

# Ontology Development

- Ontology Engineering
- Step-By-Step: Developing an ontology
- Examples of Developing an ontology

# Ontology Engineering

It is a methodology issues that is applied in building ontologies, manually, reusing ontologies, and using semi-automatic methods



Constructing Ontologies main stages:

1. Determine scope
2. Consider reuse
3. Enumerate terms
4. Define taxonomy
5. Define properties
6. Define facets

# How to build an ontology?

- **Steps:**
  - determine domain and scope
  - enumerate important terms
  - define classes and class hierarchies
  - define slots
  - define slot restrictions (cardinality, value-type)
    - » Slot-cardinality
      - *Ex: Borders\_with* multiple, *Start\_point* single
    - » Slot-value type
      - *Ex: Borders\_with*- Country

# Step 1: Determine Domain and Scope

**Domain:** geography

**Application:** route planning agent

**Possible questions:**

Distance between two cities?

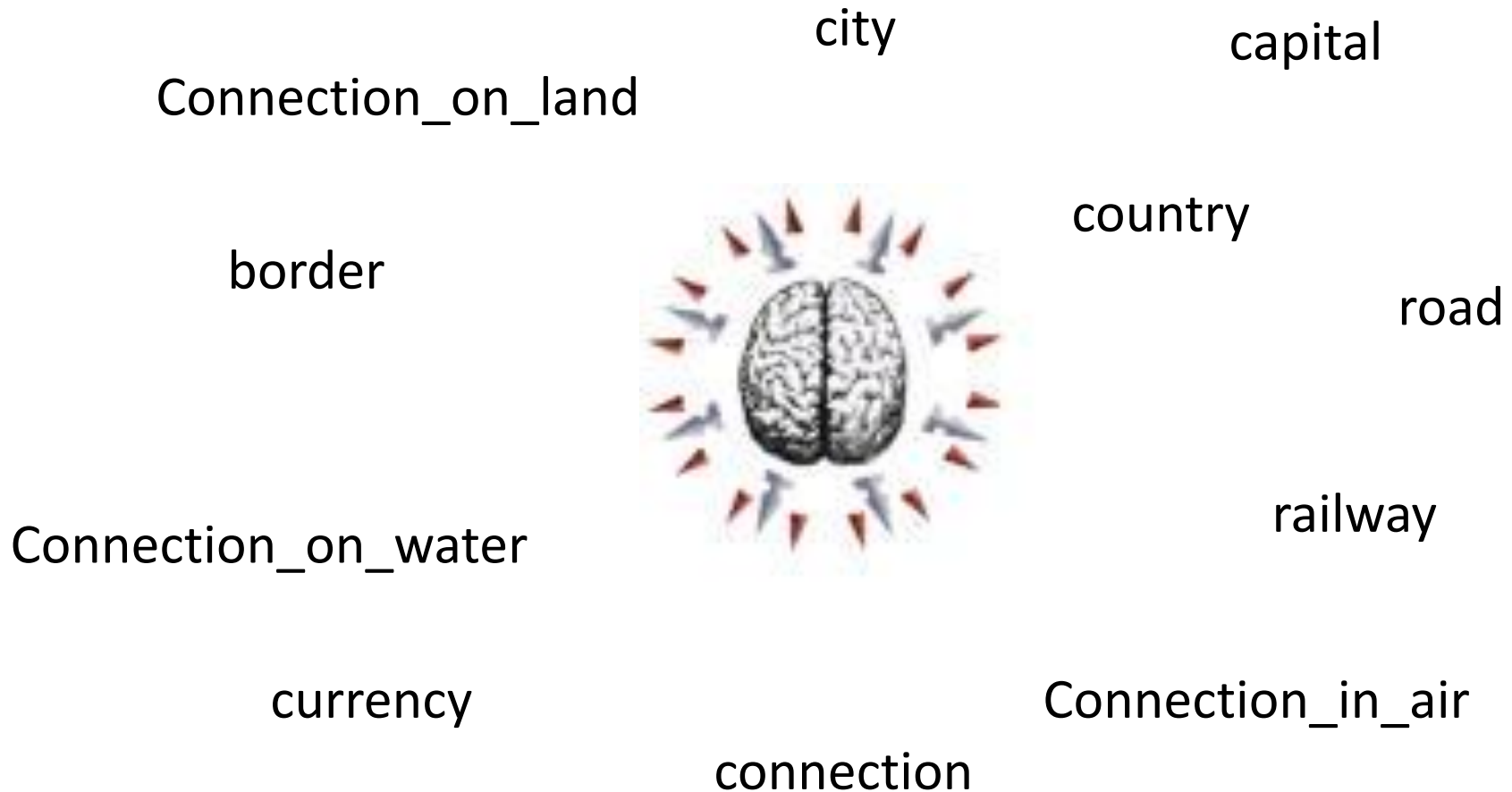
What sort of connections exist between two cities?

In which country is a city?

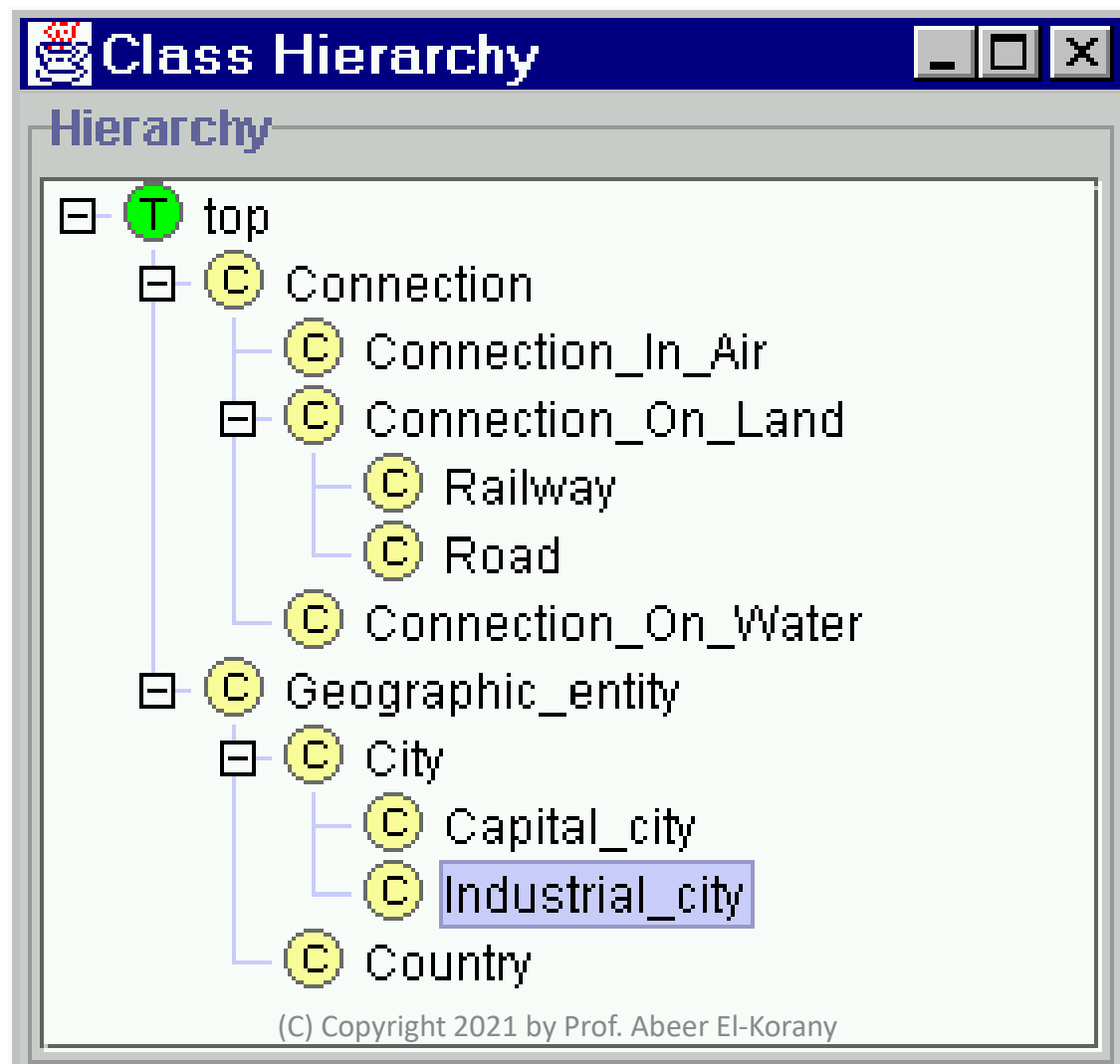
How many borders are crossed?



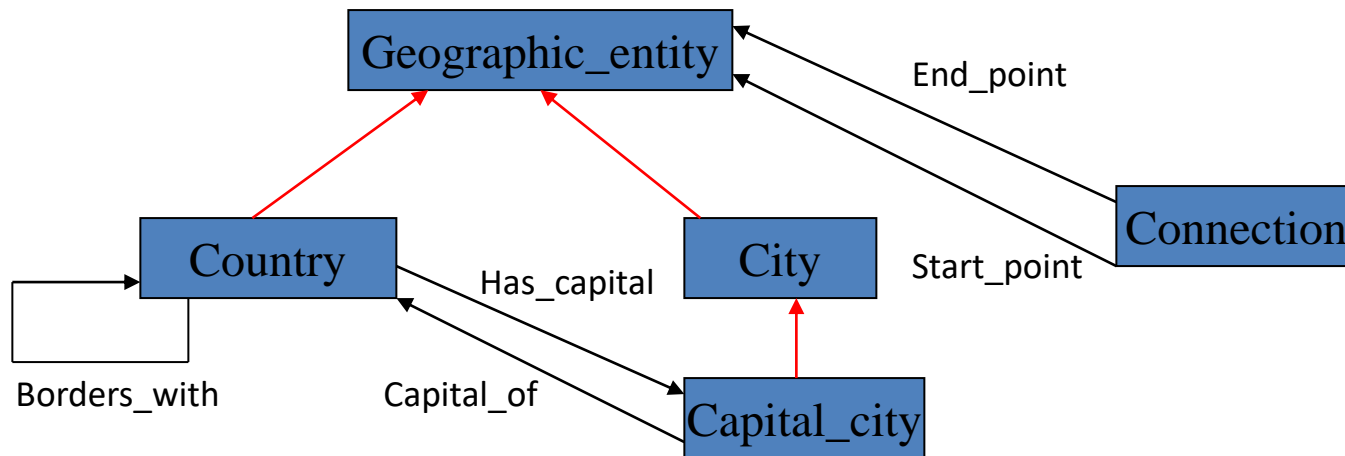
# Step 2: Enumerate Important Terms



# Step 3: Define Classes and Class Hierarchy



## Step 4: Define Slots of Classes



# Step 5: Define slot constraints

- Constraints are Background knowledge on the domain
  - **Adult\_Elephants** weigh at least 2,000 kg
  - All **Elephants** are either **African\_Elephants** or **Indian\_Elephants**
  - No individual can be both a **Herbivore** and a **Carnivore**



# Ontology Main Elements

- Concepts(classes) + their hierarchy
- Concept properties (slots/attributes) + their hierarchy
- Property restrictions (type, cardinality, domain ...)
- Relations between concepts (disjoint, equality ...)
- Instances

# Ontology Main Elements(cont.)

- Defining terms in the domain and relations among them
  - Defining concepts in the domain (classes).
  - Arranging the concepts in a hierarchy (subclass-superclass hierarchy).
  - Defining which attributes and properties (slots) classes can have and constraints on their values.
  - Defining individuals and filling in slot values.

