You’ve Heard the Jargon, Now Play the Game: Linked Data on Your Laptop!

Jean M. Pajerek, Coordinator and Speaker
Anna-Sophia Zingarelli-Sweet, Speaker
Jacqueline Magagnosc, Moderator
Jean M. Pajerek  
Director for Information Management  
Cornell Law Library  
jmp8@cornell.edu

Anna-Sophia Zingarelli-Sweet  
Cataloging and Metadata Specialist  
California State University Northridge  
annasophia.zingarellisweet@csun.edu
Outline

Part I
• Review of basic linked data concepts
• Walk-through of a pizza ontology

Part II
• Hands-on practice with Protégé ontology software
• Review of hands-on exercise
• Querying linked data using SPARQL
Information about some of Shakespeare’s plays presented in a familiar format

<table>
<thead>
<tr>
<th></th>
<th>Title</th>
<th>Author</th>
<th>Genre</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>As You Like It</td>
<td>Shakespeare</td>
<td>Comedy</td>
</tr>
<tr>
<td>3</td>
<td>Hamlet</td>
<td>Shakespeare</td>
<td>Tragedy</td>
</tr>
<tr>
<td>4</td>
<td>Henry V</td>
<td>Shakespeare</td>
<td>Historical</td>
</tr>
<tr>
<td>5</td>
<td>King Lear</td>
<td>Shakespeare</td>
<td>Tragedy</td>
</tr>
<tr>
<td>6</td>
<td>Love’s Labour's Lost</td>
<td>Shakespeare</td>
<td>Comedy</td>
</tr>
<tr>
<td>7</td>
<td>MacBeth</td>
<td>Shakespeare</td>
<td>Tragedy</td>
</tr>
<tr>
<td>8</td>
<td>Measure for Measure</td>
<td>Shakespeare</td>
<td>Comedy</td>
</tr>
<tr>
<td>9</td>
<td>Othello</td>
<td>Shakespeare</td>
<td>Tragedy</td>
</tr>
<tr>
<td>10</td>
<td>The Tempest</td>
<td>Shakespeare</td>
<td>Tragedy</td>
</tr>
<tr>
<td>11</td>
<td>The Winter’s Tale</td>
<td>Shakespeare</td>
<td>Comedy</td>
</tr>
<tr>
<td>12</td>
<td>Twelfth Night</td>
<td>Shakespeare</td>
<td>Comedy</td>
</tr>
</tbody>
</table>
Triple statements

Subject (Resource) \( \rightarrow \) Predicate (Property) \( \rightarrow \) Object (Value)

This work \( \rightarrow \) Was written by \( \rightarrow \) This author

Some of the triples expressed in the spreadsheet

• Resource 2 has title “As You Like It”
• Resource 2 has author “Shakespeare”
• Resource 2 has genre “Comedy”

In each of these triples, Resource 2 is the subject (or resource). The predicate (or property) is expressed by what is in the column heading, i.e., (has) Title, (has) Author, or (has) Genre.

The object of the triple is a value contained in a cell, e.g., “As You Like It,” “Shakespeare,” “Comedy.”
Triple statements

• Subject identifies: “Resource of interest”
• Predicate identifies: Property of the “resource of interest,” a relationship
• Object identifies: Property value, a resource that has a relationship to the “resource of interest”
Triple statements

Measure for Measure

Has author

Shakespeare

URI for work, “Measure for Measure”

URI for Dublin Core term: creator [read: has creator]

URI for author, William Shakespeare
The triple statement:

“Measure for Measure” has creator (or author) Shakespeare

can be expressed in a way that machines can interpret, using VIAF URIs for “Measure for Measure,” Shakespeare, and the Dublin Core term for “has creator”:

< http://viaf.org/viaf/181321659>
<http://purl.org/dc/terms/creator>
< http://viaf.org/viaf/96994048/>
### Triples about parts of the United Kingdom

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scotland</td>
<td>part Of</td>
<td>The UK</td>
</tr>
<tr>
<td>England</td>
<td>part Of</td>
<td>The UK</td>
</tr>
<tr>
<td>Wales</td>
<td>part Of</td>
<td>The UK</td>
</tr>
<tr>
<td>Northern Ireland</td>
<td>part Of</td>
<td>The UK</td>
</tr>
<tr>
<td>Channel Islands</td>
<td>part Of</td>
<td>The UK</td>
</tr>
<tr>
<td>Isle of Man</td>
<td>part Of</td>
<td>The UK</td>
</tr>
</tbody>
</table>
In this table, we merge some data about Shakespeare with some data about the UK. This is easily done when all of the data is expressed in triples. Notice the points of intersection. What inferences can we draw from this small data set?

