Ontology Design Patterns
Questions

• How can we represent an ordered list?
  – E.g. want to describe a bus route, how can we represent the sequence of stops?

• How can we add information to a relation (property)?
  – E.g. need to set a confidence value to the relation

• How do we represent lists of values?
  – E.g. a fixed list of airline models
Topics

• N-ary relations
  – How can we say more about a relation instance?

• Classes as property values
  – What do we do if we need to use a Class as a property value?

• Value partitions and value sets
  – How do we represent a fixed list of values?
Topics

• N-ary relations

• Classes as property values

• Value partitions and Value sets
Binary Relations

- In RDF and OWL, binary relations link two individuals, or an individual and a value.

- The properties year-of-birth and father-of are binary relations.

\[\text{Holbein the Elder} \xrightarrow{\text{year-of-birth}} 1460 \xleftarrow{\text{father-of}} \text{Holbein the Younger}\]
Relations with additional info

• In some cases, we need to associate additional info with a binary relation
  – Eg certainty, strength, dates

• For example, Holbein the Elder’s date of birth is unconfirmed
  – He was born in either 1460 or 1465
  – How can we represent this uncertainty?
N-ary Relations

- N-ary relations link an individual to more than a single individual or value
- Use cases:
  1. A relation needs additional info
     - eg a relation with a rating value
  2. Two binary relations are related to each other
     - eg body_temp (high, normal, low), and trend (rising, falling)
  3. A relation between several individuals
     - eg someone buys a book from a bookstore
  4. Linking from, or to, an ordered list of individuals
     - eg an airline flight visiting a sequence of airports

- Pattern 1: Creating a new class or relation
  - Use for cases 1, 2, and 3 above
- Pattern 2: Sequence of arguments
  - For case 4
N-ary relation - Pattern 1: Creating a new class or relation

- To represent additional information about a relation:
  - We can create a new class to represent the relation
  - The individuals of this class are instances of the relation
  - This class can have additional properties to describe more information about the relation
Use case 1: additional information about a relation

- Jack has given the film ‘I Am Legend’ a rating of 8
- We need to represent a quantitative value to describe the rating relation

- **What is wrong with this representation?**
  - What will happen when Jack rates other films?
Solution for use case 1

Jack issued_rating _:Rating_1

rated_object

_.Rating_1 rating 8

Film

rated_object (someValuesFrom, functional)

Person issued_rating (allValuesFrom)

Rating_Relation

rating_value (allValuesFrom, functional)

Rating

I am Legend

School of Electronics and Computer Science

rated_object
Use this icon to create anonymous instances.
Use case 2: different aspects of the same relation

- *Steve has temperature, which is high, but falling*
- We need to represent different aspects of the temperature that Steve has
http://www.w3.org/TR/swbp-n-aryRelations/temperature.rdf
Use case 3: N-ary relation with no distinguished participant

- John buys a “Lenny the Lion” book from books.example.com for $15 as a birthday gift

- No distinguished subject for the relation
  - i.e. no primary relation to convert into a Relation Class as in cases 1 and 2
Solution for use case 3
<rdfs:Class rdf:ID="Purchase"/>
<rdfs:Class rdf:ID="Person"/>
<rdfs:Class rdf:ID="Company"/>
<rdfs:Property rdf:ID="has_seller">  
  <rdfs:range rdf:resource="#Company"/>  
  <rdfs:domain rdf:resource="#Purchase"/>
</rdfs:Property>
<rdfs:Property rdf:ID="has_object">  
  <rdfs:range rdf:resource="#Object"/>  
  <rdfs:domain rdf:resource="#Purchase"/>
</rdfs:Property>
<rdfs:Property rdf:ID="has_buyer">  
  <rdfs:domain rdf:resource="#Purchase"/>  
  <rdfs:range rdf:resource="#Person"/>
</rdfs:Property>
<rdfs:Property rdf:ID="has_purpose">  
  <rdfs:domain rdf:resource="#Purchase"/>  
  <rdfs:range rdf:resource="#Purpose"/>
</rdfs:Property>
<rdfs:Property rdf:ID="has_amount">  
  <rdfs:domain rdf:resource="#Purchase"/>  
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
</rdfs:Property>
<Person rdf:ID="John"/>
<Purchase rdf:ID="Purchase_1">  
  <has_amount rdf:datatype="http://www.w3.org/2001/XMLSchema#float">15.0</has_amount>  
  <has_object>
    <Book rdf:ID="Lenny_The_Lion"/>
  </has_object>  
  <has_buyer rdf:resource="#John"/>
  <has_seller rdf:resource="http://books.example.com"/>
  <has_purpose>
    <Purpose rdf:ID="Birthday_Gift"/>
  </has_purpose>  
</Purchase>
N-ary Relations - Pattern 2: Sequence of arguments