# Ontology Components: **Classes**

- Classes are used to group things together.
- In most representations, members of classes must be *individuals*.
- In more expressive representations, *classes* may be also be allowed to be members of other classes.
- Classes can be *subsumed by*, or can *subsume* other classes  $\Rightarrow$  **subclasses** and **superclasses**.
- This leads to the class hierarchy, which is central to most ontologies.
- Some ontologies consist **only** of a class hierarchy – these are called **taxonomy**

# Ontology Components: Individuals

- Individuals are **instances** or **objects**
- These are usually **concrete**  
(e.g. uk\_prime\_minister, FCI\_student\_1389203)
- They can be **abstract** (e.g. numbers and words)
- Two individuals may be equivalent  
(e.g. uk\_prime\_minister, Boris Johnson)
- It is not always clear whether something ought to be an individual or a class  
(e.g.uk\_prime\_minister)

# Ontology Components: **Attributes**

- Attributes are aspects, properties, features, characteristics, or parameters that objects and classes can have.
- Attributes can link objects and classes to:
  - Specific values (integers, individuals or other literals)
  - Complex data types (e.g. enumerated lists)
  - Boolean values (true/false)
  - Other Classes

# Ontology Components: Relations

- Relations describe how classes/individual relate to one another.
- Typically, relations are defined between classes, and instantiations of relations are between individuals.
  - `course(Course_Name, instructor, Level, Credits, Year)`
  - `course(CS-SW, Abeer, 4, 3, 2021/2022)`
- More restricted representations may limit this, e.g. only allow binary relations.

# **EXAMPLES OF DEVELOPING AN ONTOLOGY**

# Example1: animals ontology

- Purpose & scope:
- To provide an ontology for an index of a children's book of **animals** including
  - Where they live
  - What they eat
    - Carnivores, herbivores and omnivores
  - How dangerous they are
  - How big they are
  - A bit of basic anatomy
    - numbers of legs, wings, toes, etc.



# Example1: Animals & Plants

## 1-Collect the concepts

- Dog
- Cat
- Cow
- Person
- Tree
- Grass
- Herbivore
- Male
- Female
- Carnivore
- Plant
- Animal
- Fur
- Child
- Parent
- Mother
- Father
- Dangerous
- Pet
- Domestic Animal
- Farm animal
- Food animal
- Fish
- Carp
- Goldfish

# Example: Animals & Plants

## Ontology Development

### 1-Organise the concepts

- 
- Dog
  - Cat
  - Cow
  - Person
  - Tree
  - Grass
  - Herbivore
  - Male
  - Female
  - Carnivore
  - Plant
  - Animal
  - Fur
  - Child
  - Parent
  - Mother
  - Father
  - Healthy
  - Pet
  - Domestic Animal
  - Farm animal
  - Draft animal
  - Food animal
  - Fish
  - Carp
  - Goldfish

# Ontology Development (cont.)

## 2-Organize the concepts

### “Laddering”

- **Add abstractions where needed**

e.g. “Living thing”

- Take a group of things and ask what they have in common
  - Then what other ‘siblings’ there might be
- e.g.
  - Plant, Animal → Living Thing
    - Might add Bacteria and Fungi but not now (scalability)
  - Cat, Dog, Cow, Person → Mammal
    - Others might be Goat, Sheep, Horse, Rabbit,...
  - Cow, Goat, Sheep, Horse → Hoofed animal
    - What others are there? Do they divide amongst themselves?
  - Wild, Domestic → Domestication

Vocabulary note:

“Sibling” = “brother or sister”

# Ontology Development (cont.)

## Define:Self\_standing\_entities

- Self-standing things vs. Modifiers
- Things that can exist on there own nouns
  - People, animals, houses, actions, processes, ...
    - Roughly nouns
- Modifiers-> Properties (value)
  - Things that modify (“inhere”) in other things
  - (e.g., wild/domestic, male/female, healthy/sick, dangerous/safe)
    - Roughly adjectives and adverbs

# Ontology Development (cont.)

## Identify definable things, and modifiers

### Arrange Concepts/Properties into Hierarchy

- Living Thing
  - Animal
    - Mammal
      - Cat
      - Dog
      - Cow
      - Person
    - Fish
      - Carp
      - Goldfish
  - Plant
    - Tree
    - Grass
    - Fruit
- Modifiers
  - domestic
    - pet
    - Farmed
      - Draft
      - Food
  - Wild
  - Health
    - healthy
    - sick
  - Gender
    - Male
    - Female
  - Age
    - Adult
    - Child

# Consider Reuse



- Why reuse other ontologies?
  - to save the **effort**
  - to **interact** with the tools that use other ontologies
  - to use ontologies that **have been validated** through use in applications

# Enumerate Important Terms



- What are the terms we need to talk about?
- What are the properties of these terms?
- What do we want to say about the terms?
- Examples in a Pizza Ontology:
  - Fish, seafood, meat, vegetables, cheese

# Ontology Development (cont.)

## Modifiers

- Identify modifiers that have mutually exclusive values (Domestication, Dangerousness, Gender, Age)
- Not mutually exclusive usage.
  - (can be both Draught and Food)
- Extend and complete lists of values
  - (Dangerousness: Dangerous, Risky, Safe)
- There are two ways of specifying values for modifiers
  - ❖ value partitions (classes that partition a quality)
  - ❖ value sets (individuals that enumerate all states of a quality)



# Ontology Development (cont.)

- **Identify relations**
  - e.g. “eats”, “owns”, “parent of”
- **Identify definable things**
  - e.g. “child”, “parent”, “Mother”, “Father”
    - **Things where you can say clearly what it means**
      - (Father, Herbivore, etc)
      - **Try to define a dog precisely – very difficult**
        - » A “natural kind”

# Ontology Development (cont.)

Reorganise everything but “definable” things into pure trees

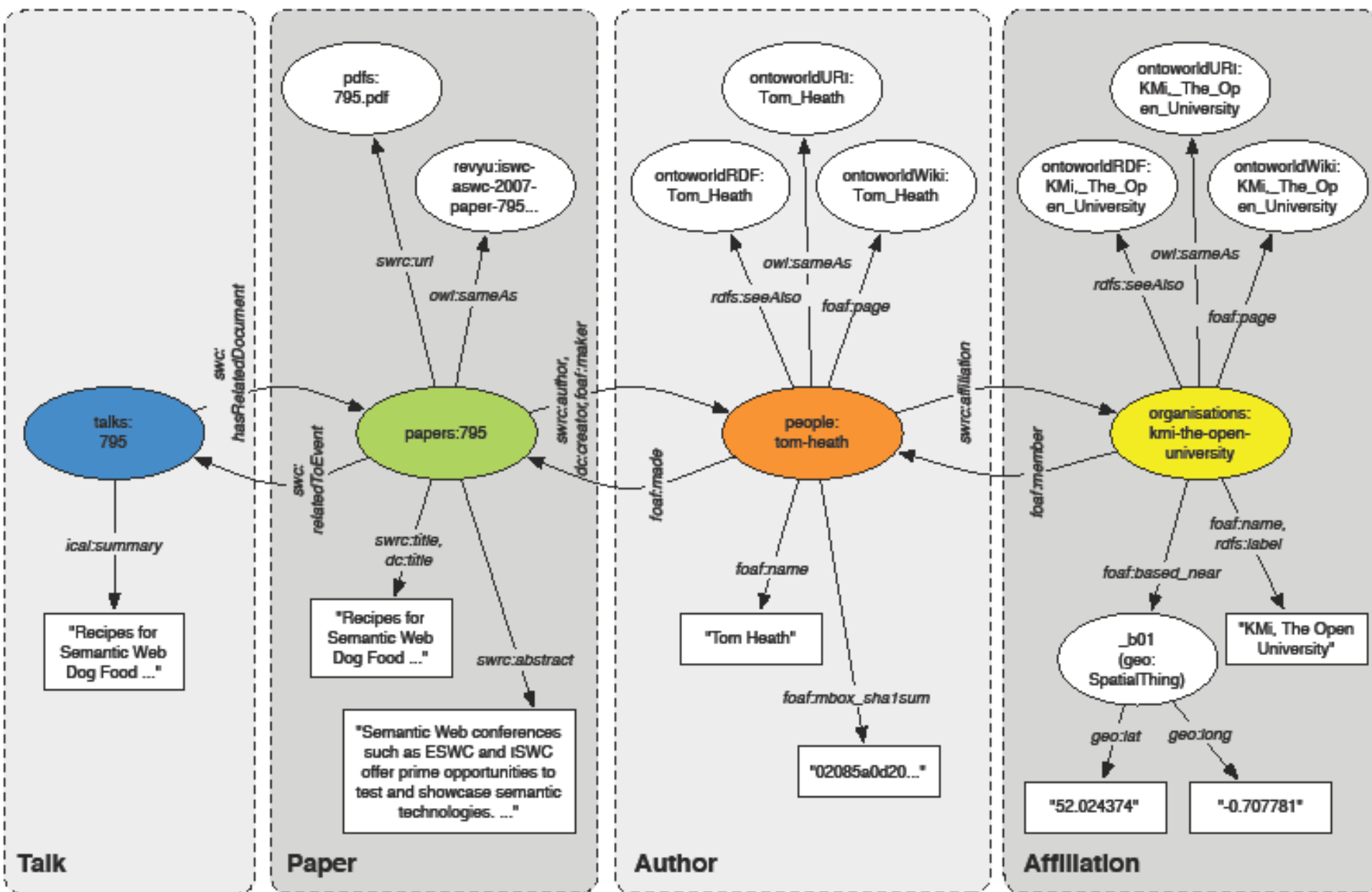
- Living Thing
  - Animal
    - Mammal
      - Cat
      - Dog
      - Cow
      - Person
    - Fish
      - Carp
      - Goldfish
  - Plant
    - Tree
    - Grass
    - Fruit
- Modifiers
  - domestic
    - pet
    - Farmed
      - Draft
      - Food
  - Wild
  - Health
    - healthy
    - sick
  - Gender
    - Male
    - Female
  - Age
    - Adult
    - Child
- Relations
  - eats
  - owns
  - parent-of
  - ...
- Definable
  - Carnivore
  - Herbivore
  - Child
  - Parent
  - Mother
  - Father
  - Food Animal

