing - next slides.



Inverse Properties

- Predicate owl:inverseOf
 - P1 and P2 are inverses
 - A P1 B
 - Inference: B P2 A
- Example: hasParent / hasChild
- Useful to make assertions about an object not in your namespace.
 - A is in my namespace - I can publish assertions about it.
 - B is not in my namespace so I can't publish assertions about it.
 - But I can use the fact that hasParent and hasChild are inverses.
 - So I assert A hasParent B and I have effectively asserted that B hasChild A.
- Can also simplify queries.



Subproperties and Equivalent Properties

- owl:subPropertyOf
 - P1 owl:subProperty of P2
 - A P1 B
 - Inference: A P2 B
 - Example:
 - hasParent owl:subPropertyOf hasRelative
 - A hasParent B
 - Inference: A hasRelative B
- owl:equivalentProperty
- Like owl:subClassOf and owl:equivalentClass, these properties enable data alignment across ontologies.



Functional and InverseFunctional Properties

- Extremely powerful because they create **sameAs** (reconciliation) entailments.
- FunctionalProperty: any individual has at most **one unique value** of Property P.
 - Example: hasBiologicalMother
 - Inference: If A hasBiologicalMother B, and A hasBiologicalMother C, then B sameAs C
- InverseFunctionalProperty: any individual that is the object of property P can be related to at most **one unique subject**.
 - Example: biologicalMotherOf
 - Inference: if A biologicalMotherOf B, and C biologicalMother of B, then A sameAs C



Some Additional OWL Property Types

- TransitiveProperty
 - Example: hasAncestor
 - Inference: If A hasAncestor B, and B hasAncestor C, then A hasAncestor C
- SymmetricProperty
 - Example: hasSibling
 - Inference: If A hasSibling B, then B hasSibling A
- AsymmetricProperty
 - Example: hasChild
 - Inference: if A hasChild B, and B hasChild C, then A differentFrom C
- IrreflexiveProperty
 - Example: hasParent
 - Inference: If A hasParent B, then A differentFrom B



Property Chains: Shortcut Properties

- Family example: We found that a property `hasGrandmother` is redundant because the relationship can be inferred from `hasParent` + `hasMother`.
- However, it may be convenient to have a property `hasGrandmother`.
 - E.g., simplifies queries.
- A property chain states formally the semantics of `hasGrandmother`.



Property Chain Inferencing

- Infer the shortcut property
 - A hasParent B
 - B hasMother C
 - A hasGrandmother C
- Inference from the shortcut property
 - A hasGrandmother C
 - Inference: there is some individual C such that
 - A hasParent C and
 - C hasMother B



Inferencing in the Open World

- **Closed world assumption:** if statement S is not part of our data set, then it is false.
- That is, our data is a closed set that cannot be added to.
- In other words, everything that is true is known.
- **Open world assumption:** if statement S is not part of our data set, it may or may not be true.
- Thus the open world limits inferencing capabilities.
- RDF, RDFS, and OWL all use the open world assumption.
- The open world assumption applies naturally to linked data: we cannot treat our data as a closed set, because it is meant to connect with other data that is currently unknown.



Conflicting Data in OWL

- Some OWL entailments can lead to contradictory / invalid data
- Example 1:
 - `hasBiologicalMother` is a `FunctionalProperty`
 - A `hasBiologicalMother` B
 - A `hasBiologicalMother` C
 - Inference: B `sameAs` C
 - **B `differentFrom` C. Contradiction!**
- Example 2:
 - Class A and B are disjoint
 - P has type A
 - **P has type C. Contradiction!**



Conflicting Data in OWL vs RDFS

- As we said, because RDFS is
 - less expressive than OWL (defines fewer predicates and classes),
 - entailments are more limited, thus
 - there is no invalid / contradictory / conflicting data
- Both a strength and a weakness of OWL.
 - OWL entailments *help find* bad data.
 - OWL entailments *result in* bad data.



