TU WIEN Informatics

192.161 Management of Graph Data

(4.0 VU / 6.0 ECTS)

2025W

Ontologies & OWL

Katja Hose, Maxime Jakubowski
Milos Jovanovik

mogda@list.tuwien.ac.at

Lecture Outline

- Introduction
- OWL's Relation to RDF and RDFS
- The OWL Language
- A few Practical Ontologies of Interest
  - Provenance: PROV-O
  - Medical: SNOMED CT
- Conclusion

* Slides adapted from Chapter 4 of "**Semantic Web Primer**", by Grigoris Antoniou, Paul Groth, Frank Van Harmelen, Rinke Hoekstra

Introduction

- We learned that RDFS can express subclass and property hierarchies, with domain and range definitions for properties.
- But, sometimes we need to define more expressive knowledge:
  - a person can have exactly one birth date
  - a course must have 6+ students enrolled

Requirements for Ontology Languages

- An *ontology* is an explicit formal specification of the concepts in a domain.
- Languages which express ontologies are called ontology languages.
- The main requirements are:
  - a well-defined syntax
  - a formal semantics
  - sufficient expressive power
  - convenience of expression
  - efficient reasoning support

Requirements for Ontology Languages

- Syntax
  - A well-defined syntax is necessary for machine processing of information.
- Formal Semantics
  - A formal semantics describes the meaning of a language precisely – it is not open to subjective interpretation by different people (or machines)

Requirements for Ontology Languages

- Expressivity
  - Class Membership
  - Classification
  - Equivalence and Equality
  - Disjointness and Difference
  - Boolean Combinations of Classes
  - Local Scope of Properties
  - Special Characteristics of Properties
  - Cardinality Restrictions
  - Consistency

Requirements for Ontology Languages

- Reasoning Support
  - Class membership
    - If x is an instance of a class C, and C is a subclass of D, then we can infer that x is an instance of D
  - Equivalence of classes
    - If class A is equivalent to class B, and class B is equivalent to class C, then A is equivalent to C, too

Requirements for Ontology Languages

- Reasoning Support
  - Consistency
    - X instance of classes A and B, but A and B are disjoint
    - This is an indication of an error in the ontology
  - Classification
    - Certain property-value pairs are a sufficient condition for membership in a class A; if an individual x satisfies such conditions, we can conclude that x must be an instance of A

Requirements for Ontology Languages

- Reasoning Support
  - Reasoning support is important for:
    - checking the consistency of the ontology and the knowledge;
    - checking for unintended relationships between classes;
    - automatically classifying instances in classes;
  - Checks like the preceding ones are valuable for
    - designing large ontologies, where multiple authors are involved;
    - integrating and sharing ontologies from various sources;

- Reasoning Support
  - Reasoning support is also important for:
    - generating new knowledge (new RDF triples) based on existing knowledge (RDF triples) in the graph and the ontology specification;

Reasoning: Example

- We want to get from something like this:

```
@prefix dbr: <http://dbpedia.org/resource/> .
@prefix rel: <http://purl.org/vocab/relationship/> .

dbr:Prince_William_of_Wales  rel:siblingOf  dbr:Prince_Harry_of_Wales .
dbr:Elizabeth_Bowes-Lyon  rel:ancestorOf  dbr:Elizabeth_II_of_the_United_Kingdom .
dbr:Elizabeth_II_of_the_United_Kingdom  rel:ancestorOf  dbr:Charles%2C_Prince_of_Wales .
dbr:Charles%2C_Prince_of_Wales  rel:ancestorOf  dbr:Prince_William_of_Wales .
```

Reasoning: Example

- To this:

```
@prefix dbr: <http://dbpedia.org/resource/> .
@prefix rel: <http://purl.org/vocab/relationship/> .

dbr:Prince_William_of_Wales  rel:siblingOf  dbr:Prince_Harry_of_Wales .
dbr:Elizabeth_Bowes-Lyon  rel:ancestorOf  dbr:Elizabeth_II_of_the_United_Kingdom .
dbr:Elizabeth_II_of_the_United_Kingdom  rel:ancestorOf  dbr:Charles%2C_Prince_of_Wales .
dbr:Charles%2C_Prince_of_Wales  rel:ancestorOf  dbr:Prince_William_of_Wales .

dbr:Prince_Harry_of_Wales  rel:siblingOf  dbr:Prince_William_of_Wales .

dbr:Elizabeth_Bowes-Lyon  rel:ancestorOf  dbr:Charles%2C_Prince_of_Wales .
dbr:Elizabeth_Bowes-Lyon  rel:ancestorOf  dbr:Prince_William_of_Wales .
dbr:Elizabeth_II_of_the_United_Kingdom  rel:ancestorOf  dbr:Prince_William_of_Wales .

dbr:Elizabeth_II_of_the_United_Kingdom  rel:descendantOf  dbr:Elizabeth_Bowes-Lyon .
dbr:Charles%2C_Prince_of_Wales  rel:descendantOf  dbr:Elizabeth_II_of_the_United_Kingdom .
dbr:Prince_William_of_Wales  rel:descendantOf  dbr:Charles%2C_Prince_of_Wales .
dbr:Charles%2C_Prince_of_Wales  rel:descendantOf  dbr:Elizabeth_Bowes-Lyond .
dbr:Prince_William_of_Wales  rel:descendantOf  dbr:Elizabeth_Bowes-Lyon .
dbr:Prince_William_of_Wales  rel:descendantOf  dbr:Elizabeth_II_of_the_United_Kingdom .
```

Tradeoff between Expressive Power and Efficient Reasoning Support

- The richer the language is, the more inefficient the reasoning support becomes.
- Sometimes it crosses the border of decidability.
- We need a compromise:
  - A language supported by reasonably efficient reasoners;
  - A language that can express large classes of ontologies and knowledge;

Limitations of the Expressive Power of RDF Schema

- Local scope of properties
  - rdfs:range defines the range of a property (e.g. eats) for all classes
  - In RDF Schema we cannot declare range restrictions that apply to some classes only
  - E.g. we cannot say that *“cows eat only plants, while other animals may eat meat, too”*

Limitations of the Expressive Power of RDF Schema

- Disjointness of classes
  - Sometimes we wish to say that classes are disjoint (e.g. male and female).
- Boolean combinations of classes
  - Sometimes we wish to build new classes by combining other classes using union, intersection, and complement.
  - E.g. person is the disjoint union of the classes male and female.

Limitations of the Expressive Power of RDF Schema

- Cardinality restrictions
  - E.g. a person has exactly two parents, a course is taught by at least one lecturer.
- Special characteristics of properties
  - Transitive property (like “greater than”).
  - Unique property (like “is mother of”).
  - A property is the inverse of another property (like “eats” and “is eaten by”).

Lecture Outline

- Introduction
- OWL's Relation to RDF and RDFS
- The OWL Language
- A few Practical Ontologies of Interest
  - Provenance: PROV-O
  - Medical: SNOMED CT
- Conclusion

TU WIEN Informatics

Compatibility of OWL with RDF/RDFS

- Ideally, OWL would extend RDF Schema
  - Adopting the meaning of rdfs:Class, rdfs:Property, etc.
- But, simply extending RDF Schema would work against obtaining expressive power and efficient reasoning
  - Combining RDF Schema with logic leads to uncontrollable computational properties

Two Semantics of OWL

- OWL2 Full: RDF-Based Semantics
  - The entire language is called OWL2 Full and uses all the OWL2 language primitives.
- OWL2 DL: Direct Semantics
  - In order to regain computational efficiency, OWL2 DL is mapped onto a description logic (DL).
  - Description logics are a subset of predicate logic for which efficient reasoning support is possible.
  - OWL2 DL restricts the way in which the primitives of OWL2, RDF, and RDFS may be used.