# Define Property

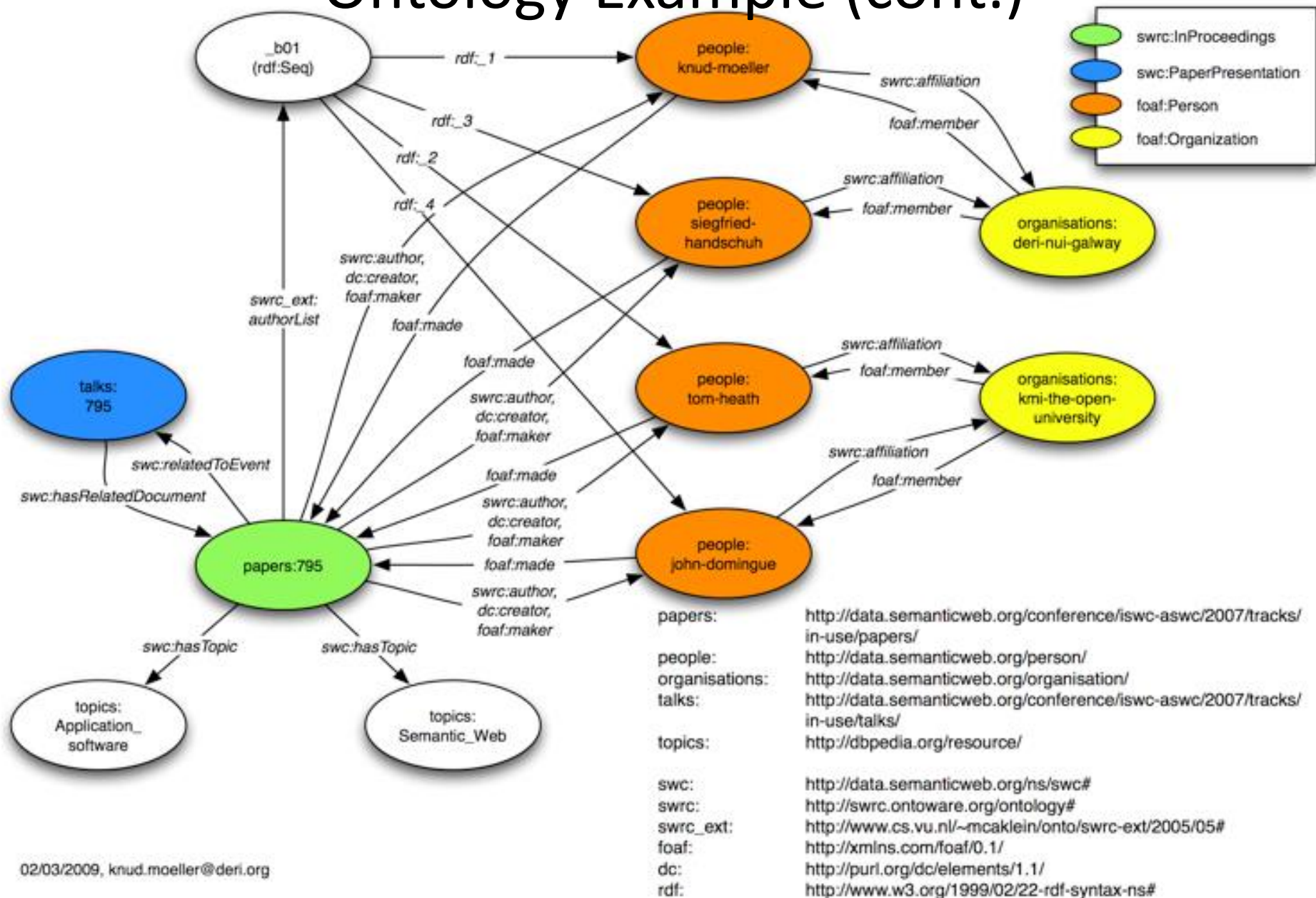
- Identify the **domain** and **range** constraints for properties
- Animal **eats** LivingThing:
  - domain: Animal range: LivingThing
- Person **owns** LivingThing except Person
  - domain: Person range: LivingThing and not Person
- Animal **parentOf** Animal
  - domain: Animal range: Animal

# Ontology Example (cont.)

**2- How can we create an ontology for the academic research domain (people, publications, etc)?**



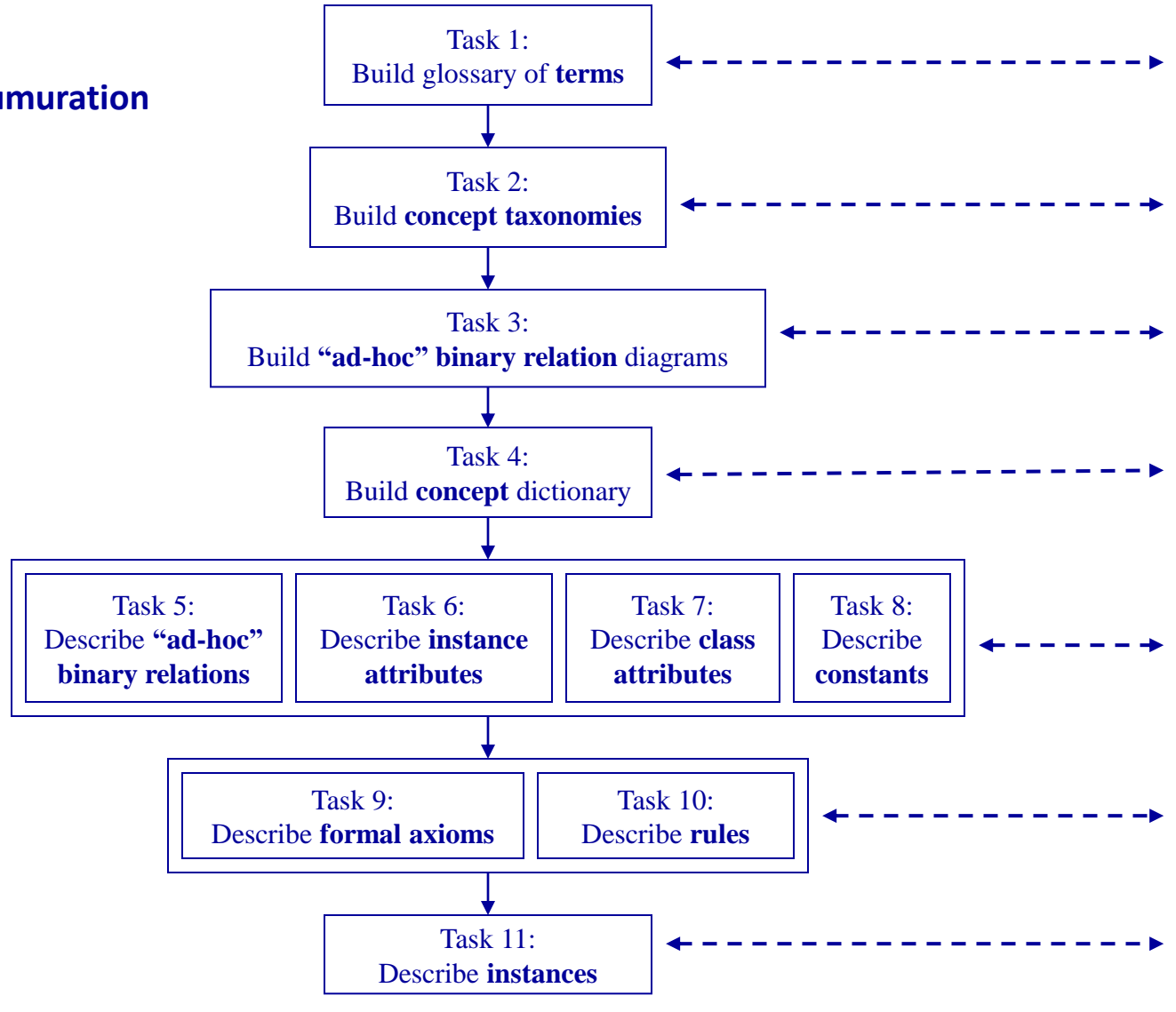
# Ontology Example (cont.)



02/03/2009, knud.moeller@deri.org

# Example3: Traffic airlines

## Tasks of the conceptualization/enumuration

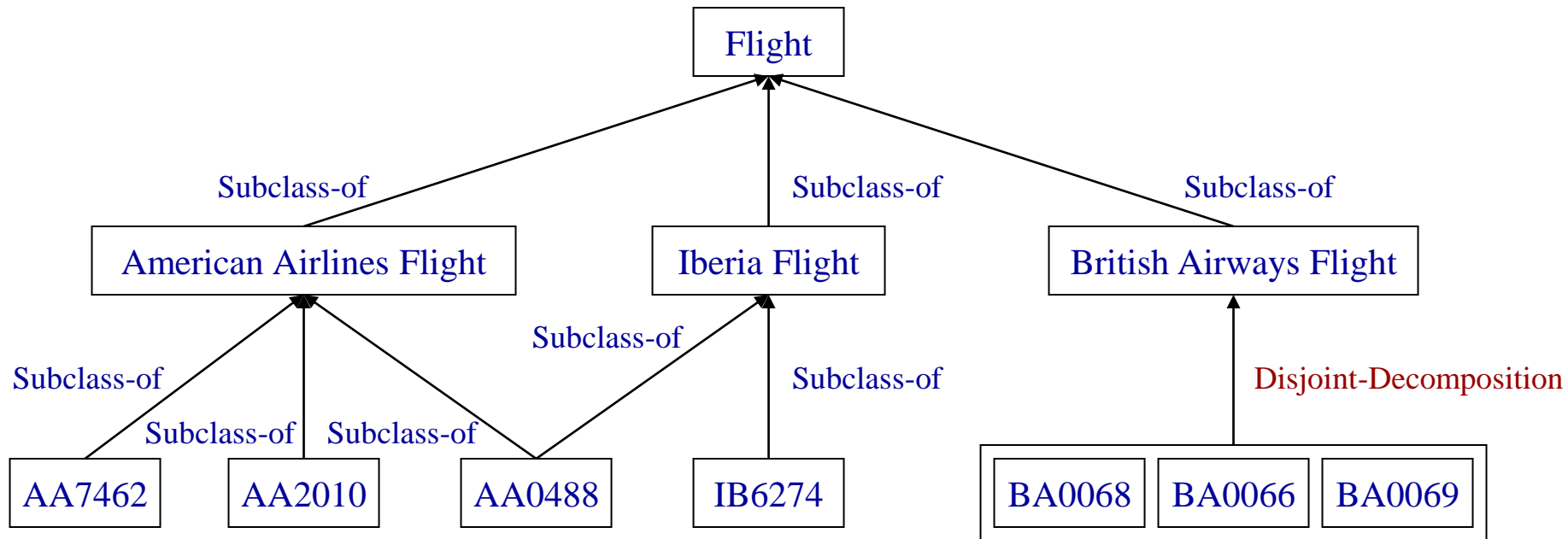


# Terms glossary

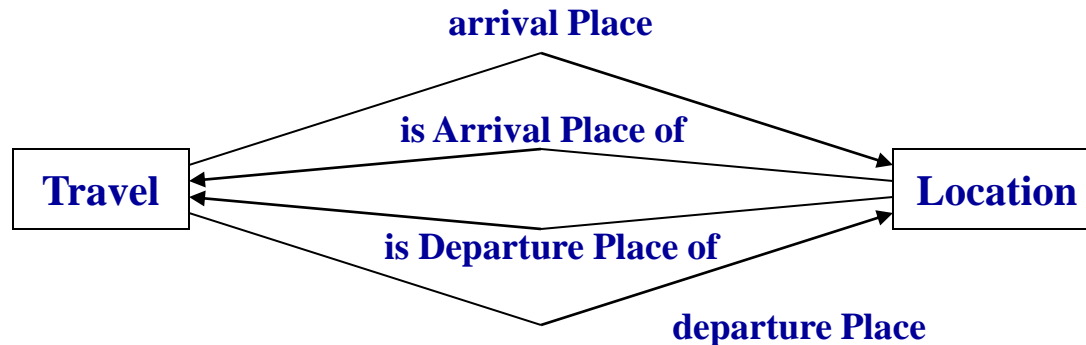
Name	Synonyms	Acronyms	Description	Type
American Airlines Flight	--	AA Flight	Flight operated by American Airlines.	Concept
Bed and Breakfast	--	--	An establishment (as an inn) offering lodging and breakfast	Concept
British Airways Flight	--	BA Flight	Flight operated by British Airways.	Concept
Business Trip	--	--	A special package for businessmen, consisting of a flight and a good quality hotel.	Concept
Camping	--	--	Temporal lodging in a camp.	Concept
Economy Trip	--	--	An economic package, usually costing less than 1000\$.	Concept
European Location	--	--	A location in Europe.	Concept
Five-stars Hotel	--	--	High quality hotel	Concept
Flight	--	--	A journey by plane identified by a flight number.	Concept
Hotel	--	--	An establishment that provides lodging and usually meals, entertainment, and various personal services for the public	Concept
Iberia Flight	--	IB Flight	Flight operated by Iberia.	Concept
Japan Location	--	--	A location in Japan.	Concept
Location	Place	--	A position or site occupied or available for occupancy or marked by some distinguishing feature.	Concept
Lodging	Accommodation	--	A temporary place to stay during a trip, sleeping accommodations.	Concept
Luxury Trip	--	--	A luxury and expensive trip.	Concept
Spain Location	--	--	A location in Spain.	Concept
Train Travel	Rail Travel	--	A journey by train.	Concept
Travel	--	--	A journey from place to place.	Concept
Travel Package	--	--	A travel package that a person can ask for. It consists of one or several means of transport and one or several accommodations.	Concept