Inferencing: Don't Overconstrain

- Constraints are fun, but
- Use them judiciously.
- Overuse of domains and ranges can create closed universes with a wave of entailments
 - E.g., prov, cidoc-crm
 - LC is removing many domains and ranges from BIBFRAME
- This limits reuse potential.
- May create contradictions in your data.
- And inferencing is *computationally extremely expensive*.



Machine-Generated Data: Two Approaches

- Two approaches to machine-generated data: Machine learning
 - Data-driven.
 - Requires large amounts of data.
 - Accuracy depends on quantity of data.
 - Almost anything can be learned from the right data and learning algorithms.
 - Examples:
 - Language translation and understanding
 - Medical diagnoses
 - Self-driving cars
- Inferencing
 - Logic-based.
 - Does not require large amounts of data.
 - Accuracy does not depend on quantity of data.
 - Limited to RDFS and OWL language constructs.



Power vs Simplicity

- We've seen how much more can be said in OWL than RDFS, and the rich inferencing it enables.
- However, we've also seen that OWL is vastly more complex than RDFS.
- And RDFS does not allow data contradictions.
- Some ontology developers choose RDFS over OWL for this reason.
 - E.g., in its latest version, the Web Annotation ontology (now a W3C standard) is an RDFS rather than an OWL ontology.
- Food for thought: If entailment is so computationally expensive that it is rarely exploited, ***what is the value (today) of OWL's increased complexity?***



Break



Serialization (Syntax)



Serialization

Semantic equivalents - only differ syntactically... but you find and develop lots of feelings about each :)

- RDFS/XML - .rdf
- Turtle - .ttl
- JSON-LD - .jsonld
- N-Triples - .nt (but not used in ontology development, only for bulk instance data, so maybe skip or just mention briefly as another serialization)



Serialization: RDF/XML

```
<!-- http://www.ontotext.com/proton/protontop#hasSibling -->  
  
<owl:ObjectProperty rdf:about="http://www.ontotext.com/proton/protontop#hasSibling">  
  <rdfs:subPropertyOf rdf:resource="http://www.ontotext.com/proton/protontop#hasRelative"/>  
  <rdfs:comment xml:lang="en">Relation between a person and his/her siblings - brother or sister.</rdfs:comment>  
  <rdfs:label xml:lang="en">has Sibling</rdfs:label>  
</owl:ObjectProperty>
```



Serialization: Turtle (Terse RDF Triple Language)

```
### http://www.ontotext.com/proton/protontop#hasSibling
ptop:hasSibling rdf:type owl:ObjectProperty ;
                 rdfs:subPropertyOf ptop:hasRelative ;
                 rdfs:comment "Relation between a person and his/her siblings - brother or sister."@en ;
                 rdfs:label "has Sibling"@en .
```



Serialization: JSON-LD

```
{
  "@id" : "http://www.ontotext.com/proton/protontop#hasSibling",
  "@type" : [ "http://www.w3.org/2002/07/owl#ObjectProperty" ],
  "http://www.w3.org/2000/01/rdf-schema#comment" : [ {
    "@language" : "en",
    "@value" : "Relation between a person and his/her siblings - brother or sister."
  } ],
  "http://www.w3.org/2000/01/rdf-schema#label" : [ {
    "@language" : "en",
    "@value" : "has Sibling"
  } ],
  "http://www.w3.org/2000/01/rdf-schema#subPropertyOf" : [ {
    "@id" : "http://www.ontotext.com/proton/protontop#hasRelative"
  } ]
}
```



Tools for Writing OWL Files

Protégé - the premier open source ontology editor

- Client - full OWL support, does not support project management, collaboration happens outside of the client (e.g. GitHub)
- WebProtege- Online tool requiring an account, limited OWL support, greater focus on collaborative ontology maintenance
- Pros: UI/form-based, Support for reasoning, Validation, Imports/exports a number of syntax
- Cons: learning curve, a lot of clicking, opinionating about files (e.g. commenting, ordering), treats `rdf:Properties` without a range as `owl:AnnotationProperties`