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shakespeare</td>
<td>wrote</td>
<td>King Lear</td>
</tr>
<tr>
<td>Shakespeare</td>
<td>wrote</td>
<td>Macbeth</td>
</tr>
<tr>
<td>Anne Hathaway</td>
<td>married</td>
<td>Shakespeare</td>
</tr>
<tr>
<td>Shakespeare</td>
<td>livedIn</td>
<td>Stratford</td>
</tr>
<tr>
<td>Stratford</td>
<td>isIn</td>
<td>England</td>
</tr>
<tr>
<td>Macbeth</td>
<td>setIn</td>
<td>Scotland</td>
</tr>
<tr>
<td>England</td>
<td>partOf</td>
<td>UK</td>
</tr>
<tr>
<td>Scotland</td>
<td>partOf</td>
<td>UK</td>
</tr>
</tbody>
</table>

Inferences we can draw from the merged data:

• Shakespeare married Anne Hathaway
• Shakespeare lived in England
• Shakespeare lived in the UK
• Stratford is part of the UK
• Macbeth is set in the UK
Graphic representation of triples describing (a) some of Shakespeare’s plays and (b) some parts of the United Kingdom.
Combined graph of triples about Shakespeare and triples about the United Kingdom.

Linked data graph showing relationships between “Shakespeare” and “Stratford”

Graph created using RelFinder:
http://www.visualdataweb.org/relfinder.php
Wikipedia defines “ontology” as:

“...a formal naming and definition of the types, properties, and interrelationships of the entities that really or fundamentally exist for a particular domain of discourse”
If B=the class of Artworks, and A=the class of Sculptures, then A is a *subclass of* B. Everything that is a member of subclass A is also a member of (super)class B. Everything that is a sculpture is also an artwork (but not every artwork is a sculpture).
Vocabularies and Ontologies

Example of discovering relationships through inferencing:

• **Data set** says “‘The Kiss’ is a sculpture”
• **Ontology** says “all sculptures are also artworks”
• A semantic web program that understands the concept of “all X are also Y” can discover (i.e., infer) a new relationship:
• “‘The Kiss’ is an artwork”
Vocabularies and ontologies:

• are necessary for discovering relationships on the semantic web
• define classes and properties
• define relationships between resources
• facilitate “inference” discovery of new information
Why use ontology software to understand linked data?
Now that we have some idea about how linked data is structured, how do we search for it and retrieve it?
Querying RDF data with SPARQL

- Data is useless unless you can access it, query it, and retrieve query results
  - SQL is a common query language for relational databases
- **SPARQL is the query language for RDF data**
  - SPARQL = SPARQL Protocol and RDF Query Language
- Based on the RDF triple pattern
- Written in a syntax very much like Turtle
- Queries can range from very simple to very complex
SPARQL as tell-and-ask system

You **tell** the system some information by the data you input, and you can then **ask** questions about that data.

For RDF data, replace one or more parts of a triple with **question words**. Question words are written with a question mark at the start.

**Example:**

```sparql
@PREFIX : <http://mydomain.org/artworks/>

• Tell: :Michelangelo :created :David.
• Ask: ?who :created :David.
• Answer: :Michelangelo
• Answer: :created
• Answer: :David
```
SPARQL SELECT queries

- SPARQL syntax is very similar to Turtle
- A SELECT query has two parts:
  1. A set of question words, indicated by the keyword `SELECT`
  2. A question pattern, written in braces, indicated by the keyword `WHERE`

- Example:
  ```sparql
  PREFIX : <http://www.sjm.org/artworks#>
  SELECT ?subject ?object
    WHERE { ?subject :created ?object }
  ```
  - The system returns every subject and object asserted for the property `:created` in the triplestore database
Try out this SPARQL query in your own Artworks knowledgebase!
PREFIX sjm: <http://www.sjm.org/artworks#>

SELECT ?subject ?object

WHERE { ?subject sjm:created ?object}

We’re looking for all the subjects and objects that are connected by the “created” property within our declared namespace, which is represented by the prefix “sjm.”
Your SPARQL screen should look like this.