# Example of a Taxonomy



# Identify Ad-hoc relations



Relation name	Source concept	Source card. (Max)	Target concept	Mathematical properties	Inverse relation
same Flight as	Flight	N	Flight	Symmetrical Transitive	--
placed in	Lodging	1	Location	--	--
accommodated in	Travel Package	N	Lodging	--	--
arrival Place	Travel	1	Location	--	is Arrival Place of
departure Place	Travel	1	Location	--	is Arrival Place of
arrival Place	Travel Package	1	Location	--	is Departure Place of
departure Place	Travel Package	1	Location	--	is Departure Place of

# Define a Concept Dictionary

Concept name	Class attributes	Instance attributes	Relations
AA 7462	--	--	same Flight as
American Airlines Flight	company Name	--	--
British Airways Flight	company Name	--	--
Five-stars Hotel	number of Stars	--	--
Flight	--	--	same Flight as
Location	--	name size	is ArrivalPlace of is Departure Place of
Lodging	--	price of Standard Room	placed in
Travel	--	arrival Date company Name departure Date return Fare single Fare	arrival Place departure Place
Travel Package	--	budget finalPrice name number of Days travel Restrictions	arrival Place departure Place accommodated in travels in
USA Location	--	--	--

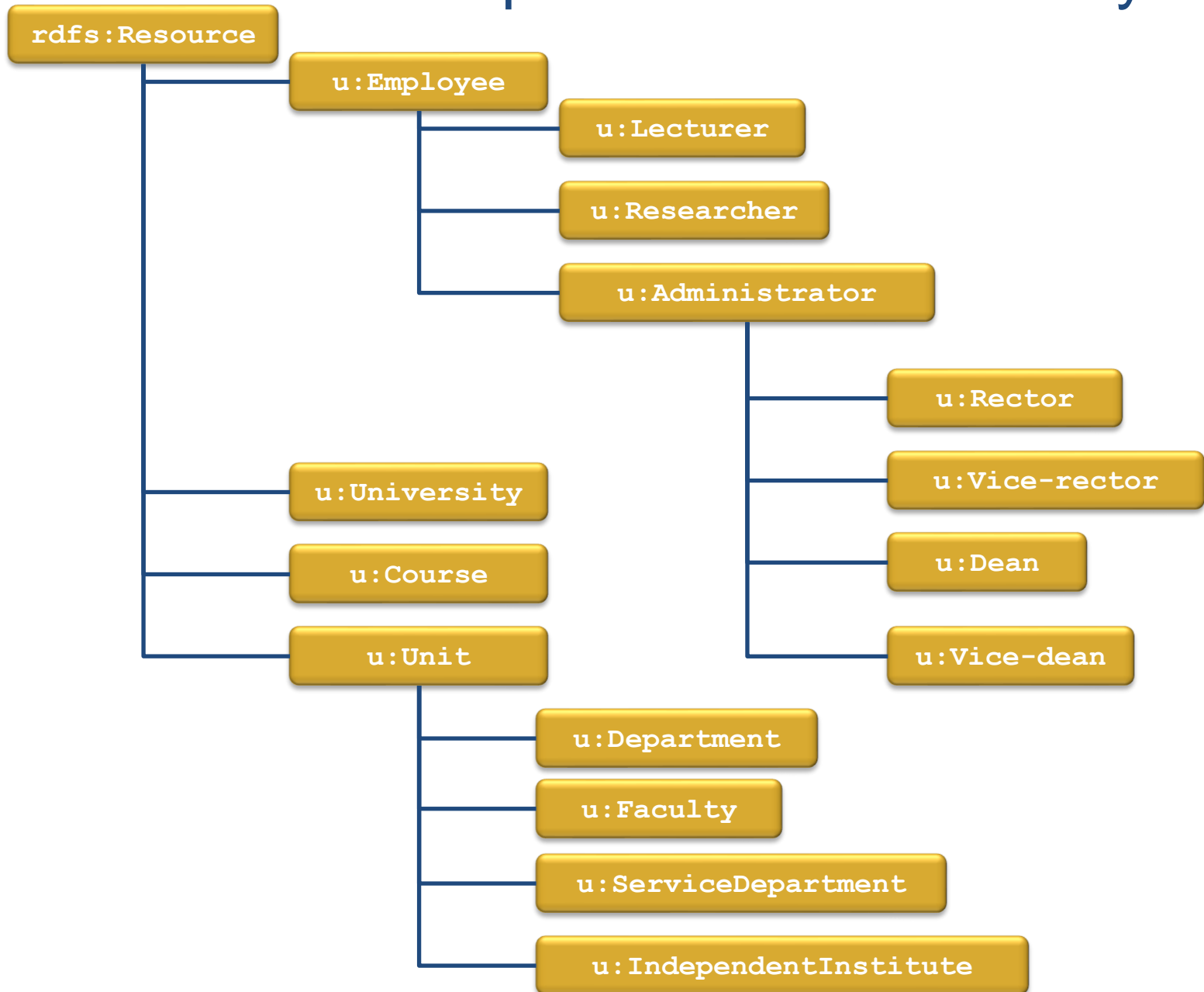
# Define Class Attributes

Attribute name	Defined at concept	Value type	Measurement unit	Precision	Cardinality	Values
company Name	American Airlines Flight	String	--	--	(1,1)	AA
company Name	British Airways Flight	String	--	--	(1,1)	BA
company Name	Iberia Flight	String	--	--	(1,1)	IE
number of Stars	Five-stars Hotel	Integer	star	1	(1,1)	5
number of Stars	Four-stars Hotel	Integer	star	1	(1,1)	4
number of Stars	Three-stars Hotel	Integer	star	1	(1,1)	3
number of Stars	Two-stars Hotel	Integer	star	1	(1,1)	2
number of Stars	One-stars Hotel	Integer	star	1	(1,1)	1

# Define in detail Instance Attributes

Instance attribute name	Concept name	Value type	Measurement unit	Precision	Range of values	Cardinality
budget	Business Trip	Float	Currency Quantity	001	1000...3000	(0,1)
budget	Economy Trip	Float	Currency Quantity	001	0...1000	(0,1)
name	Location	String	--	--	--	(1,N)
size	Location	Integer	Square Meters	1	--	(1,1)
price of Standard Room	Lodging	Float	--	--	--	(0,1)
budget	Luxury Trip	Float	Currency Quantity	001	--	(0,1)
arrival Date	Travel	Date	--	--	--	(0,1)
company Name	Travel	String	--	--	--	(0,N)
departure Date	Travel	Date	--	--	--	(0,1)
return Fare	Travel	Float	Currency Quantity	001	--	(0,1)
single Fare	Travel	Float	Currency Quantity	001	--	(0,1)
budget	Travel Package	Float	Currency Quantity	001	--	(0,1)
finalPrice	Travel Package	Float	Currency Quantity	001	--	(0,1)
number of Days	Travel Package	Integer	days	1	--	(0,1)
travel Restrictions	Travel Package	String	--	--	--	(0,1)

# Example of Class Hierarchy



# Class hierarchy: Modes of Development

- **top-down** – define the most general concepts first and then specialize them
- **bottom-up** – define the most specific concepts and then organize them in more general classes
- **combination** – define the more salient concepts first and then generalize and specialize them

# Defining Classes and a Class Hierarchy (cont.)

- Things to remember:
  - There is no single correct class hierarchy
  - But there are some guidelines
- The question to ask:
  - “Is each instance of the subclass an instance of its superclass?”

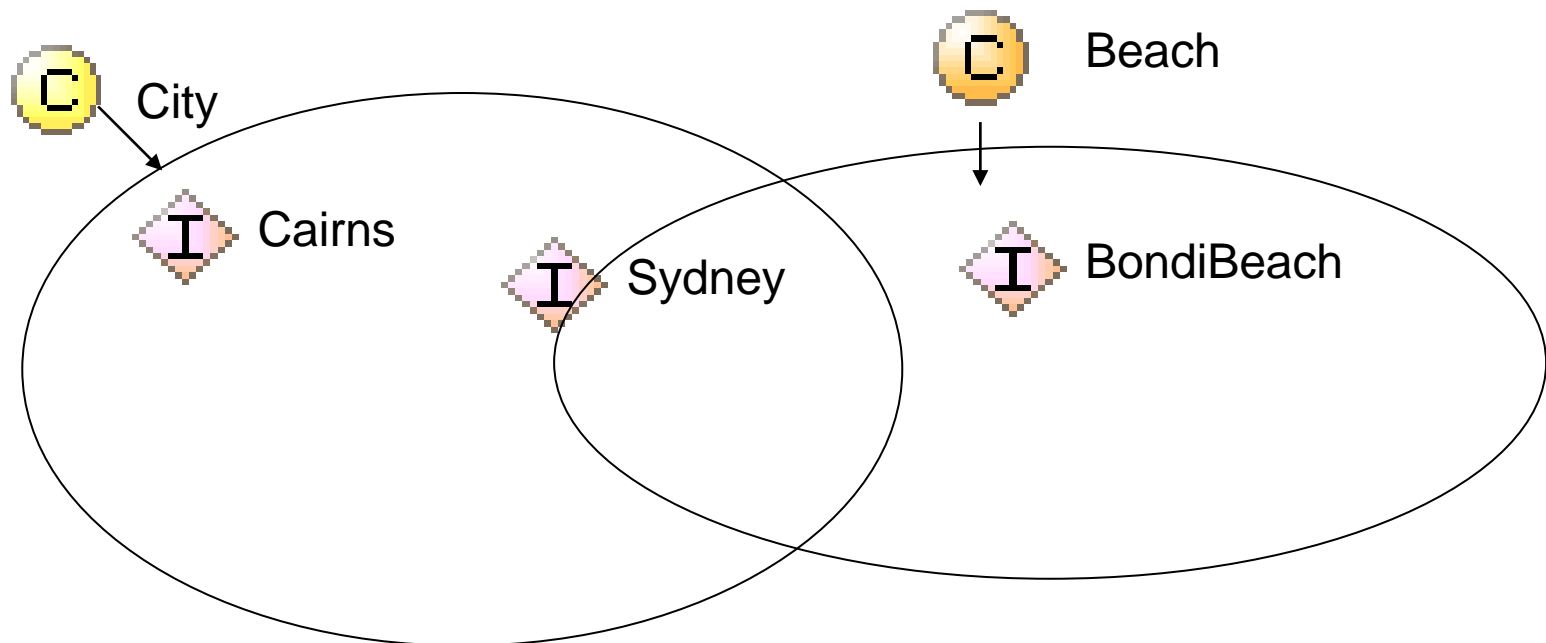


# Class Inheritance

- Classes usually constitute a **taxonomic hierarchy** (a subclass-superclass hierarchy)
- A class hierarchy is usually an IS-A hierarchy:
  - *An instance of a subclass is an instance of a superclass.*
- If you think of a class as a **set** of elements, a subclass is a **subset**
  - Apple is a subclass of Fruit
  - Every apple is a fruit*

# Class Relationships

- Classes can be organized in a hierarchy
- Direct instances of subclass are also (indirect) instances of superclasses
- Classes can overlap arbitrarily