OWL2 Full: RDF-Based Semantics

- Pros:
  - It is mapped to an RDF-based semantics.
  - It is therefore both structurally and semantically fully upward-compatible with RDF:
    - any legal RDF document is also a legal OWL2 Full document, and
    - any valid RDF Schema inference is also a valid OWL2 Full conclusion.
- Cons:
  - It is that the language has become so powerful as to be undecidable, dashing any hope of complete (or efficient) reasoning support.

OWL2 DL: Direct Semantics

- Pros:
  - The advantage of its limited expressiveness is that it permits efficient reasoning support.
- Cons:
  - We lose full compatibility with RDF:
    - An RDF document will in general have to be extended in some ways and restricted in others before it is a legal OWL2 DL document.
    - However, every legal OWL2 DL document is a legal RDF document.

Compatibility of OWL with RDF/RDFS

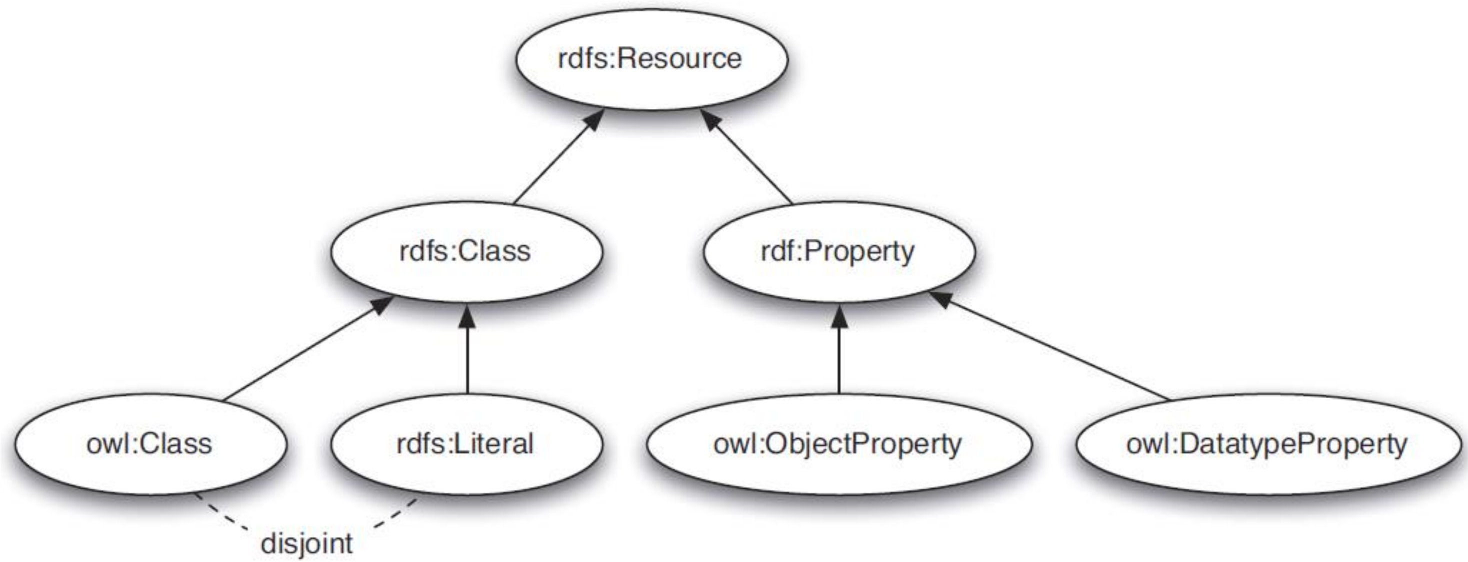


Figure 4.1: Subclass relationships between OWL2 and RDF/RDFS

Vocabularies vs. Ontologies

- Vocabularies (RDFS):
  - Terms (classes, properties) for describing data
  - Light semantics: class hierarchies, domain/range, simple inference
- Ontologies (OWL):
  - Built on vocabularies (RDFS)
  - Rich semantics: constraints, class expressions, logical axioms, advanced reasoning
- Key difference:
  - Vocabularies define terms
  - Ontologies define meaning, rules, and inferences

Lecture Outline

- Introduction
- OWL's Relation to RDF and RDFS
- The OWL Language
- A few Practical Ontologies of Interest
  - Provenance: PROV-O
  - Medical: SNOMED CT
- Conclusion

OWL Syntax

- OWL builds on RDF and can be expressed using all valid RDF syntaxes.
- Other syntactic forms for OWL have also been defined:
  - Functional-Style Syntax
  - OWL/XML
  - Manchester Syntax

Ontology Documents

- When using the Turtle syntax, OWL ontology documents, or simply *ontologies*, are just like any other RDF document.
- OWL ontologies introduce these namespaces:

```
@prefix owl:  <http://www.w3.org/2002/07/owl#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .
```

Ontology Documents

- An OWL ontology starts with a collection of assertions for *housekeeping* purposes.
- These assertions introduce a base namespace, the ontology itself, its name, possible comments, version control, and inclusion of other ontologies.

Ontology Documents

```
@prefix : <http://www.semanticwebprimer.org/ontologies/apartments.ttl#> .
@prefix dbpedia-owl: <http://dbpedia.org/ontology/> .
@prefix dbpedia:     <http://dbpedia.org/resource/> .
@base       <http://www.semanticwebprimer.org/ontologies/apartments.ttl> .

<http://www.semanticwebprimer.org/ontologies/apartments.ttl>
    rdf:type       owl:Ontology ;
    rdfs:label     "Apartments Ontology"^^xsd:string ;
    rdfs:comment   "An example OWL2 ontology"^^xsd:string ;
   owl:versionIRI <http://www.semanticwebprimer.org/ontologies/apartments.ttl#1.0> ;
    owl:imports    <http://dbpedia.org/ontology/> ;
    owl:imports    <http://dbpedia.org/resource/> .
```

Ontology Documents

- Our apartments ontology imports all axioms defined in the DBpedia ontology (http://dbpedia.org/ontology/), as well as everything in DBpedia itself (http://dbpedia.org/resource/).
- The owl:imports property is *transitive*.