When you’re ready, click on the “Execute” button to run the query.
Did you get the results you expected?

PREP// Prefixes

PREFIX sjm:<http://www.sjm.org/artworks#>

SELECT ?subject ?object

WHERE { ?subject sjm:created ?object .}

<table>
<thead>
<tr>
<th>subject</th>
<th>object</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘Buonarroti, Michelangelo, 1475-1564’</td>
<td>David</td>
</tr>
<tr>
<td>‘Cahlo, Frida’</td>
<td>‘The Love Embrace of the Universe’</td>
</tr>
</tbody>
</table>
Exploring SPARQL queries

A fun and instructive place to go to explore sample SPARQL queries is:

**Wikidata Query Service** – click on the “Examples” button, then click on the “Run” button below the query box

https://query.wikidata.org/

There’s a very helpful intro to the Wikidata Query Service at:

https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/A_gentle_introduction_to_the_Wikidata_Query_Service
Sparklis – a graphical interface for SPARQL queries

http://www.irisa.fr/LIS/ferre/sparklis/
Other sites for exploring SPARQL queries:

European Union Open Data Portal

The British National Bibliography as Linked Open Data
http://bnb.data.bl.uk/flint-sparql

Getty Vocabularies LOD (Linked Open Data) Sample Queries
http://vocab.getty.edu/queries

FactForge – Scroll down to the “Saved SPARQL queries”
http://factforge.net/
Tell me again ... WHAT does this have to do with my job?
BIBFRAME is an ontology!
Check out the BIBFRAME Comparison Tool at:
http://id.loc.gov/tools/bibframe/compare-id/full-ttl
Casalini’s SHARE-VDE project
http://www.share-vde.org/sharevde/clusters?l=en
Thank you for your attention!
Note: This is a hands-on exercise using the freely available ontology software Protégé. Protégé can be downloaded here: http://protege.stanford.edu/. You will also need a file called Exercise_artworks.owl, downloadable from this URL: http://tinyurl.com/ybodlyst.

Launch Protégé and use it to open the Artworks ontology file (Exercise_artworks.owl). To open the ontology file, click Control-O or, under the File menu, select “Open.” Navigate to where the “Exercise_artworks.owl” file is on your computer. Your screen should look something like this:

Under the “View” menu, make sure the “Render by label” view is selected. Notice the tabs near the top of the display. Click on the tab that says “Entities.” When you click the “Entities” tab, it brings up a display that looks like this, with perhaps some of the same tabs displayed again:

This exercise is adapted from course materials created by Prof. Steve J. Miller. Used with permission.
The “Object properties” tab

For the purposes of this exercise, we’ll focus on Classes, Object Properties, Data Properties, and Individuals. The screenshot above shows Object properties (within the Entities tab). Object properties assert a specific relationship between two different entities in a triple statement, for example, “Michelangelo created David.” In this triple statement, “created” is the object property (also the “predicate” of the triple), asserting a relationship between the individuals “Michelangelo” and “David.” Notice that just like in the pizza ontology, these object properties begin with lower-case letters, and are written out in what’s called “camel case.” According to Wikipedia, camel case “is the practice of writing compound words or phrases such that each word or abbreviation in the middle of the phrase begins with a capital letter, with no intervening spaces or punctuation.” Note also that in Protégé, object properties are symbolized by blue rectangles. So far, our ontology has just a few object properties. They are all subproperties of the topObjectProperty. Note that it’s possible for a property like “created” to have subproperties of its own; for example, a subproperty of “created” in this namespace might be “painted” or “sculpted.”

Next, let’s look at the “Individuals” tab. The ontology has been pre-populated for you with a few individual entities (also known as “instances”). In Protégé, individuals, or instances, are symbolized by purple diamond shapes. The caption bars in the various panes of the display are color-coded to
correspond to the parts of the ontology they refer to. The caption bars here are purple, to correspond to the purple diamonds symbolizing individuals.