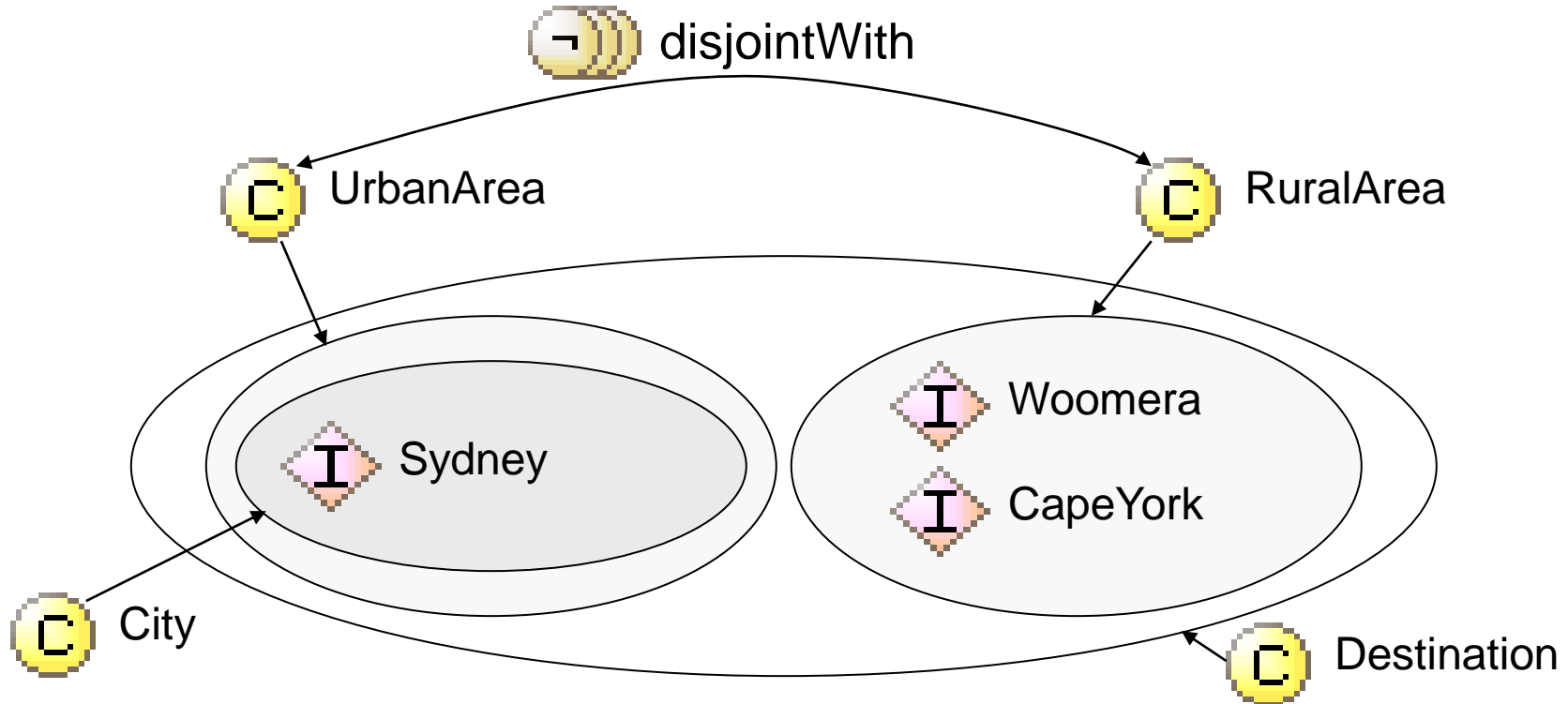


# Disjoint Classes

- They are classes that members of the selected class cannot also belong to.
- Classes are **disjoint** if they cannot have common instances.
- Example: There can be no animal that can be both an **Elephant** and a **Dog**.

# Class Disjointness

- All classes could potentially overlap
- In many cases we want to make sure they don't share instances

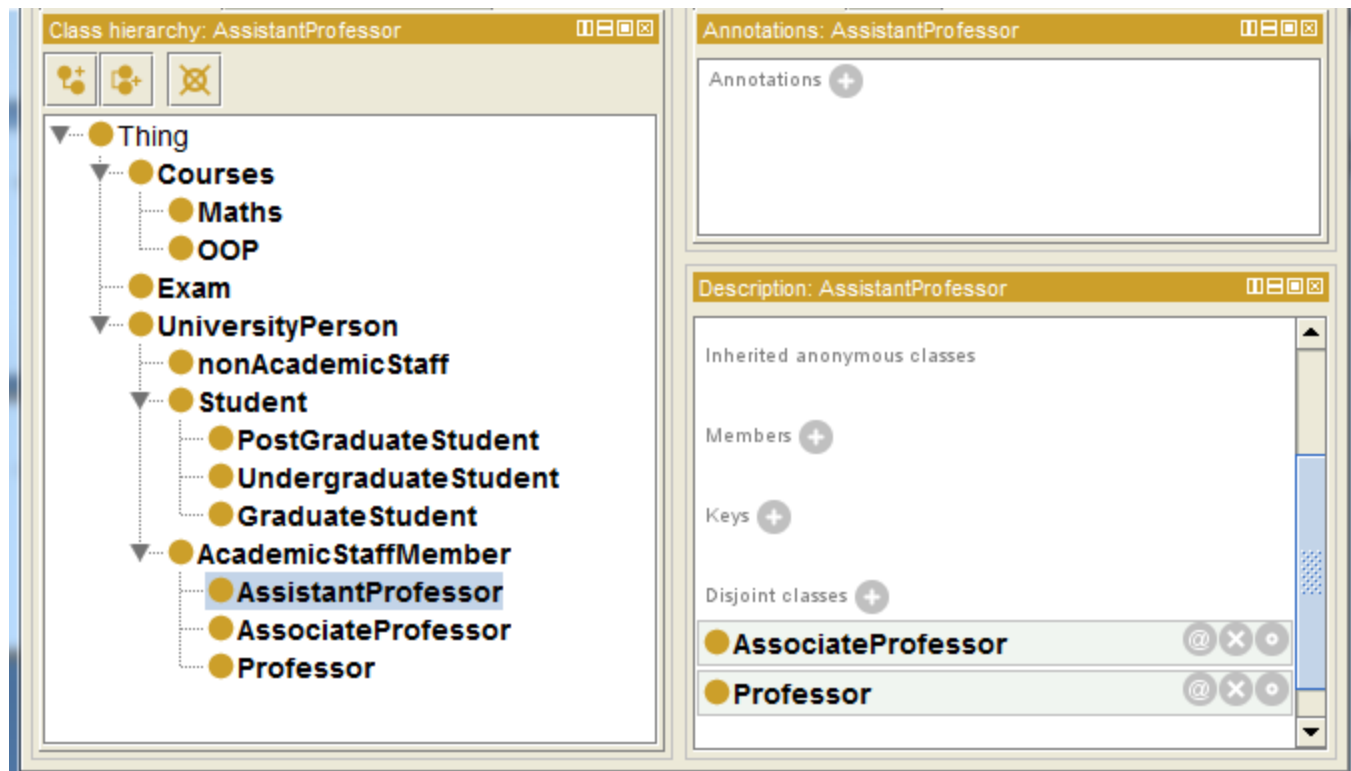


# Classes distinction

- To keep primitives in disjoint
  - need to distinguish the roles things play in different situations from what they are for example:
    - “pet”, “farm animal”, “draft animal”,
    - “professor”, “student”, ...
    - “doctor”, “nurse”, “patient”
- Often need to distinguish qualifications from roles
  - A person may be qualified as a doctor but playing the role of a patient

# DisjointWith

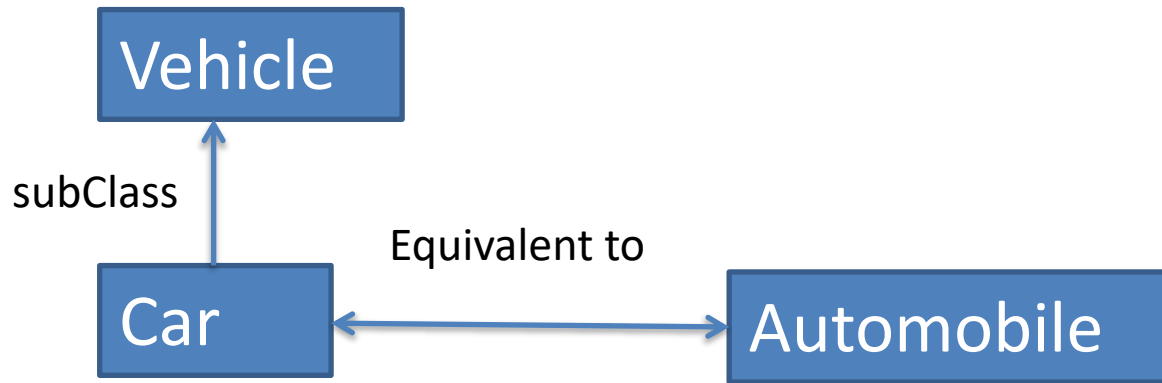
- Professor and Assistant professor are disjoint



# Equivalent classes

- Equivalent classes: other classes or groups that are equivalent to the selected class.
- In the case of **Parent**: it is “**any Person who has a child**”.
- Example:
  - **USPresident**
  - **PrincipalResidentOfWhiteHouse**
  - ```
<owl:Class rdf:about="#US_President">  
<owl:equivalentClass  
  rdf:resource="#PrincipalResidentOfWhiteHouse"/>  
</owl:Class>
```

# Equivalent classes: Example







# Golden Rules

- *There is no one correct way to model a domain*
  - There are always viable alternatives.
  - Best solution depends on
    - Application
    - Ability to cover extensions
- *Ontology development is an iterative process.*

# Golden Rules

- *Concepts in the ontology should be close to objects (physical or logical) and relationships*
- *Relations between Concepts (inheritance, disjoints, equivalents)*
- Very likely: in sentences that describe your domain:
  - Nouns are 
  - Verbs and prepositions are 

# Golden Rules

- *Concepts in the ontology should be close to objects (physical or logical) and relationships*
- *Relations between Concepts (inheritance, disjoints, equivalents)*
- Very likely: in sentences that describe your domain:
  - **Nouns** are **objects (concepts)**
  - **Verbs** and **prepositions** are **relationships**

# Golden Rules

- *Concepts in the ontology should be close to objects (physical or logical) and relationships*
- *Relations between Concepts (inheritance, disjoints, equivalents)*
- Very likely: in sentences that describe your domain:
  - **Nouns** are **objects (concepts)**
  - **Verbs** and **prepositions** are **relationships**

# Define Properties of Classes (Slots)



- Slots in a class definition describe Part of the meaning of a class (and thus participate in classification).
- Derived property to be inferred once class membership is known.

*E.g.: Each plant will have name, stem, root, leaves, etc.*

# Property and Class Inheritance

- A subclass **inherits all the slots** from the superclass.
  - *If a student has a **name** and an **id**, an undergraduate student also has a **name** and an **id**.*
- If a class has **multiple** superclasses, it inherits slots from **all of them** (Yes, there is multiple inheritance).