Hand coding with text editors (lots of options with different strengths)

- Oxygen XML Editor - useful for any text editing, built-in support for XML and JSON
- Pros: Once comfortable with a syntax can be quicker to work in (copy and paste)
- Cons: Limited UI, requires external semantic validation and possibly syntax validation



Validation



Validation Tools

- Syntax validation
 - Online Turtle validators
 - e.g. <http://ttl.summerofcode.be/> - this one only seems to check for syntax
 - Online RDF/XML validators
 - e.g. <https://www.w3.org/RDF/Validator/>
 - Protégé will not load OWL files with syntax errors, including prefix checking
 - TopBraid Composer - similarly has syntax validation, including prefix checking
- Semantic validation:
 - Protégé - has a number of built in reasoners (more in the Protege demo soon)
 - TopBraid Composer - similarly has reasoning capabilities, also has built-in SHACL support



Demo: Building an OWL Ontology in Protégé



<Protege Client: The “Cliff Notes” version>

rdfs:seeAlso <theProtegeShortCourse> .

Protege Client: Views and Eclipse

1. Window/Views
2. Splitting Views
3. Active Ontology
4. Help

The screenshot shows the Protege client interface with several numbered annotations:

- 1**: Points to the **Window** menu in the top menu bar.
- 2**: Points to the **Ontology metrics** panel on the right side of the interface.
- 3**: Points to the browser address bar showing the URL `http://www.semanticweb.org/sf433/ontologies/2018/0/untitled-ontology-53`.
- 4**: Points to the **Ontology header** panel on the left side of the interface.

The interface includes a menu bar (Protégé, File, Edit, View, Reasoner, Tools, Refactor, Window, Ontop, Help), a browser address bar, a tab bar (Active Ontology, Entities, Individuals by class, Individual Hierarchy Tab, DL Query), and several panels: Ontology header, Ontology metrics, Annotations, and Imported ontologies.

Metric	Count
Axiom	0
Logical axiom count	0
Declaration axioms count	0
Class count	0
Object property count	0
Data property count	0
Individual count	0
Annotation Property count	0
DL expressivity	AL

Axiom	Count
SubClassOf	0
EquivalentClasses	0
DisjointClasses	0
GC count	0
Hidden GC Count	0