The "Individuals" tab

Something interesting to notice about this screen is that there are two individuals labelled "Italian." By clicking on each instance labeled "Italian," you will see that one instance refers to Italian as a language, while the other instance refers to Italian as a nationality. Alternatively, under the View menu at the top of the screen, select “Render by prefixed name.” Notice the changes in the appearance of the list of individuals. In looking at the prefixed names (indicated by the preceding colons), we can see from the comment in the Annotations pane that these two entities actually have different names, but their rdfs (RDF Schema) labels are the same (see below):

Since we’re in the Individuals tab, let’s make an object property assertion (i.e., a triple statement) about Michelangelo. To do this, select Michelangelo in the list of Individual entities (you might have to double
click). Notice that information about Michelangelo now appears in some of the other panes of the display, noting in particular the “Property assertions” pane on the bottom right. We can see that there are some “object property” assertions that have been made about Michelangelo, as well as some “data property” assertions (i.e., his birth and death dates). We’re going to add an object property assertion that Michelangelo worked in Florence. Remember that an object property asserts a specific relationship between two different entities in a triple statement; in this case, the relationship being asserted (“workedIn”) is between Michelangelo and Florence.

Under the “View” menu at the top of screen, select “Render by label” to return the display to the non-prefixed view. If you don’t do this, you’ll have to remember to add the colon to your triples – it works, but how annoying! To add the object property assertion that “Michelangelo worked in Florence” to our ontology, click on the plus sign in the Property assertions pane, right next to where it says “Object property assertions” (where the red circle is in the illustration below).

Once you’ve clicked the plus sign, a small screen will pop up, with two blanks for you to fill in like this:
We are asserting an object property with Michelangelo as the subject, workedIn as the predicate, and Florence as the object. **Data entry tip: once you’ve typed in a few letters, try using Control-space to auto-complete the field.** Once you click OK, your object property assertion will appear in the Property assertions pane, along with the other property assertions about Michelangelo!

Now we’re going to add a new individual to the list of individuals, Frida Kahlo, and make some property assertions about her.

Under the **View menu**, switch the view to “Render by prefixed name.” In the pane displaying the list of individuals, click near the upper left-hand corner, where you see the purple diamond with a plus sign next to it:

In the box that appears, type Frida Kahlo, and click OK. Notice that :FridaKahlo appears in the list of individuals. In this case, Protégé supplies the preceding colon for you.

Now we’re going to add a lexical label for our new entity. Make sure Frida Kahlo is selected in the list of individuals. In the “Annotations” pane, click on the plus sign. On the screen that appears, select rdfs:label in the left-hand pane. Type Kahlo, Frida in the right-hand pane, making sure that you are in the
“Literal” tab of that pane. At the bottom right-hand side of the screen, select \texttt{xsd:string} from the drop-down menu next to the word “Type.” Click OK when you’re done.

Under the View menu, switch the View to “Render by label” and notice what Frida Kahlo’s name looks like. Switch the view back to “Render by prefixed name.”

Add a new individual with the name “Work9876.” Give the entity (a work this time) the annotation label: The Love Embrace of the Universe. Select the individual Frida Kahlo and make an object property assertion that she created Work9876 (The Love Embrace of the Universe). Be sure to add the colon before the object property “created” and in front of “Work9876.” Your new object property assertion should look like this:
While remaining within the **Individuals** tab, make a **data property assertion** (within the **Property assertions pane**, below the Object property assertions) that Kahlo’s birthdate was 1907-07-06. In Protégé, data properties are represented by green rectangles. Data properties relate an entity to literal data, like numbers or a textual string; in this case, we’re asserting an individual’s date of birth. When making this assertion, in the dropdown menu for “Type,” click on “string,” then on the “OK” button. This asserts that the datatype of this value is a string of characters. (Note: her death date was 1954-07-13 if you want to add that.) You can always go back and edit an assertion by clicking on the gray circle next to it. Your new data property assertion(s) should look like this:

Now we’re going to create some classes in our ontology. Classes are sets of types of things and subclasses are subsets of their parent class. Every class should have at least a few individual entities as members of that class or set; otherwise, there is no point in the existence of that class. **Individuals may be asserted to be instances of one or more classes.** All classes in OWL are subclasses of **owl:Thing.** Every member is a “thing” or “entity” in the RDF / linked data /Semantic Web sense of that term.

Click on the “Classes” tab in Protégé. In Protégé, classes are symbolised by gold circles. Notice that owl:Thing is already at the top of the class hierarchy. All the classes we’ll create will be subclasses of owl:Thing. This is true of every OWL ontology.