Property Types

- OWL distinguishes two types of properties:
  - Object properties: relate individuals to other individuals;
  - Datatype properties: relate individuals to literal values of a certain data type;

Property Types

- Object property

```
:rents rdf:type            owl:ObjectProperty ;
       rdfs:domain         :Person ;
       rdfs:range          :Apartment ;
       rdfs:subPropertyOf  :livesIn .
```

Property Types

- Datatype property

```
:age rdf:type   owl:DatatypeProperty ;
    rdfs:range xsd:nonNegativeInteger .
```

Other Property Types

- Transitive Properties
  - greater than, is part of, …
- Symmetric and Asymmetric Properties
  - spouse of, based near, same as, similar to, …
  - larger than, mentors, directs, …
- Functional and Inverse-Functional Properties
  - an instance can have only one value (age, height, student ID, …)
  - it can be the value of only one instance (has room, …)
- Reflexive and Irreflexive Properties
  - Each instance is in this relation with itself (is part of, …)
  - Each instance is not in this relation with itself (rents, …)

- Transitive Properties

```
:isPartOf rdf:type owl:ObjectProperty ;
          rdf:type owl:TransitiveProperty .
```

Other Property Types

- Symmetric and Asymmetric Properties

```
:isAdjacentTo  rdf:type owl:ObjectProperty ;
               rdf:type owl:SymmetricProperty .

:isCheaperThan rdf:type owl:ObjectProperty ;
               rdf:type owl:AsymmetricProperty ;
               rdf:type owl:TransitiveProperty .
```

Other Property Types

- Functional and Inverse-Functional Properties

```
:hasNumberOfRooms rdf:type owl:DatatypeProperty ;
                  rdf:type owl:FunctionalProperty .
```

  - If two apartments, a_1 and a_2, are related via :hasRoom to the same room r, a reasoner will infer that they are *the same*.

```
:hasRoom          rdf:type owl:ObjectProperty ;
                  rdf:type owl:InverseFunctionalProperty .
```

Other Property Types

- Reflexive and Irreflexive Properties

```
:isPartOf    rdf:type owl:ObjectProperty ;
             rdf:type owl:ReflexiveProperty .

:rents       rdf:type owl:ObjectProperty ;
             rdf:type owl:IrreflexiveProperty .
```

Property Axioms

- In addition to these property types, we can specify additional characteristics of properties in terms of how they relate to classes and other properties.
- Some of these are familiar from RDF Schema; others are completely new.

Property Axioms

- Domain and Range
  - OWL treats domain and range for properties is exactly the same as in RDF Schema.
  - If more than one rdfs:range or rdfs:domain is asserted for a property, the actual range or domain is the *intersection* of the classes specified.

Property Axioms

- Inverse Properties
  - A typical example is the pair :rents and :isRentedBy

```
:isRentedBy rdf:type      owl:ObjectProperty ;
            owl:inverseOf :rents .
```

  - A reasoner will determine that our two individuals p and m have the relation *m :isRentedBy p* in addition to *p :rents m*.

Property Axioms

- Equivalent Properties
  - Every two individuals related via a property will always be related via its equivalent, and vice versa.
  - It’s a convenient mechanism for mapping elements of different ontologies to each other.

```
:isPartOf  rdf:type                owl:ObjectProperty ;
           owl:equivalentProperty dbpedia:partOf .
```

- Disjoint Properties
  - No two individuals related via one property can be related via the other: the sets of pairs of individuals for which the properties can hold are *disjoint.*

```
:rents rdf:type             owl:ObjectProperty ;
        rdfs:domain         :Person ;
        rdfs:range          :Apartment ;
        owl:disjointProperty :owns .
```

- Property Chains
  - Sometimes it is useful to specify shortcuts along the graph of properties relating various individuals
  - For instance, if we know that :Paul :rents the :BaronWayApartment, and that the :BaronWayApartment :isPartOf the :BaronWayBuilding, for which the dbpedia:location is dbpedia:Amsterdam, we know that :Paul must have a :livesIn relation with :Amsterdam.

```
:livesIn rdf:type               owl:ObjectProperty ;
         owl:propertyChainAxiom ( :rents :isPartOf :location ) .
```

Property Axioms

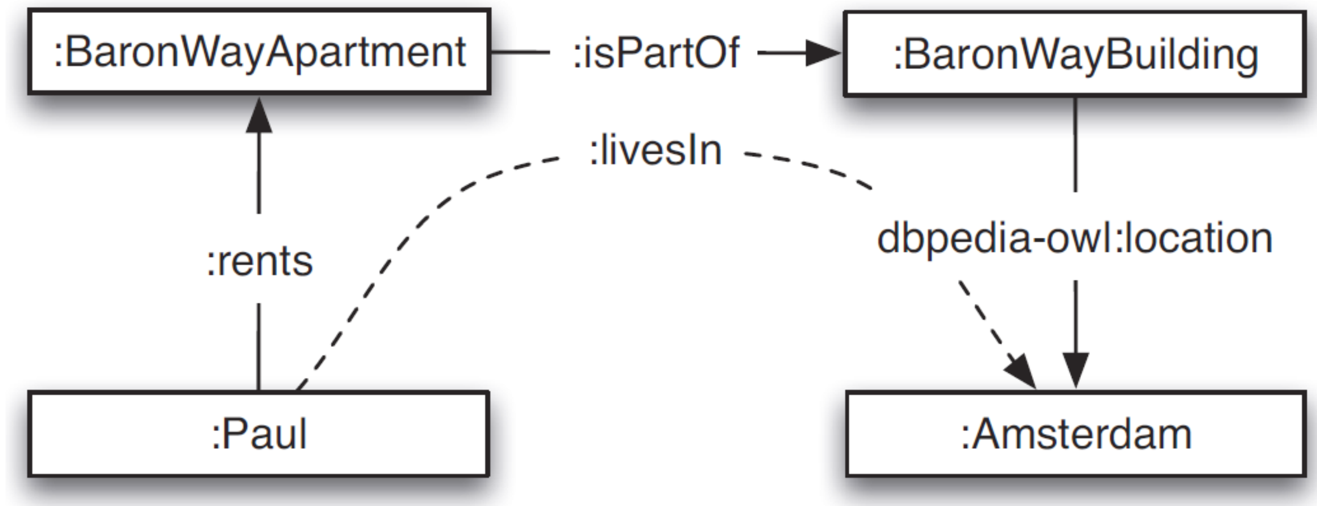


Figure 4.2: Property chains (dotted lines are inferred by the reasoner)

Class Axioms

- Most general class: owl:Thing.
  - Every owl:Class is a subclass of it.
- Empty class: owl:Nothing.
  - Every owl:Class is a superclass of it.
- They are important for reasoning, as we’ll see later.

Class Axioms

- Subclass Relations
  - Subclass relations are defined as in RDF Schema.
  - For example, we can define a class :LuxuryApartment as follows:

```
:LuxuryApartment rdf:type        owl:Class ;
                 rdfs:subClassOf :Apartment .
```

Class Axioms

- Class Equivalence
  - Equivalence of classes means that every member of a class must also be a member of the equivalent class, and vice versa.
  - In other words, both classes cover *exactly the same* set of individuals.

```
:Apartment      owl:equivalentClass dbpedia:Apartment .
```

Class Axioms

- Enumerations
  - The most straightforward way to define a class is by explicitly enumerating all individuals it consists of.
    - Inexpressive
    - Computationally expensive

```
:BaronWayRooms rdf:type      owl:Class;
          owl:oneOf     ( :BaronWayKitchen
                          :BaronWayBedroom1
                          :BaronWayBedroom2
                          :BaronWayBedroom3
                          :BaronWayLivingroom
                          :BaronWayBathroom
                          ... ) .
```

Class Axioms

- Disjoint Classes
  - Disjointness of classes means that no member of one class can also be a member of the other class.
  - The sets of individuals described by the classes do not overlap.

```
:LuxuryApartment owl:disjointWith :ColdWaterFlat .
```