Axiom	Count
SubObjectPropertyOf	0

Protege Client: The Entities tab is your friend

1. Hierarchy
2. Annotations
3. Formal axioms

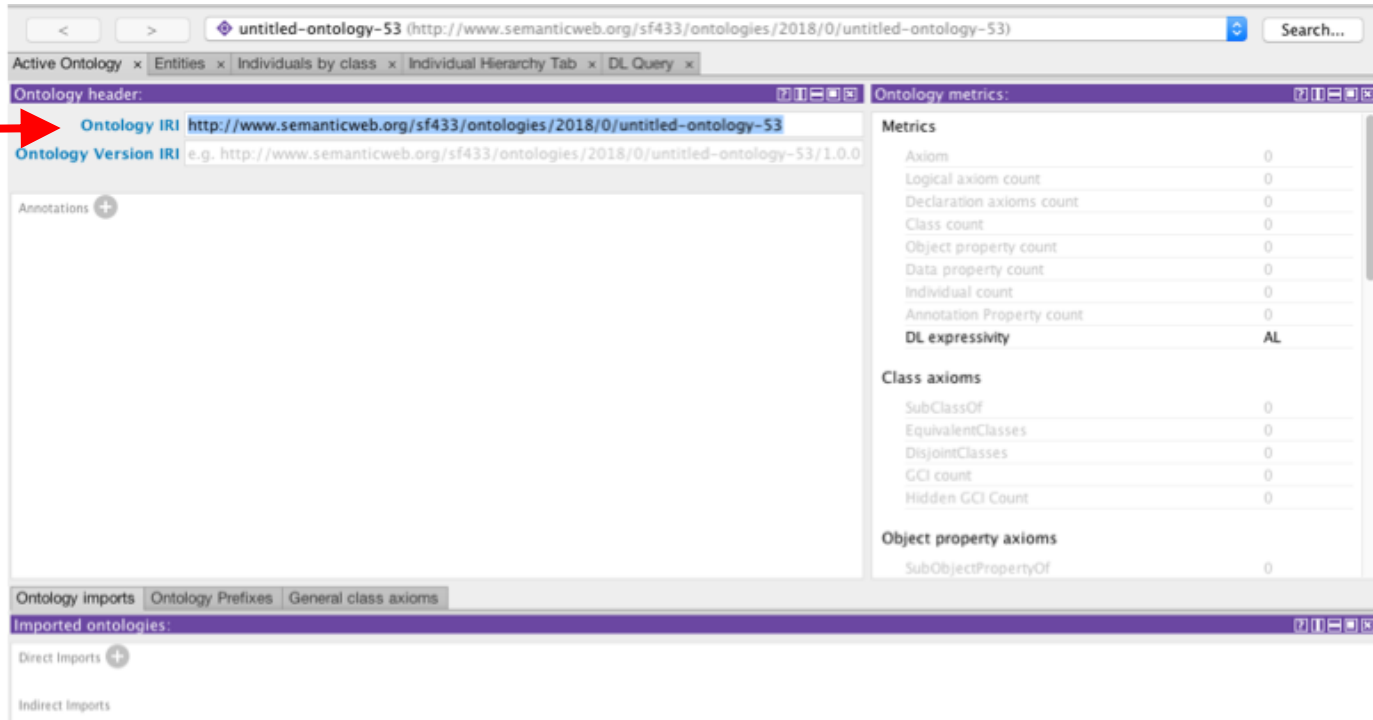
The screenshot displays the Protege Client interface for the ontology `http://bibliotek-o.org/1.1/ontology/`. The **Entities** tab is active, showing a class hierarchy on the left and class annotations on the right. A red arrow points to the **Entities** tab in the top navigation bar.

1. Hierarchy: The left pane shows a tree view of classes under `owl:Thing`. The `Activity` class is expanded, and the `Arranger` class is selected and highlighted in blue.

2. Annotations: The right pane shows the annotations for the `Arranger` class. The `rdfs:label` annotation is highlighted, with the value `Arranger` and a red '2' next to it. Other annotations include `skos:definition`, `dcterms:issued`, `dcterms:modified`, `skos:editorialNote`, and `skos:scopeNote`.

3. Formal axioms: The bottom pane shows the description for the `Arranger` class. It includes the `Equivalent To` and `SubClass Of` sections. The `SubClass Of` section shows that `Arranger` is a subclass of `Activity`, with a red '3' next to the `Activity` class name.

Protege Client: Setting the Base URI



The screenshot displays the Protege Client interface for an ontology. The browser address bar shows the URL: `http://www.semanticweb.org/sf433/ontologies/2018/0/untitled-ontology-53`. The 'Ontology header' tab is active, showing the following information:

- Ontology IRI:** `http://www.semanticweb.org/sf433/ontologies/2018/0/untitled-ontology-53` (highlighted by a red arrow)
- Ontology Version IRI:** e.g. `http://www.semanticweb.org/sf433/ontologies/2018/0/untitled-ontology-53/1.0.0`

The 'Ontology metrics' panel on the right provides a summary of the ontology's statistics:

Metric	Count
Axiom	0
Logical axiom count	0
Declaration axioms count	0
Class count	0
Object property count	0
Data property count	0
Individual count	0
Annotation Property count	0
DL expressivity	AL