To add classes to the class hierarchy, select “owl:Thing” and notice that the button in the upper-left corner of the pane becomes enabled. This is the “Add subclass” button; click on it.

In the window that appears, type Artist, and click OK. “Artist” now appears in the class hierarchy as a subclass of owl:Thing. Add Artwork and Institution as subclasses of owl:Thing in the same way. Notice that we are adhering to the ontological convention of naming classes with words that begin with capital letters.
Now we are going to assert that some individuals are instances of the classes we created. To do this, click on the “Individuals” tab and select Michelangelo. In the bottom middle pane labeled “Description,” click on the plus sign next to the word “Types”:

In the window that appears, select the tab that says “Class hierarchy” and then click on “Artist” within the hierarchy that appears; then click OK.

Returning to the “Classes” tab, when you click on “Artist” within the class hierarchy, you’ll see in the “Description” pane that Michelangelo has been added as an “Instance” of the class “Artist.” Following the same steps, assert that Frida Kahlo is an instance of the “Artist” class, and that “David” and “The Love Embrace of the Universe” are instances of the class “Artwork.” The same result can be achieved from within the Classes tab, by selecting one of the classes and clicking on the plus sign next to Instances in the Description pane on the bottom right side of the screen, then selecting the appropriate individual entity from the list that appears.
In the **Classes** tab, create the following class hierarchy (hint: “Person” is a subclass of **owl:Thing**; you’ll have to click on owl:Thing to position your cursor correctly):

- Person
  - Artist
    - Painter
    - Sculptor
    - Architect

Start by creating a new top-level class called “Person” directly under “Thing.” Then click and drag the already existing Artist class on top of Person, and it will become a subclass. An alternative is to click on SubClassOf in the Description pane for Artist and select Person.

Add more new top-level classes named Style/Period, Language, and GeographicLocation.

Add three sibling subclasses under GeographicLocation named Continent, Country, and City. Your class hierarchy should look like this:

![Class hierarchy diagram]

Go the the **Individuals** tab and assert that GabrieleMunter is a member (or instance) of the Painter class.

In the **Object properties** tab, add a new object property named “hasStyle/Period.”

In the **Individuals** tab, add a new individual, “NaiveArt.” In the Annotations pane of the Individuals tab, create an rdfs:label for this individual, Naive art.

Assert that NaiveArt is a Style/Period.
In the Individuals tab, make an **object property assertion** about the individual artwork “Love Embrace of the Universe” that it hasStyle/Period “NaiveArt.”

Add MexicoCity, Coyoacan, Italy, Mexico, Europe and NorthAndCentralAmerica to the list of Individuals.

Assert that Florence, MexicoCity, and Rome are each a City (members of the City class), that Italy and Mexico are each a Country, and that Europe and NorthAndCentralAmerica are each a Continent.

Assert that GalleriaDellAccademia is an Institution.

Assert that Frida Kahlo was born in Coyoacan and that she worked in Mexico City. Hint: “wasBornIn” and “workedIn” are object properties. Your property assertions for Frida Kahlo should look like this:

In the **Object properties** tab, add a new object property called **isLocatedIn**. Then assert that Mexico City isLocatedIn Mexico, that Mexico isLocatedIn NorthAndCentralAmerica, that Rome isLocatedIn Italy, and Italy isLocatedIn Europe.

**Question**: why did we use the object property “isLocatedIn” to show the relationship between cities, countries, and continents instead of making a class hierarchy for them? Why is it valid to have all three of these as sibling subclasses of GeographicLocation, but not to have City as a child subclass of Country and Country a subclass of Continent?

Add a new class named **Gender**. We want to make the statements that FridaKahlo hasGender Female and Michelangelo hasGender Male, so we’ll also need to create a new object property, **hasGender**. In a traditional controlled vocabulary consisting of broader and narrower terms in a hierarchy, Male and Female could be narrower terms under Gender. Male and Female are also types of gender. In our ontology, should Male and Female be Classes or Individuals? Why? Try adding one as a
subclass of Gender and then try making a statement about one of the artists using the hasGender property. Why is this impossible to do? Instead, add Male and Female as individuals and then make the statements about Kahlo’s and Michelangelo’s genders.

Save your ontology (Ctrl-S or “Save” under the File menu.).

We’ve made some assertions that enable some inferences about class membership to be made. To see the inferences, click on the Reasoner menu at the top of the Protégé window and click on “Start Reasoner.” If you are shown a choice of reasoners, choose HermiT.