- Complement
  - The complement *C* of a class *A* is the class of all things not belonging to *A*.
  - In other words, the union of *A* and *C* is equivalent to owl:Thing.

```
:FurnishedApartment rdfs:subClassOf  :Apartment .

:UnFurnishedApartment  rdfs:subClassOf  :Apartment ;

                       owl:complementOf :FurnishedApartment .
```

Class Axioms

- Union
  - We often know for some class that it is equivalent to two or more other classes: every member of the class is a member of at least one of the classes in the union.

```
:Apartment   rdf:type     owl:Class ;
             owl:unionOf  ( :ColdWaterFlat
                            :LuxuryApartment
                            :PenthouseApartment
                            :StudioApartment
                            :BasementApartment
                            :FurnishedApartment
                            :UnFurnishedApartment
                          ) .
```

Class Axioms

- Disjoint Union
  - In many cases, the member classes of the union are mutually disjoint, so we can use:

```
:Apartment    rdf:type              owl:Class;
          owl:disjointUnionOf (
                                  :FurnishedApartment
                                  :UnFurnishedApartment ) .
```

Class Axioms

- Intersection
  - We can state that a class is exactly the *intersection* of two or more other classes: every member of the class is a member of each of the classes in the intersection.

```
:LuxuryApartment
        rdf:type          owl:Class ;
        owl:intersectionOf ( :GoodLocationApartment
                             :LargeApartment
                             :NiceViewApartment
                             :LuxuryBathroomApartment ) .
```

Class Axioms on Properties

- OWL allows for more fine-grained control of class definitions than what we’ve seen so far.
  - We can specify additional class axioms that restrict the set of individuals that may be considered to be a member of a class by looking at their properties.
  - This allows us, for instance, to automatically infer class membership.
  - Class restriction axioms are attached to an owl:Class by relating them to a special type of anonymous class (owl:Restriction) that collects all individuals that satisfy the restriction.

Class Axioms on Properties

- Universal Restrictions
  - A universal restriction on a class C and property p states that for every member of C *all* values of p belong to a certain class.
  - In other words, the universal restriction can be used to specify a range for a property that is *local* to the restricted class.

```
:LuxuryBathroomApartment
        rdf:type          owl:Class;
        rdfs:subClassOf  [ rdf:type           owl:Restriction;
                           owl:onProperty     :hasBathroom ;
                           owl:allValuesFrom  :LuxuryBathroom
                         ] .
```

Class Axioms on Properties

- Existential Restrictions
  - An existential restriction on a class C and property p states that for every member of C there exists *at least some* value for p that belongs to a certain class.

```
:LuxuryBathroomApartment
        rdf:type          owl:Class;
        rdfs:subClassOf  [ rdf:type          owl:Restriction;
                           owl:onProperty     :hasBathroom ;
                           owl:someValuesFrom :LuxuryBathroom
                         ] .
```

Class Axioms on Properties

- Value Restrictions
  - Value restrictions come in handy when we want to define a class based on relations with known individuals, or specific values for datatype properties.

```
:AmsterdamApartment
    rdf:type              owl:Class;
    owl:equivalentClass [ rdf:type        owl:Restriction;
                          owl:onProperty  dbpedia-owl:location ;
                          owl:hasValue    dbpedia:Amsterdam
                        ] .
```

Class Axioms on Properties

- Cardinality Restrictions
  - A cardinality restriction constrains the number of values a certain property may have for a class.
  - If we additionally specify the class these values need to belong to, the restriction is said to be *qualified*.

```
:StudioApartment
        rdf:type          owl:Class;
        rdfs:subClassOf  [ rdf:type         owl:Restriction;
                           owl:onProperty   :hasRoom ;
                           owl:cardinality  "1"^^xsd:integer
                         ] .
```

Class Axioms on Properties

- Data Range Restrictions and Datatypes
  - Universal and existential restrictions on datatype properties allow members of a class to have *any* value from the specified datatype as value for the property.
  - Sometimes we need more precise definitions to define, for instance, the class of adults who can rent apartments, or the minimum size of apartments.
    - This example defines :Adult as the subclass of persons that have a value for the :hasAge that falls within the range of integers equal to or larger than 18.
    - You can see that the data range is defined as an anonymous class of type rdfs:Datatype.

```
:Adult    rdfs:subClassOf   dbpedia:Person ;
          rdfs:subClassOf [ rdf:type      owl:Restriction ;
                            owl:onProperty :hasAge ;
                            owl:someValuesFrom
                             [ rdf:type      rdfs:Datatype ;
                               owl:onDatatype xsd:integer ;
                               owl:withRestrictions (
                                [ xsd:minInclusive "18"^^xsd:integer ]
                               )
                             ]
                          ] .
```

Class Axioms on Properties

- Keys
  - Databases typically use keys to identify records in a table.
  - OWL allows us to indicate that for certain classes (read: tables) the value of a specific datatype property (or combination of properties) should be regarded as a *unique identifier*.

Class Axioms on Properties

- Keys
  - For example, the combination of postcode and street address number will provide a unique identifier for any dwelling in the Netherlands:

```
:postcode       rdf:type    owl:DatatypeProperty .
:addressNumber  rdf:type    owl:DatatypeProperty .

:Dwelling
        rdf:type    owl:Class ;
        owl:hasKey ( :postcode :addressNumber ) .
```

Individual Facts

- Now that we have a general idea of how we define properties and classes in OWL, we can turn our attention to the individual entities governed by our model.
- In many cases we already have a lot of knowledge about these entities and only need class axioms to infer extra information.

- Creating the individual entities, we get:
  - *ontology + data --> Knowledge Graph*

Individual Facts

- Class and Property Assertions
  - Class membership and property assertions in OWL are stated in the same way as in RDF Schema:

```
:BaronWayApartment rdf:type           :Apartment ;
                   :hasNumberOfRooms "4"^^xsd:integer ;
                   :isRentedBy        :Paul .
```

- Identity Assertions
  - Because OWL has the open world assumption, we can never assume that two individuals with different URIs must be different entities.
  - We might be dealing with a single individual that has multiple names.

Individual Facts

- Identity Assertions
  - Although we have seen that in some cases we can infer identity relations automatically, it is often more convenient to state them explicitly.

```
:BaronWayApartment owl:sameAs        :PaulsApartment ;
                   owl:differentFrom :FranksApartment .
```

Individual Facts

- Identity Assertions
  - The list of different individuals can easily grow quite long.
  - Then we use:

```
_:x  rdf:type     owl:AllDifferent ;
     owl:members  ( :FranksApartment :PaulsApartment ) .
```

Reasoning: Example (with OWL)

- From this RDF graph:

```
@prefix dbr: <http://dbpedia.org/resource/> .
@prefix rel: <http://purl.org/vocab/relationship/> .

dbr:Prince_William_of_Wales  rel:siblingOf  dbr:Prince_Harry_of_Wales .
dbr:Elizabeth_Bowes-Lyon  rel:ancestorOf  dbr:Elizabeth_II_of_the_United_Kingdom .
dbr:Elizabeth_II_of_the_United_Kingdom  rel:ancestorOf  dbr:Charles%2C_Prince_of_Wales .
dbr:Charles%2C_Prince_of_Wales  rel:ancestorOf  dbr:Prince_William_of_Wales .
```

Reasoning: Example (with OWL)