# Specify Values for each:

## Two methods

- Value partitions
  - Classes that partition a Quality
    - The disjunction of the partition classes equals the quality class
- Symbolic values (value set)
  - Individuals that enumerate all states of a Quality
    - The enumeration of the values equals the quality class

# Note any hierarchies of values

- **Modifiers**
  - **Domestication**
    - Domestic
    - Wild
    - Feral
  - **Risk**
    - Dangerous
    - Risky
    - Safe
  - **Gender**
    - Male
    - Female
  - **Age**
    - Child
      - Infant
      - Toddler
    - Adult
      - Elderly

- Identify modifiers that have mutually exclusive values
  - Domestication
  - Risk
  - Gender
  - Age
- Make meaning precise
  - Age → Age\_group
- NB Uses are not mutually exclusive
  - Can be both a draft and a food animal



# Method 1: Value Partitions- example “Dangerousness”

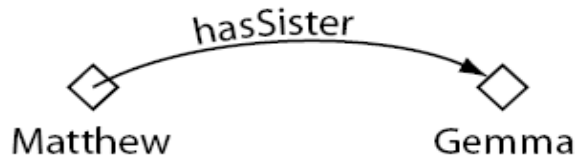
- Identify properties that have mutually exclusive values
- Dangerousness
- Subqualities for each degree
  - Dangerous, Risky, Safe
- all subqualities are disjoint – subqualities ‘cover’ parent quality,
- i.e., Dangerousness  $\equiv$  Dangerous  $\cup$  Risky  $\cup$  Safe
  - Dangerousness = Dangerous OR Risky OR Safe
- Define a property **has\_dangerousness**
  - Range is parent quality, e.g. Dangerousness
  - Domain must be specified separately
- **Dangerous\_animal** =  
Animal *and* **has\_dangerousness** some Dangerous
- DangerousAnimal  $\equiv$  Animal  $\cap \exists$ hasDangerousness.Dangerous

# Method 2: Value sets- example “Gender”

- A parent quality – `gender_value`
- Individuals for each value
  - `male`, `female`
- Values all different (NOT assumed by OWL)
- Value type is enumeration of values
  - `gender_value = {male, female}`
- Define a property `has_gender`
  - Range is parent quality, e.g. `gender_value`
  - Domain must be specified separately
- `Male_animal =`  
*Animal and `has_gender` is `male`*
- $\text{MaleAnimal} \equiv \text{Animal} \cap \exists \text{hasGender.male}$

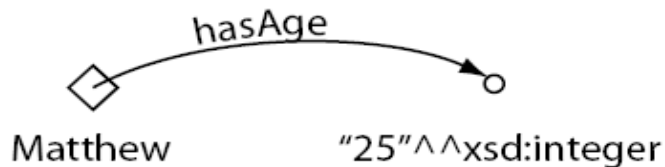
# Properties (Slots)

- Types of properties
  - Fundamental properties (**data properties**) :
    - attributes of instances of the class
      - **name** and **address** of student
      - **name** and **price** of dish
    - parts: **engine** in a car; **ingredients** in a dish
  - Relations to other object (**object properties**):
    - **publisher** of book or **producer** of an ingredient
- Simple and complex properties
  - **simple** properties (**data properties**): contain primitive values (strings, numbers).
  - **complex** properties (**object properties**): contain (or point to) other objects (e.g., **producer** of an ingredient, a publisher of a book).



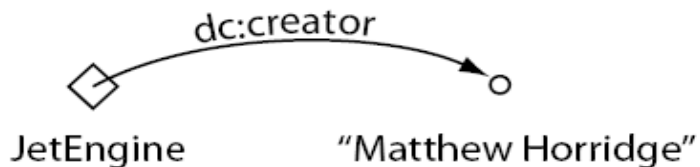
Object Property

An object property linking the individual Matthew to the individual Gemma



Data Property

A datatype property linking the individual Matthew to the data literal '25', which has a type of an xml:integer.

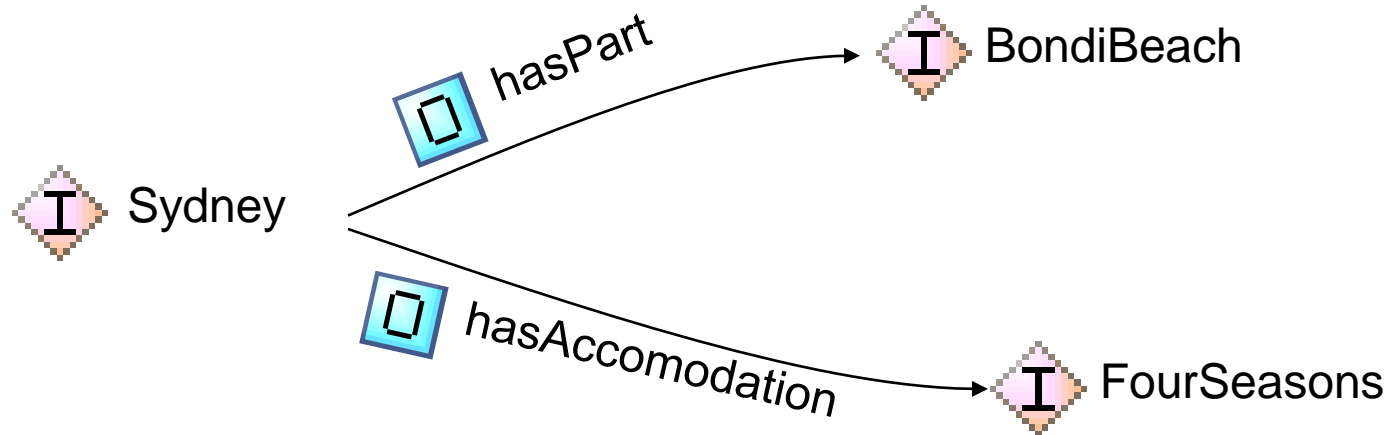


Data Property

An annotation property, linking the class 'JetEngine' to the data literal (string) "Matthew Horridge".

# ObjectProperties

- Link two individuals together
- Relationships (**0..n**, **n..m**)



# Domain and Range of Property

- **Domain** of a slot – the class (or classes) that have the slot
  - More precisely: class (or classes) instances of which can have the slot
- **Range** of a slot – the class (or classes) to which slot values belong
  - Example: TV show is ‘produced by’ TV
    - **Domain: TV show**
    - **Range: TV**

# Domain and Ranges of Property (cont.)



- When defining a domain or range for a slot, find **the most general class** or classes
- Consider the **publisher** slot for a **book**:
  - Domain: Book
  - Range: Publisher Instance [Amazon, Elsevier, Springer,...]

## Properties

- In ontologies we define property's *domain* and *range*
  - *Domain*: What can have this property
  - *Range*: What can be the value of this property

med:hasDiagnosis

D: Human

R: Diagnosis

hum:isAttractedBy

D: Human

R: Human

phy:isAttractedBy

D: Particle

R: Particle

hum:hasSurname

D: Human

R: rdfs:Literal

psy:hasAtomicNumber

D: Atom

R: xsd:string

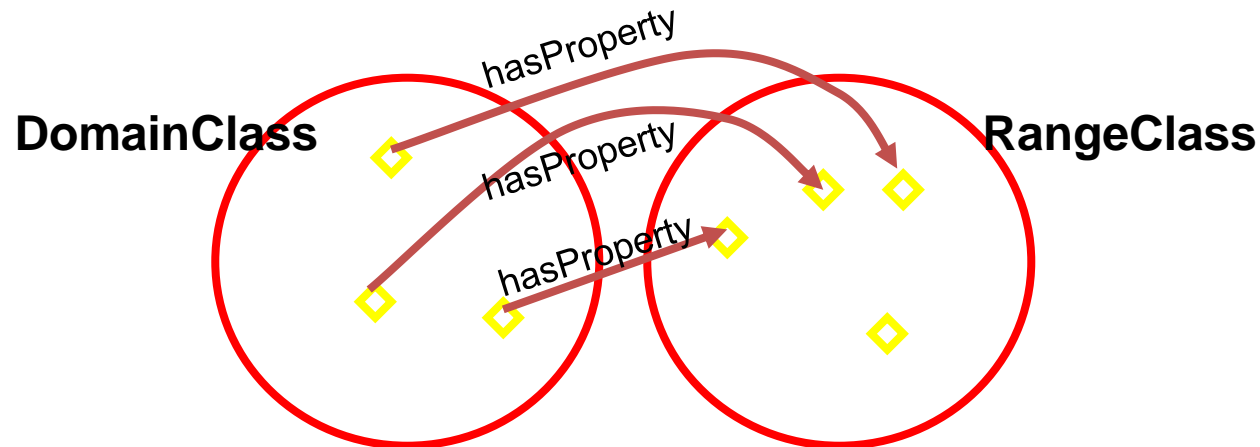


# Domain and Ranges of Property(cont.)

- *When defining a domain or a range for a slot, find the most general classes or class that can be respectively the domain or the range for the slots .*
- *All the classes in the domain of a slot should be described by the slot*
- *Instances of all the classes in the range of a slot should be potential fillers for the slot.*
- *Do not choose an overly general class for range (i.e., one would not want to make the range **THING**)*

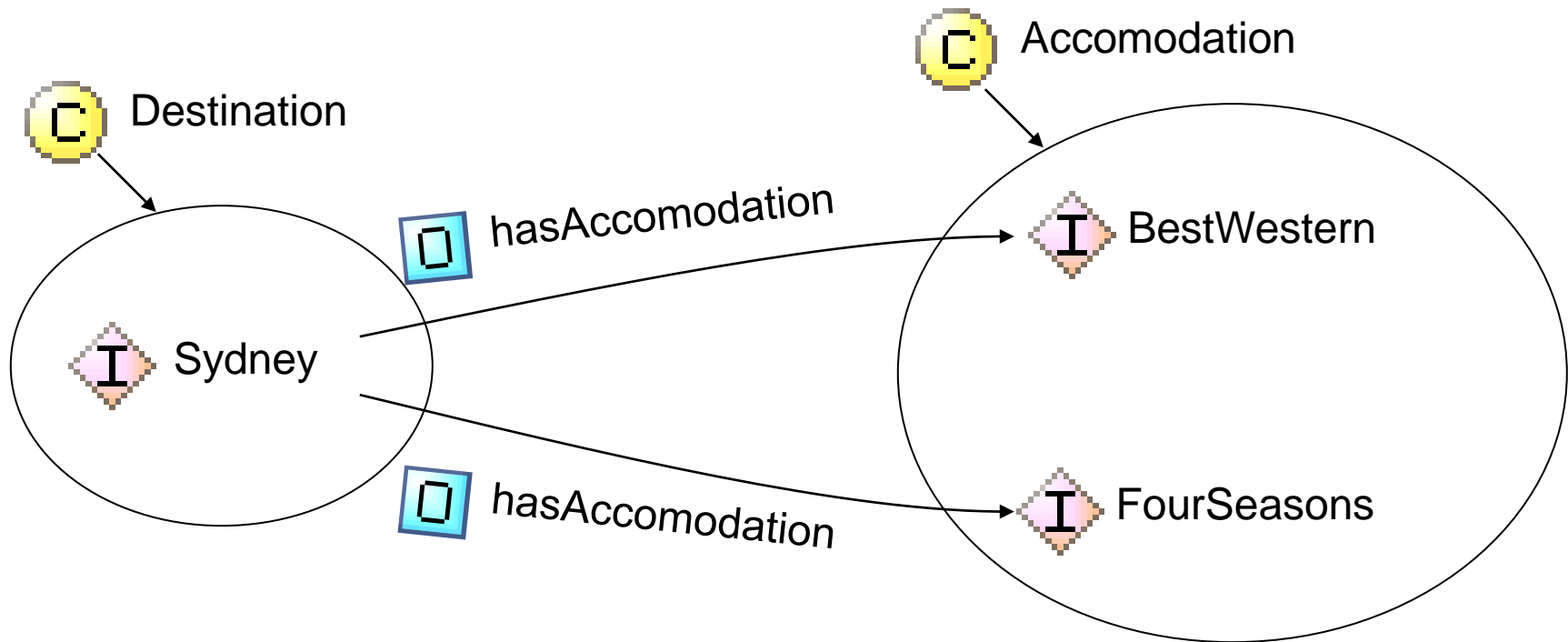
# Property Domain & Range

- If a relation is:  
subject\_individual  $\rightarrow$  hasProperty  $\rightarrow$  object\_individual
- The **domain** is the class of the **subject** individual
- The **range** is the class of the **object** individual (or a datatype if hasProperty is a Datatype Property)