- *United Airlines, flight 1377 visits the following airports: LAX, DFW, and JFK*

- For such an example, we need to represent a sequence of arguments

Source: W3C
N-ary Relations - Pattern 2: Sequence of arguments

- This is the OWL:Lite ontology to represent a sequence

```owl
:FinalFlightSegment a owl:Class ;
  rdfs:comment "The last flight segment has no next_segment";
  rdfs:subClassOf :FlightSegment ;
  rdfs:subClassOf  
    [ a owl:Restriction ; owl:maxCardinality "0"; 
```

Source: W3C
Topics

• N-ary relations

• Classes as property values

• Value partitions and Value sets
Classes as property values

- In some cases, it is convenient to put a class as a value of some property

- Classes can be property values in RDFS and OWL Full, with no restrictions

- In OWL DL and OWL Lite, classes cannot be property values
  - Because nothing can be both a class and an individual
  - Need to use alternative mechanisms
Use case example

- Represent two books about lions, one is about the species of lion, and the other about the species of African lion
- Retrieve both books when asking for books about lions
Approach 1: Use classes directly as property values

- The property dc:subject has the Animal classes as values

Source: W3C
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>  
PREFIX base: <http://www.ecs.soton.ac.uk/teaching/COMP3028/book1.owl#>

SELECT ?book
        ?subject rdfs:subClassOf base:Lion}
Notes on Approach 1

• This approach is the most intuitive

• Resulting ontology is compatible with RDFS and OWL Full, but not OWL DL or OWL Lite

• The subjects are in a hierarchy (AfricanLion isA Lion isA Animal)
  – Application can use this hierarchy to find books about Lion as well as books about its sub-subject; AfricanLion

• Good approach if:
  – Want to keep things simple
  – Don’t mind being in OWL Full
  – Don’t mind using the class hierarchy as book subject
Approach 2: Using special instances

- Use instances of classes as property values
Approach 2
SPARQL Query

PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dc:  <http://purl.org/dc/elements/1.1/>  
PREFIX base:  <http://protege.stanford.edu/swbp/books2.owl#>

SELECT  ?book  
WHERE  {  
  ?subject rdf:type base:Lion  
}  

<http://protege.stanford.edu/swbp/books2.owl#LionsLifeInThePrideBook>  
<http://protege.stanford.edu/swbp/books2.owl#AfricanLionBook>
Notes on Approach 2

- Classes are not used as values directly
  - Using their instances as property values instead
- Ontology is compatible with OWL DL and OWL Lite
- We used the class Lion for the subject lion
  - Need a different one to refer to actual lions!
  - Shouldn’t use the same concept for two conceptually different things
  - We need to be extra careful if the Animal ontology is important
    - Changing the meaning of classes may cause some interpretation problems
- No direct relation between the subjects
  - But the instance AfricanLionSubject is also an instance of Lion
- Use this approach if:
  - Want to stick to OWL DL or OWL Lite
  - Won’t be changing the original meaning of any of the classes
  - Not concerned with the subjects not having direct links
Approach 3: Using a parallel instance hierarchy

- Create a separate subject class
Approach 3

http://www.w3.org/TR/swbp-classes-as-values/books3.owl
SPARQL Query

- One way of querying this model is by using the seeAlso annotation property.
- You can also query the transitive parentSubject property.