- We get to this RDF graph:

```
@prefix dbr: <http://dbpedia.org/resource/> .
@prefix rel: <http://purl.org/vocab/relationship/> .

dbr:Prince_William_of_Wales  rel:siblingOf  dbr:Prince_Harry_of_Wales .
dbr:Elizabeth_Bowes-Lyon  rel:ancestorOf  dbr:Elizabeth_II_of_the_United_Kingdom .
dbr:Elizabeth_II_of_the_United_Kingdom  rel:ancestorOf  dbr:Charles%2C_Prince_of_Wales .
dbr:Charles%2C_Prince_of_Wales  rel:ancestorOf  dbr:Prince_William_of_Wales .

dbr:Prince_Harry_of_Wales  rel:siblingOf  dbr:Prince_William_of_Wales .

dbr:Elizabeth_Bowes-Lyon  rel:ancestorOf  dbr:Charles%2C_Prince_of_Wales .
dbr:Elizabeth_Bowes-Lyon  rel:ancestorOf  dbr:Prince_William_of_Wales .
dbr:Elizabeth_II_of_the_United_Kingdom  rel:ancestorOf  dbr:Prince_William_of_Wales .

dbr:Elizabeth_II_of_the_United_Kingdom  rel:descendantOf  dbr:Elizabeth_Bowes-Lyon .
dbr:Charles%2C_Prince_of_Wales  rel:descendantOf  dbr:Elizabeth_II_of_the_United_Kingdom .
dbr:Prince_William_of_Wales  rel:descendantOf  dbr:Charles%2C_Prince_of_Wales .
dbr:Charles%2C_Prince_of_Wales  rel:descendantOf  dbr:Elizabeth_Bowes-Lyond .
dbr:Prince_William_of_Wales  rel:descendantOf  dbr:Elizabeth_Bowes-Lyon .
dbr:Prince_William_of_Wales  rel:descendantOf  dbr:Elizabeth_II_of_the_United_Kingdom .
```

Reasoning: Example (with OWL)

- Using RDF reasoning based on this OWL ontology:

```
rel:siblingOf rdf:type owl:ObjectProperty ;
              rdf:type owl:SymmetricProperty ;
              rdfs:comment "A person having one or both parents in common with this person." ;
              rdfs:domain foaf:Person ;
              rdfs:range foaf:Person .

rel:ancestorOf rdf:type owl:ObjectProperty ;
               rdf:type owl:TransitiveProperty ;
               rdfs:comment "A person who is a descendant of this person." ;
               rdfs:domain foaf:Person ;
               rdfs:range foaf:Person ;
               owl:inverseOf rel:descendantOf .

rel:descendantOf rdf:type owl:ObjectProperty ;
                 rdf:type owl:TransitiveProperty ;
                 rdfs:comment "A person from whom this person is descended." ;
                 rdfs:domain foaf:Person ;
                 rdfs:range foaf:Person ;
                 owl:inverseOf rel:ancestorOf .
```

Lecture Outline

- Introduction
- OWL's Relation to RDF and RDFS
- The OWL Language
- A few Practical Ontologies of Interest
  - Provenance: PROV-O
  - Medical: SNOMED CT
- Conclusion

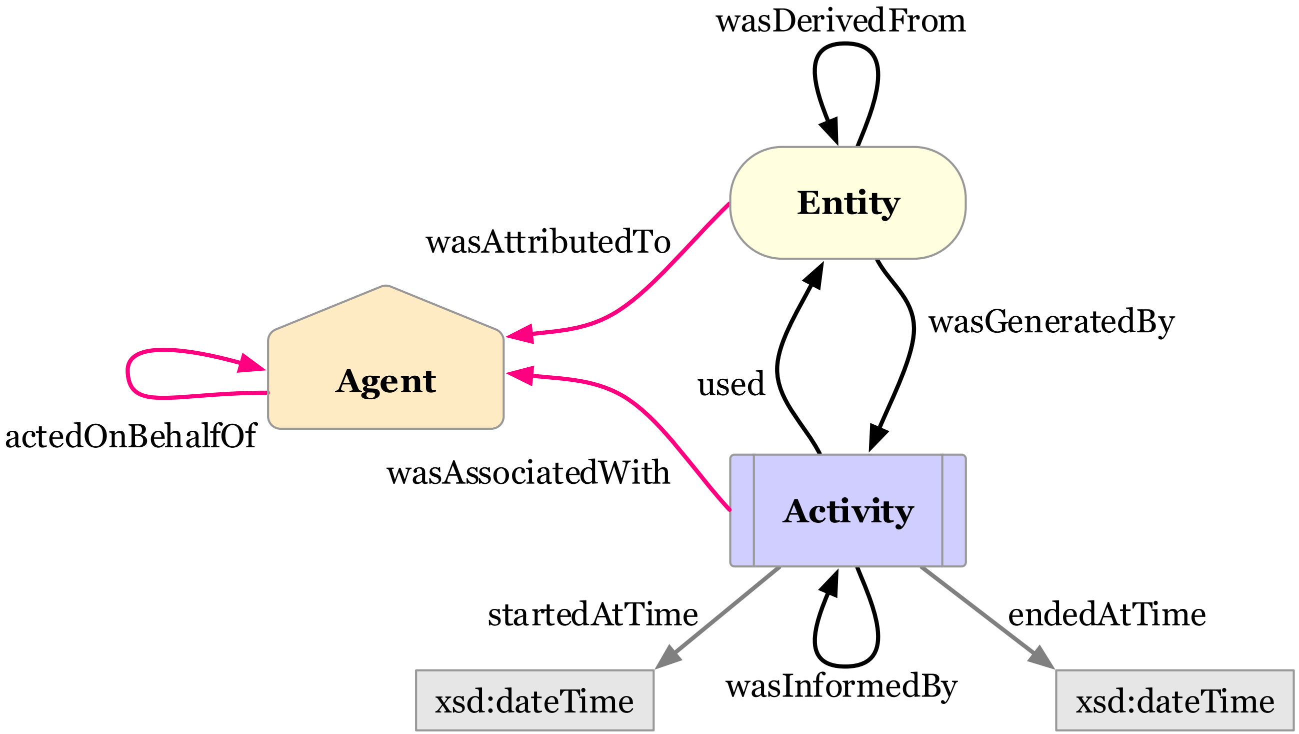
The PROV-O Ontology

- PROV-O
  - The de-facto ontology for provenance
  - It’s a lightweight ontology that can be adopted in a wide range of applications
  - Developed as a W3C recommendation
  - https://www.w3.org/TR/prov-o/