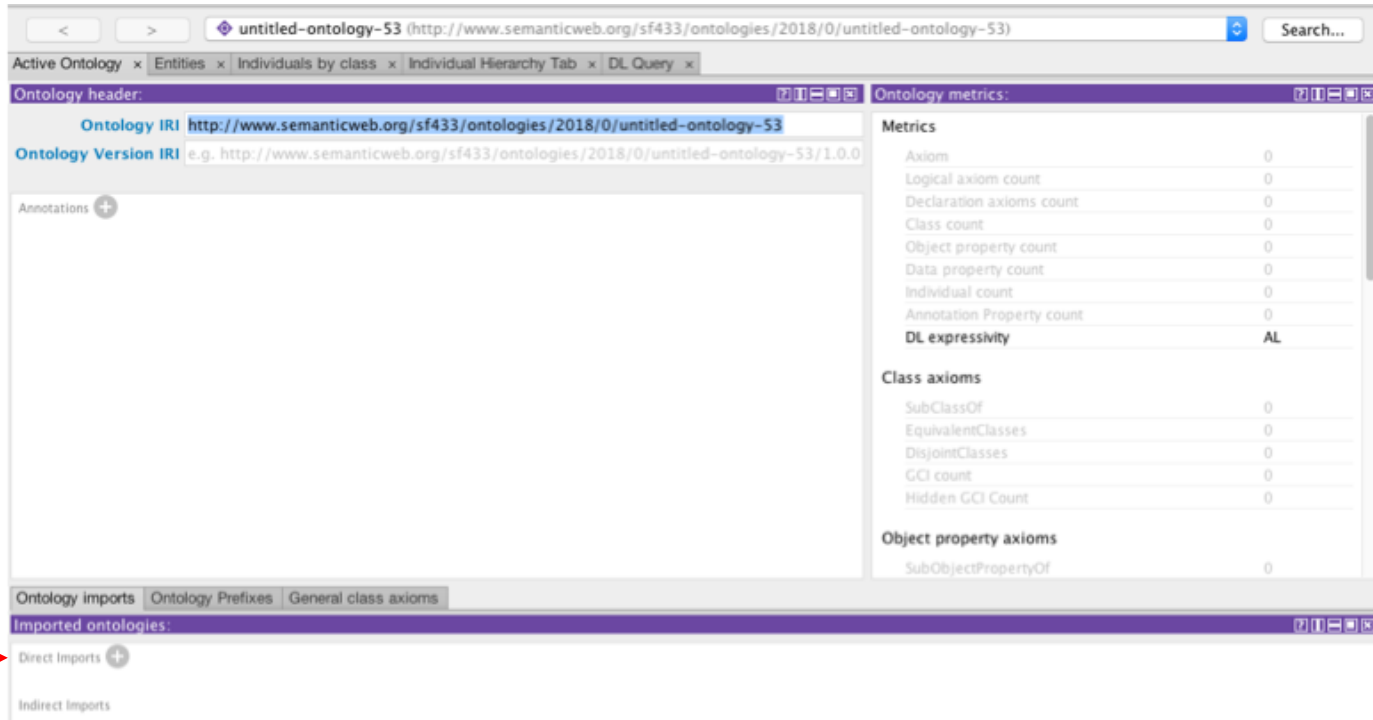
Below the metrics, the 'Class axioms' and 'Object property axioms' sections are visible, both showing zero counts for their respective sub-categories.

Protege Client: Naming Terms

The image shows a screenshot of the Protege Client interface. The main window displays a class hierarchy for `owl:Thing`. Overlaid on this is the "Entity Creation Preferences" dialog box, which is used to configure how new entities are named. The dialog is divided into three sections:

- Entity IRI:** This section allows the user to specify how the IRI for a new entity should be constructed. It includes:
 - Start with:** Radio buttons for "Active ontology IRI" and "Specified IRI". The "Specified IRI" option is selected, with the text `www.sf433.me/ontology` entered in the adjacent text field.
 - Followed by:** Radio buttons for "#", "/", and ":". The "/" option is selected.
 - End with:** Radio buttons for "User supplied name" and "Auto-generated ID". The "User supplied name" option is selected.
- Entity Label (for use with Auto-generated ID):** This section is only active if "Auto-generated ID" is selected. It includes:
 - Radio buttons for "Same as label renderer" and "Custom label". "Same as label renderer" is selected.
 - An "IRI" text field containing `http://www.w3.org/2000/01/rdf-schema#label`.
 - A "Lang" dropdown menu.
- Auto-generated ID:** This section is also active if "Auto-generated ID" is selected. It includes:
 - Radio buttons for "Numeric (iterative)" and "Globally unique". "Globally unique" is selected.
 - Fields for "Prefix" (containing `[type]_`), "Suffix" (empty), "Digit count" (set to 20), "Start" (set to 0), and "End" (set to -1).
 - A checkbox for "Remember last ID between Protégé sessions", which is currently unchecked.

Protege Client: owl:imports (remember reuse :)



The screenshot displays the Protege Client interface for an ontology named "untitled-ontology-53". The browser address bar shows the URL: <http://www.semanticweb.org/sf433/ontologies/2018/0/untitled-ontology-53>. The interface includes a search bar, tabs for "Active Ontology", "Entities", "Individuals by class", "Individual Hierarchy Tab", and "DL Query".

The "Ontology header" section shows the "Ontology IRI" as <http://www.semanticweb.org/sf433/ontologies/2018/0/untitled-ontology-53> and the "Ontology Version IRI" as [e.g. http://www.semanticweb.org/sf433/ontologies/2018/0/untitled-ontology-53/1.0.0](http://www.semanticweb.org/sf433/ontologies/2018/0/untitled-ontology-53/1.0.0). There is also an "Annotations" section with a plus sign.

The "Ontology metrics" section provides a table of counts for various metrics:

Metric	Count
Axiom	0
Logical axiom count	0
Declaration axioms count	0
Class count	0
Object property count	0
Data property count	0
Individual count	0
Annotation Property count	0
DL expressivity	AL

The "Class axioms" section shows counts for:

Class Axiom	Count
SubClassOf	0
EquivalentClasses	0
DisjointClasses	0
GCI count	0
Hidden GCI Count	0

The "Object property axioms" section shows a count for:

Object Property Axiom	Count
SubObjectPropertyOf	0

At the bottom, the "Ontology imports" section is active, showing "Imported ontologies" with a plus sign. A red arrow points to the "Direct Imports" section.

Protege Client: Importing individuals (reuse :)

The image shows a screenshot of the Protege Client interface. The main window displays the ontology editor for `urn:absolute:www.sf433.me/ontology`. The interface includes a top menu bar with options like "Active Ontology", "Entities", "Individuals by class", "DL Query", and "Individual Hierarchy Tab". Below the menu bar, there are tabs for "Data properties", "Annotation properties", "Datatypes", and "Individuals". The main workspace shows the "Class hierarchy" for `owl:Thing` and the "Class Annotations" for `owl:Thing`. A dialog box titled "Create a new OWLClass" is open in the foreground, with the following fields and buttons:

- Name:** Short name or full IRI or Prefix-Name
- IRI:** IRI (auto-generated)
- New entity options...** button
- Cancel** and **OK** buttons

The dialog box is positioned over the main workspace, which also shows a list of instances for `owl:Thing` at the bottom.

Protege Client: Inferencing

1. Turning on
2. Viewing
3. Highlighted

The screenshot displays the Protege Client interface. The 'Reasoner' menu is open, showing options: Start reasoner, Synchronize reasoner, Stop reasoner (marked with a red '1'), Explain inconsistent ontology, and Configure... The 'Object Properties' list on the left shows various properties, with 'has preferred title' highlighted in blue. The 'Annotations' panel on the right shows the inferred annotations for 'has preferred title', including 'rdfs:label', 'skos:definition', and 'skos:scopeNote'. The 'Description' panel at the bottom right shows the 'Inverse' property highlighted in yellow (marked with a red '3').