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX base: <http://protege.stanford.edu/swbp/books3.owl#>

SELECT ?book
    ?subject rdfs:seeAlso ?class .
    ?class rdfs:subClassOf base:Lion
}
```

<http://protege.stanford.edu/swbp/books3.owl#LionsLifeInThePrideBook>
<http://protege.stanford.edu/swbp/books3.owl#AfricanLionBook>
Approach 3

- Compatible with OWL DL and OWL Lite
  - Using classes as values for annotation properties (e.g., rdfs:seeAlso) does not change OWL DL compatibility

- The subject hierarchy can be recreated using the parentSubject
  - This property is transitive
  - Most reasoners can infer the parentSubject transitive property
    - But they won’t be able to infer that a book about LionSubject is also about Animals

- Semantics for Lion and for the Lion subject are preserved

- The Animal and Subject hierarchies are independent of each other

- Maintenance is increased
  - Need to make sure all these classes and instances are consistent

- Use if:
  - Need to stay in OWL DL
  - Need to reason over the subject hierarchy
  - Not bothered by having parallel hierarchies
Approach 4
Using special restrictions

• Restrictions are used instead of specific values
Approach 4

http://www.w3.org/TR/swbp-classes-as-values/books4.owl
Approach 4
Sparql Query

PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX base: <http://protege.stanford.edu/swbp/books3.owl#>

SELECT ?book
WHERE { ?book rdf:type base:BookAboutLions }

<http://protege.stanford.edu/swbp/books4.owl#LionsLifeInThePrideBook>
<http://protege.stanford.edu/swbp/books4.owl#AfricanLionBook>

• Only the first book will be returned if no reasoner is used
Approach 4

• Compatible with OWL DL

• A reasoner can infer that a book with subject Lion also has the subject Animal
  – Can use a DL reasoner to classify specific books

• Subjects are assigned to books by creating instances of the relevant book subject class
  – No need to explicitly set any subject values
  – Can also use unspecified individuals of the class as property values, rather than the class itself
  – Interpretation: the subject is a prototypical lion, rather than the Lion class

• Use if:
  – Want to be in OWL DL
  – Want to use DL reasoners to classify your ontology
Approach 5: Using annotation properties

- Link individuals of Book with subjects using an annotation property
Approach 5

- Implementing this ontology in Protégé turns the ontology into OWL:FULL
  - Because the property becomes both owl:ObjectProperty and owl:AnnotationProperty

- Better to write/fix it by hand

- Download it from:
  http://users.ecs.soton.ac.uk/ha/teaching/COMP3028/approach5-books5.owl
Validating the Ontology

OWL Species Validation Report

Conclusion

Lite: YES

URL: None

RDF:

```xml
<?xml version="1.0"?>
<rdf:RDF
xmlns="http://www.ecs.soton.ac.uk/teaching/COMP3028/approach5-books5.owl"
xml:base="http://www.ecs.soton.ac.uk/teaching/COMP3028/approach5-books5.owl"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:owl.rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:owl="http://www.w3.org/2002/07/owl#">
<owl:Ontology rdf:about="/>
<owl:Class rdf:ID="AfricanLion">
<rdfs:subClassOf rdf:resource="#Lion"/>
</owl:Class>
```

Why?
SELECT ?book

    ?class rdfs:subClassOf base:Lion }

<http://www.ecs.soton.ac.uk/teaching/COMP3028/approach5-books5.owl#LionsLifeInThePrideBook>
<http://www.ecs.soton.ac.uk/teaching/COMP3028/approach5-books5.owl#AfricanLionBook>
**Approach 5**

- Compatible with OWL DL
  - Annotation properties can have classes as values in OWL DL

- Annotation properties cannot have different types
  - *dc:subject* cannot be an annotation property and an object or datatype property
  - This will render the ontology OWL FULL

- Restrictions cannot be applied to annotation properties

- DL reasoners don’t use annotation values
Topics

• N-ary relations

• Classes as property values

• Value partitions and Value sets
Value Partition

- Descriptive features are quite common in ontologies.

- Examples:
  - Size \{small, medium, large\}
  - Risk \{dangerous, risky, safe\}
  - Health status \{good health, medium health, poor health\}

- Also called “qualities”, “modifiers”, “attributes”

- A property can have only one value for each feature to ensure consistency.

- Such features can be represented as:
  - Enumerated individuals
  - Disjoint classes
  - Datatype values
Approach 1
Values as sets of individuals

- Class Health_Value is an enumeration of three individuals

What happens if we don’t add this axiom?
:has_health_status
  a        owl:ObjectProperty , owl:FunctionalProperty ;
  rdfs:range :Health_Value .

John
  a        :Person ;
  :has_health_status :good_health .

:good_health
  a        :Health_Value .

:Healthy_person
  a        owl:Class ;
  owl:equivalentClass
  owl:intersectionOf (:Person [ a        owl:Restriction ;
          owl:hasValue :good_health ;
          owl:onProperty :has_health_status
          ] ) .
Create an individual of the class Person, add good_health as value for has_health_status.

If you add an individual to Healthy_person directly, then property has_health_status will automatically be given the value good_health.
Approach 1: Values as sets of individuals

• Need an axiom to set the three health values to be different from each other
  – This way, a person cannot have more than one health value at a time

• Values cannot be further partitioned
  – Eg we cannot have moderately_good_health as a subtype of good_health
  – Only equality and difference between individuals is allowed in OWL

• Only one set of values is allows for a feature
  – The class cannot be equivalent to more than one set of distinct values
  – Doing so will cause inconsistencies

• OWL DL compatible
Approach 2: Values as subclasses

- Values are disjoint subclasses
The inference engine can now infer that John is a Healthy_person.
Approach 2: Values as subclasses

- The instance Johns_Health can be made anonymous
Approach 2: Values as subclasses

- OWL DL compatible
- DL reasoners can classify the ontology
- Values can be further partitioned
  - Simply add subclasses to the value classes
- Can have alternative partitioning of the same feature
OWL Wizards

- Protégé has OWL wizards for creating n-ary relations, value partitions and enumerations (values as individuals)
Meronymies (part-whole relations)

- Taxonomies are not the only hierarchical relation that we wish to model

- A spark plug isn’t a kind of engine (class-instance)

- A spark plug is a **part of** an engine
Simple Part-Whole Representation

- We need two properties:
  - partOf (a transitive property)
  - directPartOf (a subproperty of partOf)
Part-Whole Hierarchies

- Represent part-whole relationships between classes using someValuesFrom restrictions

SparkPlug $\subseteq \exists$ directPartOf.Engine

Engine $\subseteq \exists$ directPartOf.Car
Defining Classes of Parts

- Extend the ontology with classes of parts for each level
  - Reasoner can automatically derive a class hierarchy

\[
\text{CarPart} \equiv \exists \text{partOf}.\text{Car} \\
\text{DirectCarPart} \equiv \exists \text{directPartOf}.\text{Car} \\
\text{EnginePart} \equiv \exists \text{partOf}.\text{Engine}
\]
Fault location

• Allows reasoner to conclude that a fault in a part is a fault in a whole

• Need a new property for the location of a fault: hasLocus

• Need a new class for faults: Fault

\[\text{FaultInCar} \equiv \text{Fault} \sqcap \exists \text{hasLocus.CarPart} \]
\[\text{FaultInEngine} \equiv \text{Fault} \sqcap \exists \text{hasLocus.EnginePart} \]