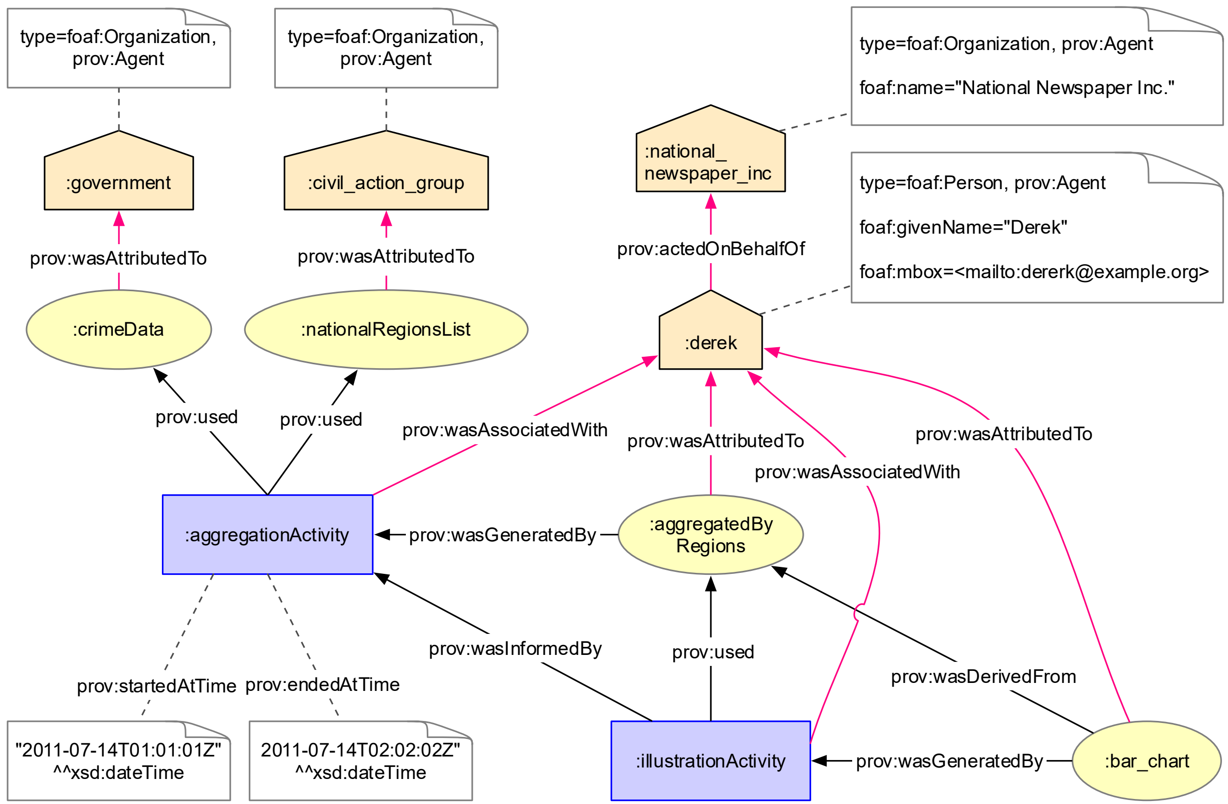
- *“Provenance is defined as a record that describes the people, institutions, entities, and activities involved in producing, influencing, or delivering a piece of data or a thing” – W3C Provenance Incubator Group*