Protege Client: Inferencing and errors

The screenshot displays the Protege Client interface for the 'protontop' ontology. The main window shows the 'Individuals' tab for the class 'Participant in a Happening'. A dialog box titled 'Help for inconsistent ontologies' is open, providing instructions on how to handle an inconsistent ontology. The dialog text reads:

Your ontology is inconsistent which means that the OWL reasoner will no longer be able to provide any useful information about the ontology.

You have several options at this point:

- Click the Explain button to try the Protege explanation facility.
- If you think you know what the problem is, click Cancel to fix the ontology yourself.
- Some reasoners come with command line tools that will provide complete explanations for inconsistent ontologies.

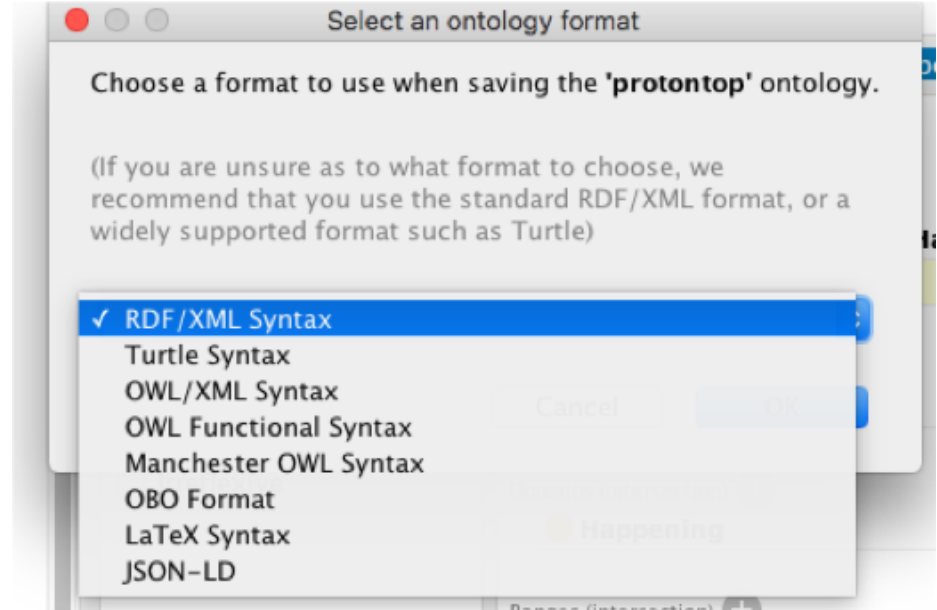
The dialog has 'Cancel' and 'Explain' buttons. In the background, the 'Description: Participant in a Happening' panel shows the following relationships:

- Equivalent To: (empty)
- SubProperty Of: 'Entity Participating in a Happening'
- Inverse Of: 'Involved in'
- Domain Intersections: 'Happening'
- Range Intersections: 'Agent'
- Disjoint With: (empty)
- SuperProperty Of (Chain): (empty)

The status bar at the bottom indicates 'Reasoner active but the ontology is inconsistent' and 'Show Inferences' is checked.

Protege Client: Exporting

1. Save as...
2. Then choose format and destination





Hand-coding

- Some people prefer it because copy-paste-edit is faster than clicking around the Protégé UI.
- Not limited to Protégé's set of annotation properties.
- Complete control over final output.
- Does require more knowledge than using a tool like Protégé.
- Start with Protégé, look at the output, then try hand-coding if you're interested.
- Many people make the switch after they are comfortable with OWL.
 - Similarly, many people switch to hand-coding SHACL from TopBraid Composer.
- An editor like OxygenXML checks syntax as you edit.
- Otherwise, use online syntax validation tools.
- Semantic validation: load ontology into Protégé to run reasoner, or run an external reasoner.
- Protégé is also very useful to view an ontology, even if you prefer to create one via hand-coding.



Exercise

Building OWL in Protégé



Discussion