Be sure that the “Show Inferences” box in the bottom right corner of the Protégé window has been checked. Based on what we have, you won’t see any inferences immediately.

Select the Classes tab, then go to the Window menu option at the top of the Protégé screen and select Window > Views > Ontology views > Classification results. Drag and then release the black circle somewhere in the Classes frame area – try dragging it into the lower part of the Annotations frame until you see a black line appear (showing that a new frame will be added) then release. You should now see a new frame that looks like this:
For the next part of the exercise, stop the reasoner by selecting “Stop reasoner” under the Reasoner menu at the top of the screen.

**Domain and Range**

This part of the exercise will introduce the concepts of “Domain” and “Range.”

- “Domain” indicates that the subject of a property must be a member of a specific class.
- “Range” indicates that the object of a property must be a member of a specific class.

For example, let’s say we want to connect the property “created” with the types of entities (classes) that can be used with it:

The subject of the “created” property must always be a member of the class “Artist”
The object (value) of the “created” property must always be a member of the class “Artwork”

Put another way, the domain of the “created” property is “Artist,” and the range of the “created” property is “Artwork.” Therefore if we assert that Michelangelo created David, the ontology can infer that Michelangelo must be an Artist and that David must be an Artwork. The concepts of domain and range can apply to both object properties and data properties.

Domain and range assertions are added by clicking on the plus signs next to the words “Domains” and “Ranges” in the “Description” pane in the bottom right corner of the Object properties screen. Click on
the **Object properties** tab and add domain and range specifications for some of the object properties we’ve asserted:

Object property - **created**  
domain - Artist  
range - Artwork

Object property - **isHousedIn**  
domain - Artwork  
range - Institution

Object property - **isLocatedIn**  
domain - GeographicLocation  
range - GeographicLocation

In the screenshot below, we’ve asserted that for the object property “**created**,” the domain is “**Artist**” and the range is “**Artwork**.” Trying to translate this into English, we could say that “Some member of the **Artist** class **created** some member of the **Artwork** class.” We have asserted that Frida Kahlo and Michelangelo are members of the Artist class. (Gabriele Münter was asserted to be a member of the class “Painter,” but her membership in the class “Artist” has only been inferred by the reasoner, not explicitly asserted by us.) We have asserted that “David” and “The Love Embrace of the Universe” are members of the Artwork class. Having asserted a domain and range for the property “created” within our small knowledgebase, triple statements that use the object property “created” must have a subject who is either Frida Kahlo or Michelangelo, and an object that is either “David” or “The Love Embrace of the Universe.”

Notice that for the object property “**isLocatedIn**,” the domain and range are the same, i.e., the class “GeographicLocation.” In English, we’re saying something like: “Some member of the GeographicLocation class is located in some other member of the GeographicLocation class.” For example, “**MexicoCity**” (a member of the “City” subclass of the “GeographicLocation” class) **isLocatedIn**  
**Mexico** (a member of the “Country” subclass of the “Geographic Location” class). Assertions that are true of a class are also true of its subclasses.
To see the impact that asserting domain and range can have on inferencing, click on the **Individuals** tab and select Michelangelo. In the **Description** pane (the bottom center pane of the Individuals display) where we asserted that Michelangelo is a member of the “Artist” class, click on the X button to remove this assertion. Then, under the **Reasoner** menu at the top of the screen, select “Start reasoner.” Make sure the “Show inferences” box in the bottom right corner of the screen is checked. Note that an **inference** appears (highlighted in pale yellow) which says, essentially: “Michelangelo belongs to the “Artist” class.” Clicking on the question mark next to the inference, we see this rationale for the inference:

Because we asserted that the domain for the object property “created” has to be an artist, and we previously asserted that “Michelangelo created David,” the reasoner was able to infer correctly that Michelangelo must be an artist.

Data properties can also have domain and range specifications. Click on the **Data properties** tab and select the data property “hasBirthDate.” In the “Description” pane, in the bottom right corner of the screen, assert that the domain for this data property is the class “Person” and the range is **xsd:string**; xsd:string is a “Built in datatype” in Protégé, so you’ll need to access the “Built in datatypes” list after clicking on the plus sign next to the word “Ranges” to select xsd:string (see the following screenshot). After you’ve selected it, click OK.
We’ve asserted that an instance of the “Person” class has to be the subject of the property (or predicate) “hasBirthDate.” The object of such an assertion has to be a string of “literal” values, in this case, the numerals that indicate the birthdate of whoever the subject of the triple is, e.g., “Frida Kahlo (an instance of the “Person” class) hasBirthDate 1907-07-06 (a string of literal values).