TU WIEN Informatics

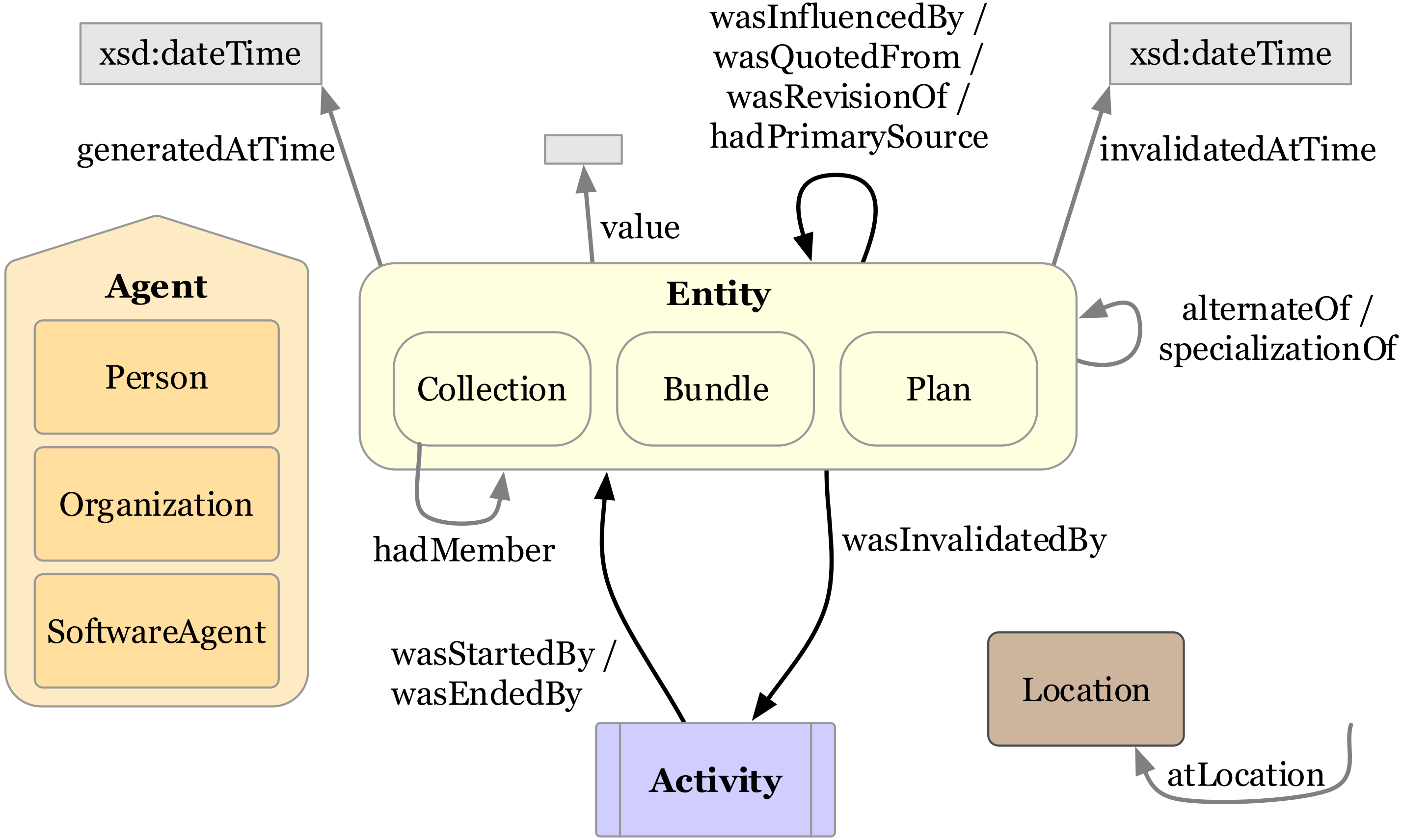
The PROV-O Ontology: Starting Point



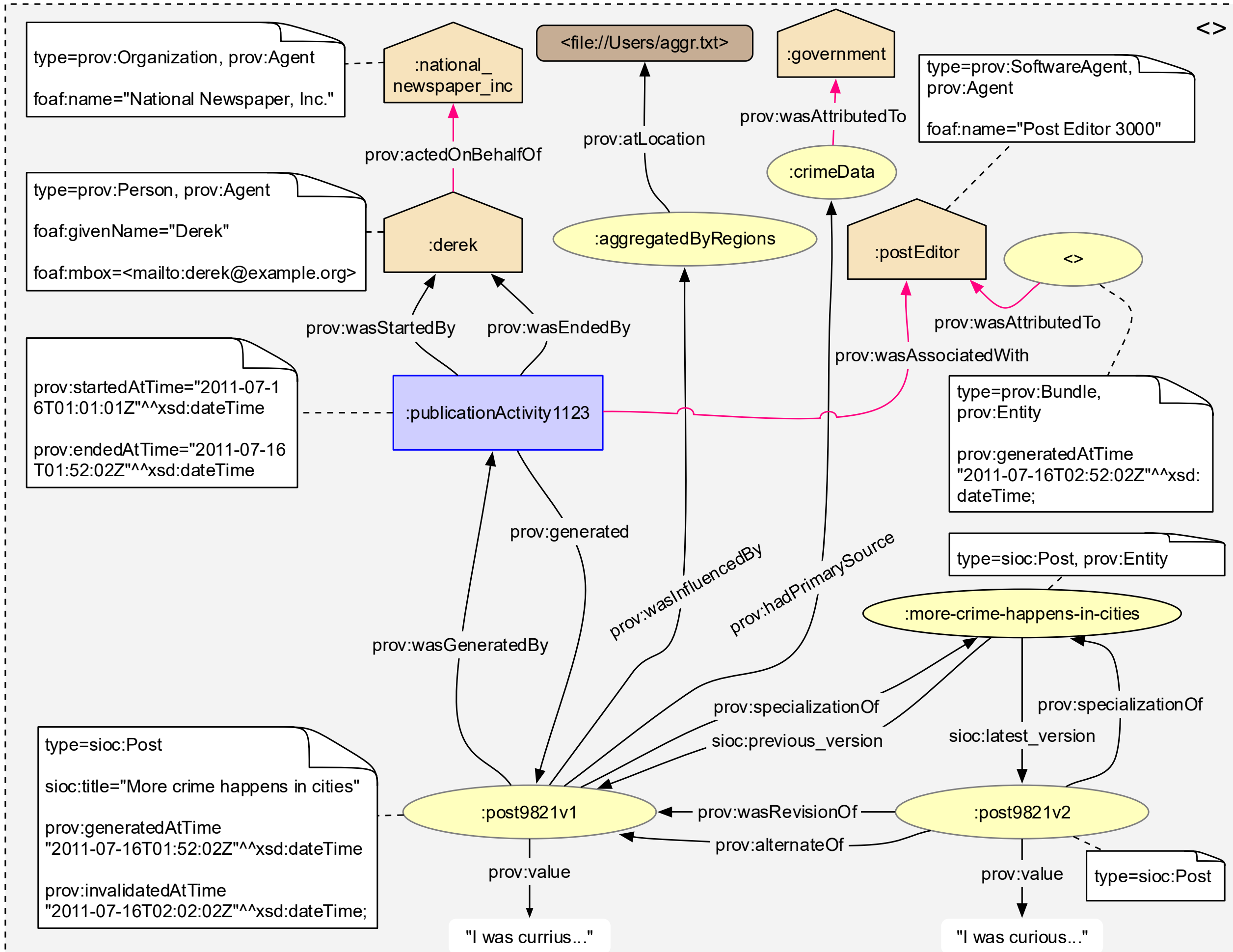
The PROV-O Ontology: Example 1



The PROV-O Ontology: Expanded Terms



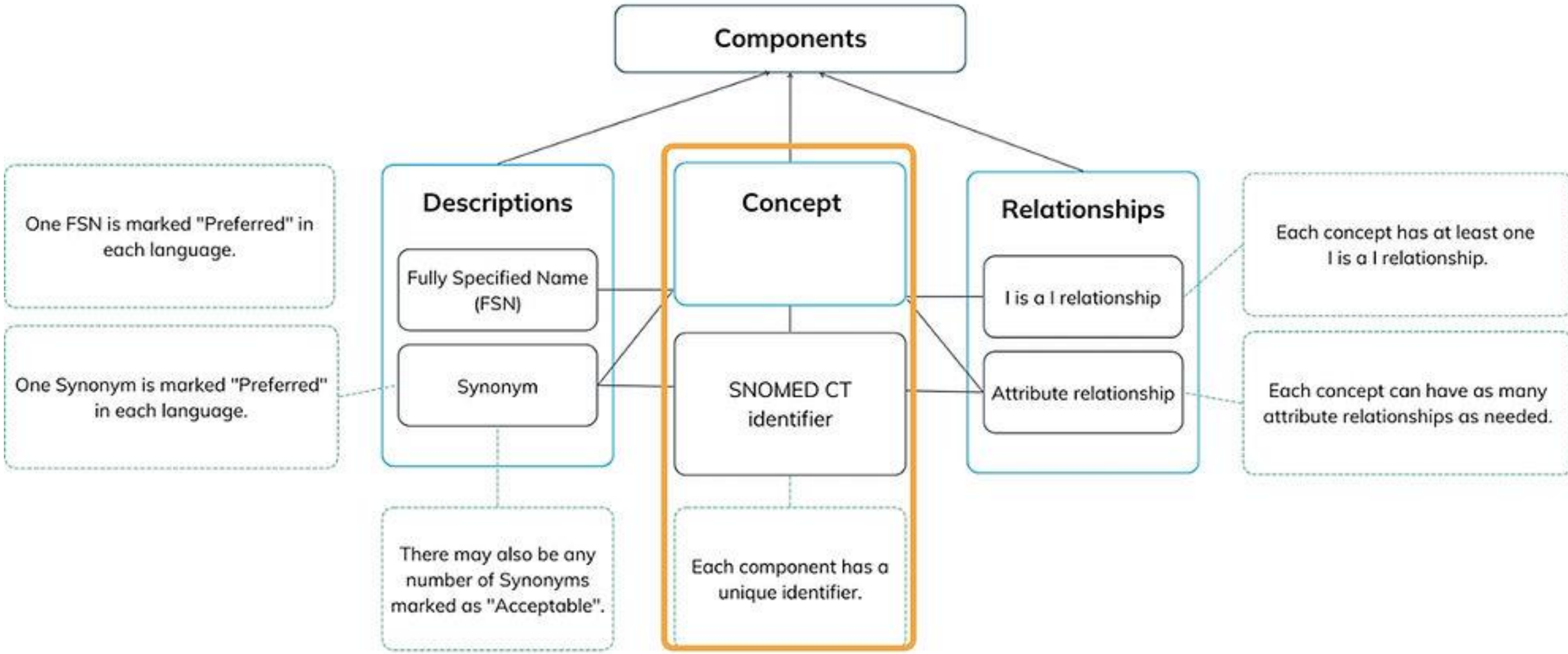
The PROV-O Ontology: Example 2



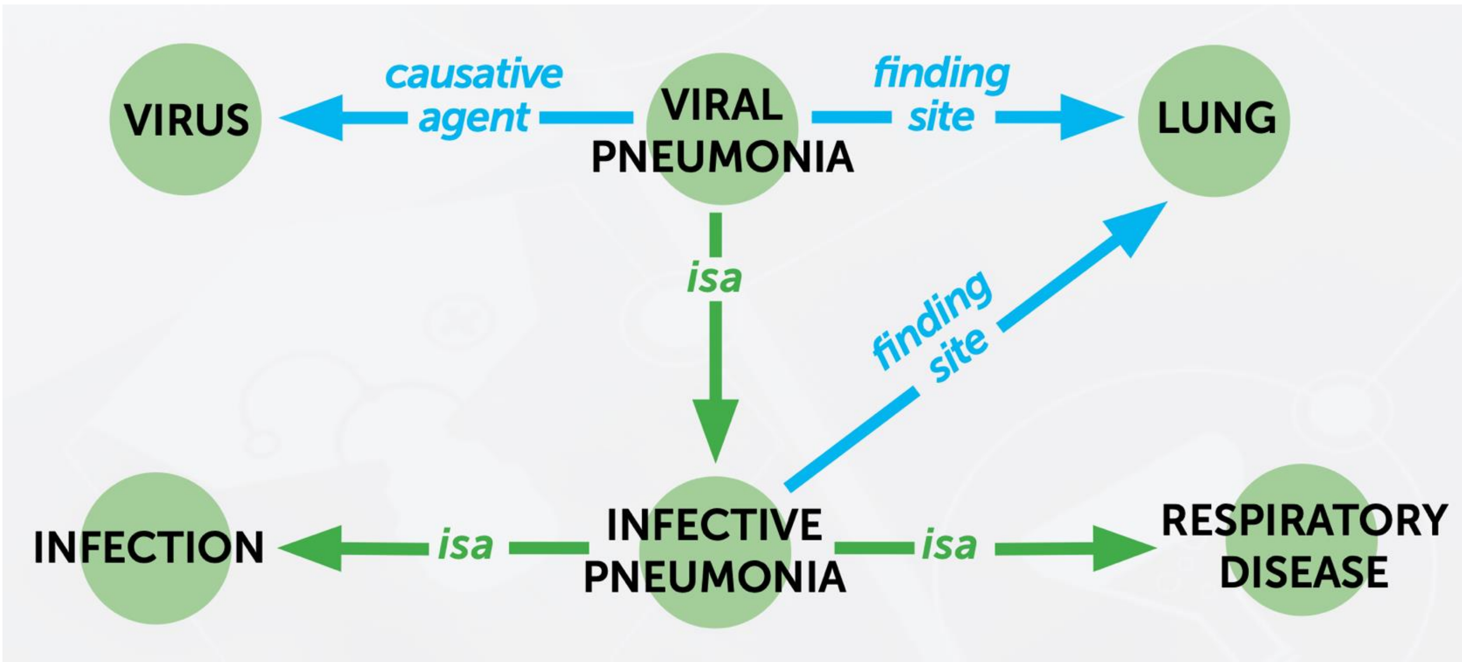
The SNOMED CT Ontology

- SNOMED CT
  - Provides the core general terminology for the electronic health records (EHR)
  - https://bioportal.bioontology.org/ontologies/SNOMEDCT
  - Includes:
    - over 300,000 unique *concepts*
    - over 1,000,000 *descriptions*, including synonyms that can be used to refer to a concept
    - over 900,000 links or semantic *relationships* between the concepts

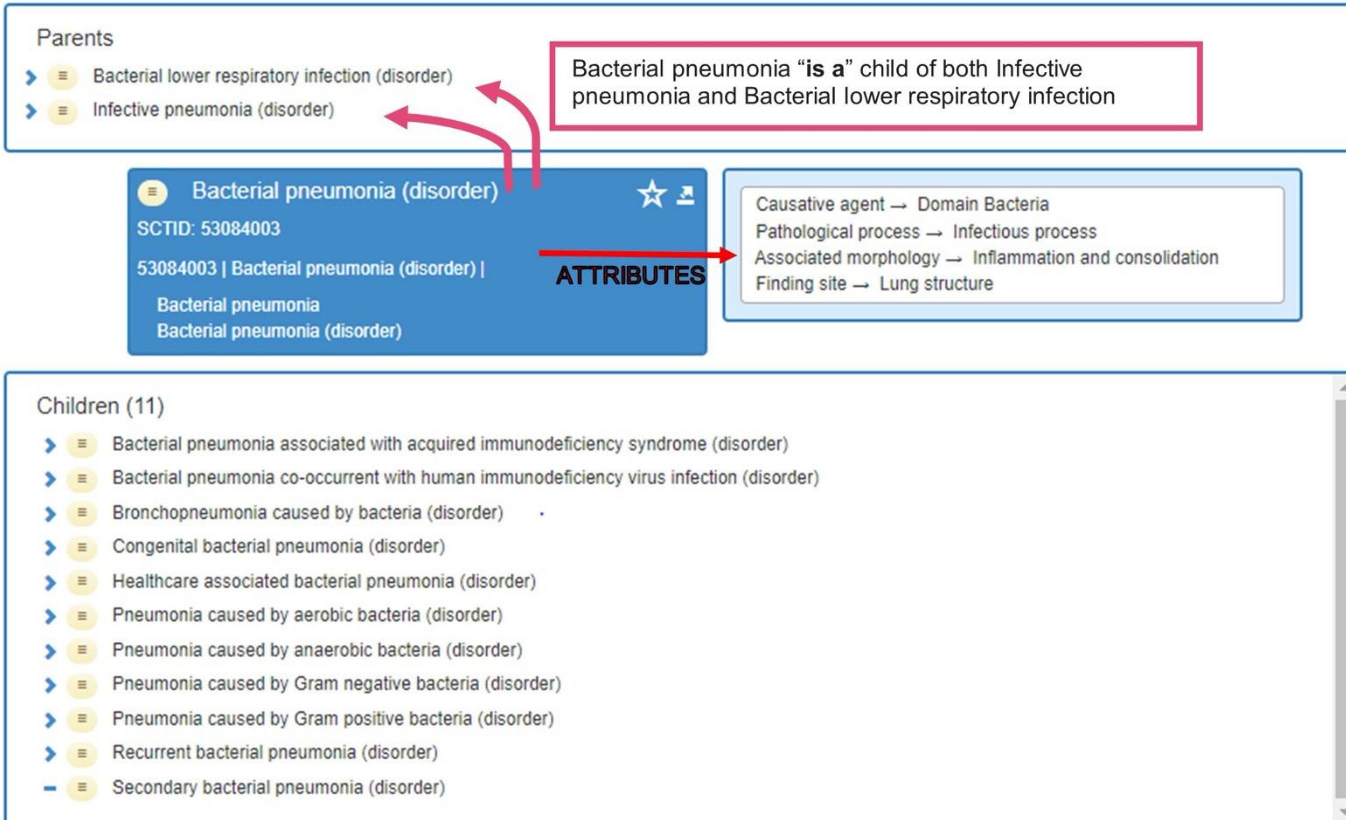
SNOMED CT: Components



SNOMED CT: Example 1



SNOMED CT: Example 2



Lecture Outline

- Introduction
- OWL’s Relation to RDF and RDFS
- The OWL Language
- A few Practical Ontologies of Interest
  - Provenance: PROV-O
  - Medical: SNOMED CT
- Conclusion

Summary

- OWL is the W3C standard for Web ontologies
- OWL extends RDF and RDF Schema with a number of very expressive language features, such as cardinality constraints, class equivalence, intersection, and disjunction.
- Formal semantics and reasoning support is provided through the correspondence of OWL with logics.
- OWL comes in two flavors:
  - OWL2 DL is a language that imposes some restrictions on the combination of OWL2 and RDFS language elements to retain decidability.
  - OWL2 Full is a fully compatible extension of RDF Schema with all OWL2 language features, but it is known to be undecidable.

Questions?