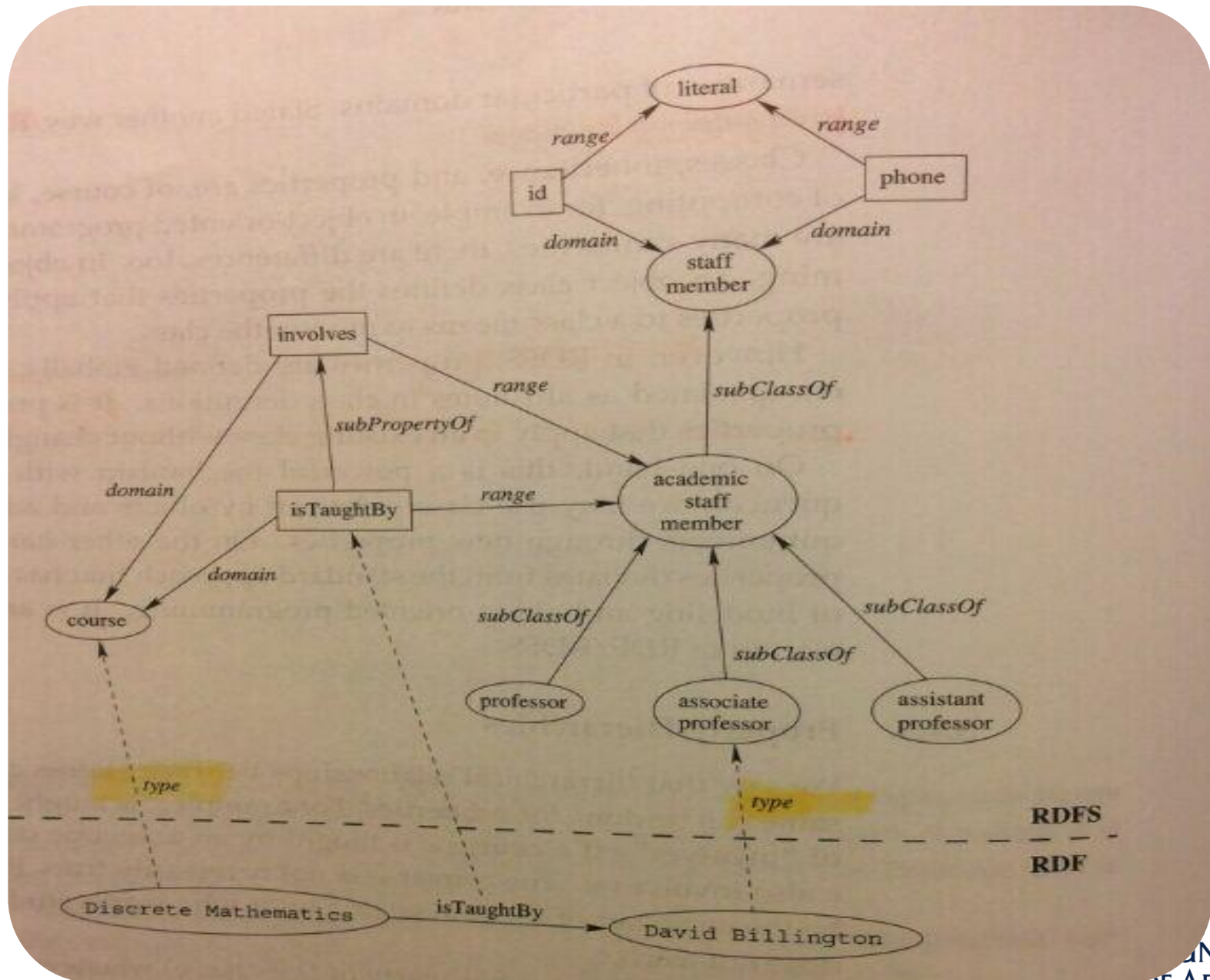


# Properties, Range and Domain

- Property characteristics
  - Domain: “left side of relation” (Destination)
  - Range: “right side” (Accommodation)



# Example



# Golden Rules

- A subclass inherits all the slots from the superclass
- A subclass can override the restrictions to “narrow” the list of allowed values
- Make the cardinality range smaller
- Replace a class in the range with a subclass

# Property Restriction

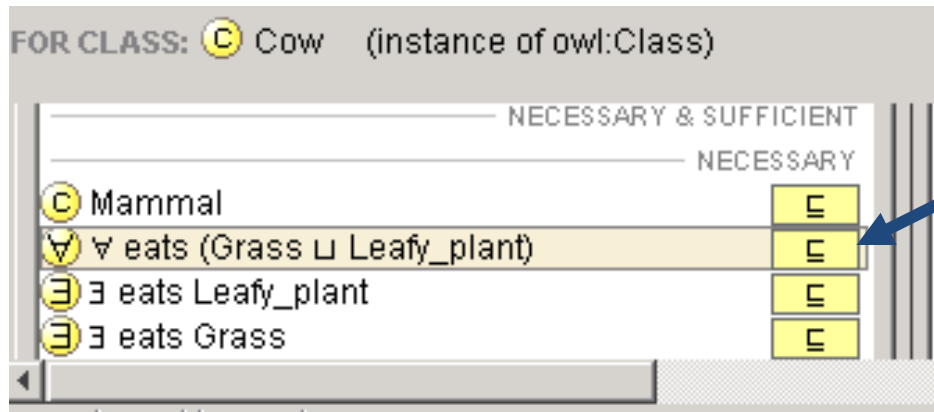
- Whenever required to describe the leave of a class, it is required to *close* down descriptions of entities
- Person owns LivingThing **except** Person
  - domain: Person
  - range: LivingThing and not Person
- “A ‘Herbivore’ is an animal that **only eats** plants”  
(NB All animals eat some living thing)
  - Herbivore=
    - eats domain: Herbivore  
range: Living\_thing & not Plants
    - Animal and eats only Plant
- “An ‘omnivore’ is an animal that eats both plants and animals”
  - Omnivore=
    - Animal and eats some Animal and eats some Plant

# Which properties can be filled in at the class level

- What can we say about all members of a class:
- For example: Eats

Cows are animals and, amongst other things,  
eat some plants and eat only plants

- all Cats eat some Animals
- all Omnivores eat some Animals and eat some Plants



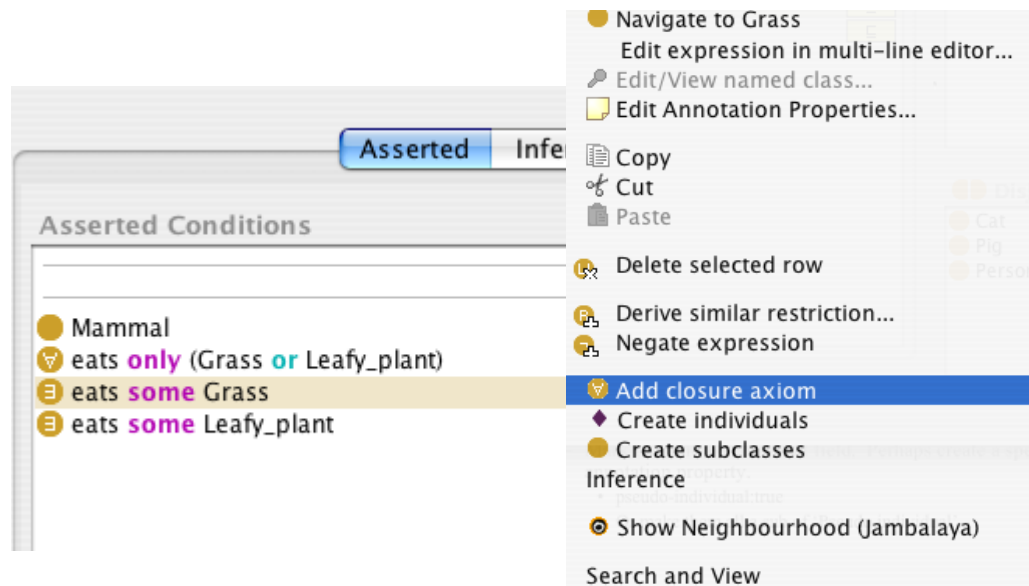
Closure  
Axiom

# In the tool

- Right mouse button short cut for closure axioms

- for any existential restriction

adds closure  
axiom





# Property Constraints

- Property constraints (**facets**) describe or limit the set of possible values for a slot
  - *The name of a student is a string*
  - *The book publisher is an instance of class publisher*
  - *A lecture has exactly one location*
  - *A lecture has exactly one time slot*

# Define the Values of the data property

- **Slot cardinality**
  - defines how many values a slot can have.
  - **Minimum and maximum** value – a range of values for a numeric slot.
  - **Default** value – the value a slot has unless explicitly specified otherwise.
- **Slot-value type**
  - what types of values can fill in the slot.
  - common value types:
    - **String**
    - **Number**
    - **Boolean**
    - **Enumerated**

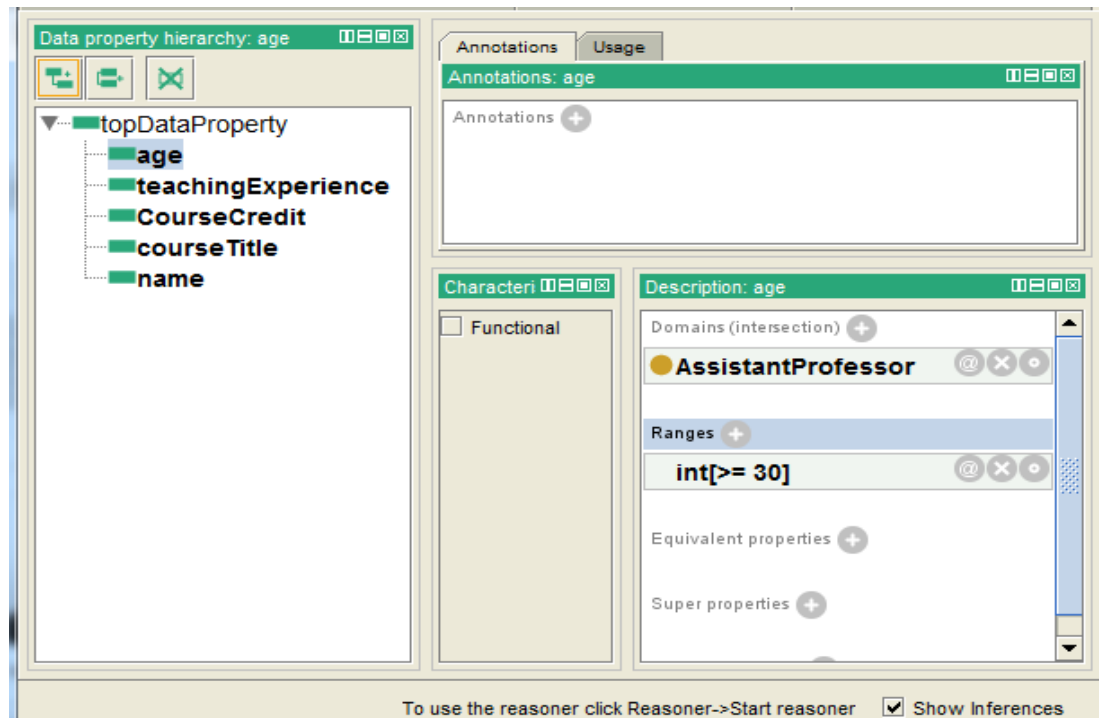
# 1-Common Facets: Value Type

- **String**: a string of characters (“Château Lafite”)
- **Number**: an integer or a float (15, 4.5)
- **Boolean**: a true/false flag
- **Enumerated type**: a list of allowed values (high, medium, low).
- **Complex type**: an instance of another class
  - Specify the class to which the instances belong

*The **Authors** is the value type for the slot “**Authored by**” at the **Book**.*

# Data Type property

- Data type properties will have ranges assigned to literal types
  - name, courseTitle, ...