If you want to save your ontology to look at later, select “Save” under the **Edit** menu, or type Control-S.

The screenshot below shows a portion of what our ontology looks like when serialized (formatted) using Turtle (Terse RDF Triple Language) syntax. Turtle is considered a relatively human-friendly RDF syntax, but it is still not as easy for humans to interpret as the Protégé version.
The bracketed lines at the top are called the “namespace declaration.” When a namespace is defined for an element, all child elements with the same prefix are associated with the same namespace.
Viewing ontologies in Protégé

It can be instructive (or completely baffling; you decide!) to look at existing ontologies using Protégé. For example, to look at the Getty Vocabularies ontology in Protégé, click on File menu and select “Open from URL.” If a dialogue box appears, asking whether you want to open the ontology in the current window, click on “No.” Type the following in the box labeled “URI”:

http://vocab.getty.edu/ontology

Then click “OK.” You should see something like this:

Here are a couple law-related ontologies you can open the same way:

http://publications.europa.eu/mdr/resource/eli/eli.owl (European Legislation Identifier Metadata Ontology) - this one is fun because, in addition to being law-related, it uses some concepts and terminology taken from FRBR and RDA!

http://justinian.leibnizcenter.org/MetaLex/metalex-cen.owl (Open XML Interchange Format for Legal and Legislative Resources)
You can also open the BIBFRAME 2.0 vocabulary from within Protégé; in this context, “vocabulary” refers to definitions of both classes and properties:

http://id.loc.gov/ontologies/bibframe.rdf
GLOSSARY

**Classes**: Also known as sets. Members of classes share some properties or characteristics.

**Entities**: Entities are the fundamental building blocks of OWL ontologies, and they define the vocabulary — the named terms — of an ontology. Classes, datatypes, object properties, data properties, annotation properties, and named individuals are entities, and they are all uniquely identified by an identifier.

**Individuals**: Individuals (instances) are the basic, "ground level" components of an ontology. The individuals in an ontology may include concrete objects such as people, animals, tables, automobiles, molecules, and planets, as well as abstract individuals such as numbers and words (although there are differences of opinion as to whether numbers and words are classes or individuals). An ontology need not include any individuals, but one of the general purposes of an ontology is to provide a means of classifying individuals, even if those individuals are not explicitly part of the ontology. Individuals may be members of one or more classes. The term *instances* is sometimes used interchangeably with *individuals*.

**Namespace**: Namespaces identify vocabularies and syntaxes used to make semantic statements about resources. Namespaces are used to declare formats, syntaxes, and sources of terms. In RDF, the assumption is that a class or property from any namespace can be used at any point. It is quite normal for the subject, predicate, and object of a triple to be URIs from different namespaces. RDF promotes the mixing and matching of vocabularies, and the goal of Linked Data is to connect as many data points from as many different sources as possible. Namespace *prefixes* are often used to stand in for a URI to streamline the syntax of RDF metadata.

**Ontology**: A formal representation of a set of concepts within a domain (i.e., an area of knowledge) and the relationships between those concepts.

**Properties**: Properties describe the relationships between entities, or between entities and data values.

**RDF - The Resource Description Framework**: This is the basic framework that the rest of the Semantic Web is based on. RDF provides a mechanism for allowing anyone to make a basic statement about anything and layering these statements into a single model. RDF has been a recommendation from the W3C since 1999.

**RDFS (RDF Schema)**: RDF Schema provides a data-modelling vocabulary for RDF data. RDF Schema is a semantic extension of RDF. It provides mechanisms for describing groups of related resources and the relationships between these resources. RDF Schema is written in RDF. These resources are used to determine characteristics of other resources, such as the domains and ranges of properties (LC BIBFRAME training, module 2, slide 44). RDFS has been a W3C recommendation since 2004.

**RDF triple**: The fundamental data structure of RDF. A triple is made up of a subject, predicate, and object. Vocabularies and ontologies define relationships within a triple statement.
**Triplestore**: A database for storage and retrieval of RDF triple statements; searched using RDF query language called “SPARQL.”

**Simple Knowledge Organization System (SKOS)**: SKOS is a W3C recommendation designed for representation of thesauri, classification schemes, taxonomies, subject-heading systems, or any other type of structured controlled vocabulary. SKOS is part of the Semantic Web family of standards built upon RDF and RDFS, and its main objective is to enable easy publication and use of such vocabularies as linked data.

**SPARQL**: RDF query language; searches triplestore data sets

**SPARQL endpoint**: a service that accepts SPARQL queries and returns results

**Serialization**: RDF serialization formats are the programming languages that are used to encode RDF triple statements. Serialization is the process of translating data structures into a format that can be stored or transmitted across a network connection link and reconstructed later in the same or another computer environment.

**Web Ontology Language (OWL)**: The Web Ontology Language (OWL) is a family of knowledge representation languages of varying complexity for authoring ontologies. OWL has more facilities for expressing meaning and semantics than XML, RDF, and RDF-S, and thus OWL goes beyond these languages in its ability to represent machine-interpretable content on the Web. OWL is used to formally document RDF classes and properties.

---

**Sources**


*W3C website* (https://www.w3.org/).

Pizza Data → Pizza Ontology!

Anna-Sophia Zingarelli-Sweet  
Cataloging and Metadata Specialist, Oviatt Library  
California State University Northridge 
annasophiazingarellisweet@csun.edu 
@aszingarelli
Objectives

• Introduce the Protégé interface
• Demonstrate primary functionalities
• Illustrate key terms
• Make it look easy!
Protégé

- I’m using Protégé 5.1.0
- A data utility
- Provides an interface for structuring data and saving it in a variety of formats.
Source


Matthew Horridge, University of Manchester, ©2011

Based on an idea by Chris Wroe and Robert Stevens
Quick note on rights:

• Tutorial by Matthew Horridge is copyright of the University of Manchester.
• Pizza ontology is CC-BY-3.0. Contributors are Alan Rector, Chris Wroe, Matthew Horridge, Nick Drummond, and Robert Stevens.
• These slides, by me, are CC-0.
Key Terms: Not Just Jargon

• Classes and Subclasses
• Disjoint Classes
• Object Properties
• Data Properties
• Inverse Properties
• Domain
• Range
Opening from URL
This is an example ontology that contains all constructs required for the various versions of the Pizza Tutorial run by Manchester University (see http://owl.cs.manchester.ac.uk/publications/talks-and-tutorials/protege-owl-tutorial).

An ontology about pizzas and their toppings.

Annotators:
- rdfs:label [type: xsd:string]
  - pizza
- dct:title [language: en]
  - pizza
- dc:description [language: en]
  - An ontology about pizzas and their toppings.

Creative Commons Attribution 3.0 (CC BY 3.0)
Adding a comment
Creating a Class
Specifying Disjoint Classes
so an individual or object cannot be a member of both classes
Protégé doesn’t know that something can’t be both a PizzaTopping and a PizzaBase unless you tell it!
Creating Subclasses
Making sibling classes disjoint
OWL Properties

• Object properties
  – Define relationships between two individuals
  – e.g. “Anna-Sophia” hasPet “Amory”

• Data properties
  – Assigns data values to individuals
  – e.g. “Anna-Sophia” hasAge “29”^xsd:integer

• Annotation properties
Creating Object Properties
Creating Inverse Properties
Types of Object Properties

- Functional
- Transitive
- Symmetric
- Asymmetric
- Reflexive
- Irreflexive
Domains and Ranges

• Object properties link individuals, from a domain, to other individuals, from a range.
Adding a Range
Adding a Domain
Description of hasTopping

<table>
<thead>
<tr>
<th>Equivalent To</th>
<th>hasIngredient</th>
</tr>
</thead>
<tbody>
<tr>
<td>SubProperty Of</td>
<td>isToppingOf</td>
</tr>
</tbody>
</table>

Domains (intersection):
- hasTopping some Pizza

Ranges (intersection):
- hasTopping some PizzaTopping

Disjoint With

SuperProperty Of (Chain)
“Pizza” hasTopping “PizzaTopping”
Class Description
Existential Descriptions

“What is a HawaiianPizza”?
Other Tools

• Reasoners
• Inferences
Save Formats
Questions?