# 2-Common Facets: Slot Cardinality

- Cardinality
  - Cardinality N means that the slot must have N values
- Minimum cardinality
  - Minimum cardinality 1 means that the slot must have a value **(required)**
  - Minimum cardinality 0 means that the slot value is **(optional)**
- Maximum cardinality
  - Maximum cardinality 1 means that the slot can have at most one value **(single-valued slot)**
  - Maximum cardinality greater than 1 means that the slot can have more than one value **(multiple-valued slot)**

# 3-Default Values

- Default value – a value the slot gets when an instance is created.
- A default value can be changed.
- The default value is a **common** value for the slot, but is not **a required value**.
- For example, the default value for quadratic shape can be Parallelogram.

# Slot cardinality

- Slot **cardinality** – the number of values a slot has
- Slot **value type** – the type of values a slot has
- **Minimum and maximum** value – a range of values for a numeric slot
- **Default** value – the initial value for a slot when the instance is created

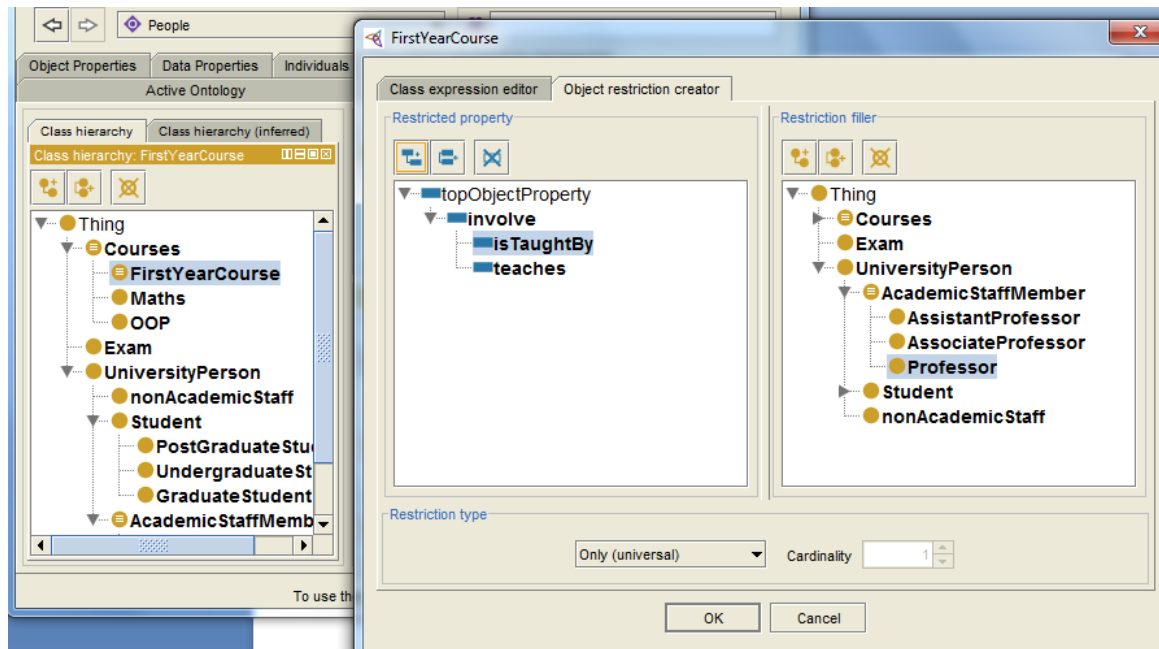
The screenshot shows a configuration window for a slot named 'wines\_00095' of type 'STANDARD-SLOT'. The window contains several fields and controls:

- Name:** A text field containing 'wines\_00095'.
- Documentation:** A large text area for additional information.
- Value Type:** A dropdown menu currently set to 'String'.
- Cardinality:** Two checkboxes, 'required' and 'multiple', both of which are unchecked. To the right of these are labels 'at least' and 'at most' followed by a small numeric input field containing '1'.
- Default:** A text area for specifying a default value.
- Minimum and Maximum:** Two numeric input fields for defining a range.
- Inverse Slot:** A checkbox that is currently unchecked.

At the top right of the window, there are window control buttons (minimize, maximize, close) and a small 'C' button. At the bottom right, there are additional controls labeled 'V', 'C', '+', and '-'.

# Property restriction

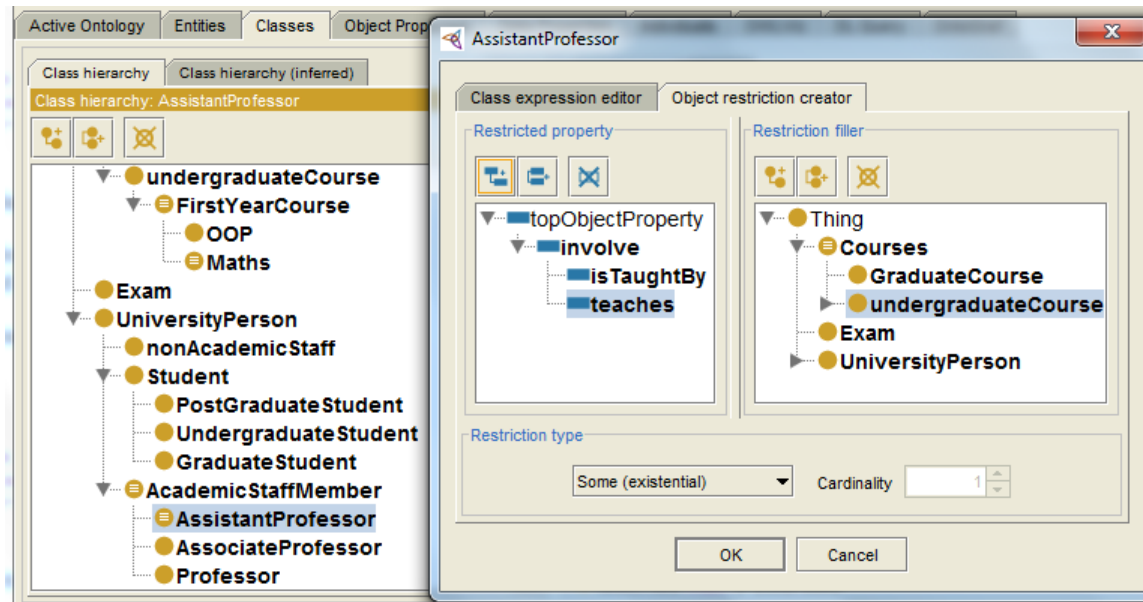
- allValuesFrom
- This constraint is analogous to the universal (for-all) quantifier of Predicate logic





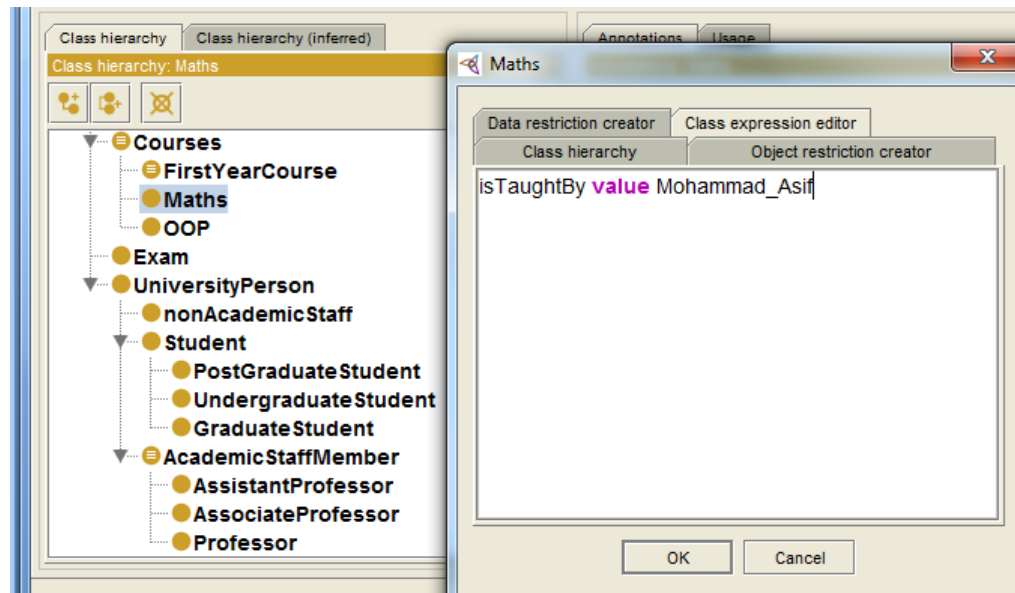
# Property restriction

- SomeValuesFrom
- constraint is analogous to the existential quantifier of Predicate logic



# Property Restriction

- hasValue

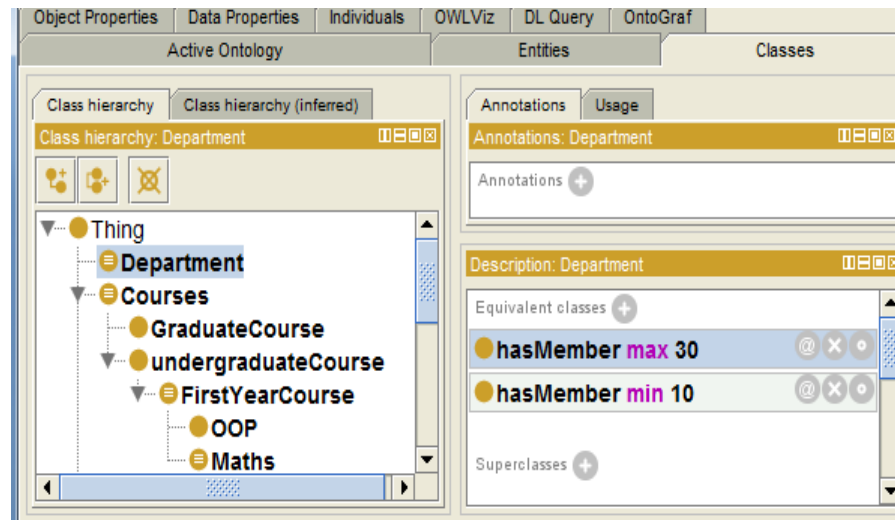


It is used to refer to an [individual](#) or a [data value](#).

A restriction containing a **hasValue** constraint describes a class of all individuals for which the property concerned has at least one value *semantically* equal to V

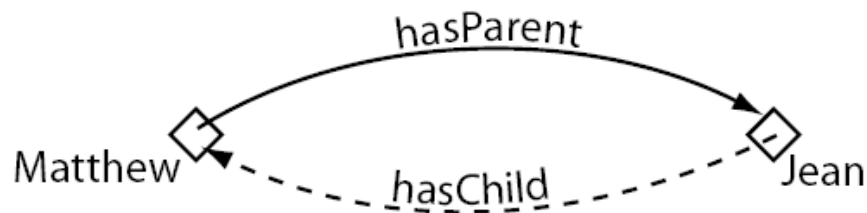
# Cardinality Restriction

- Department has min 10 members
- Department has max 30 members



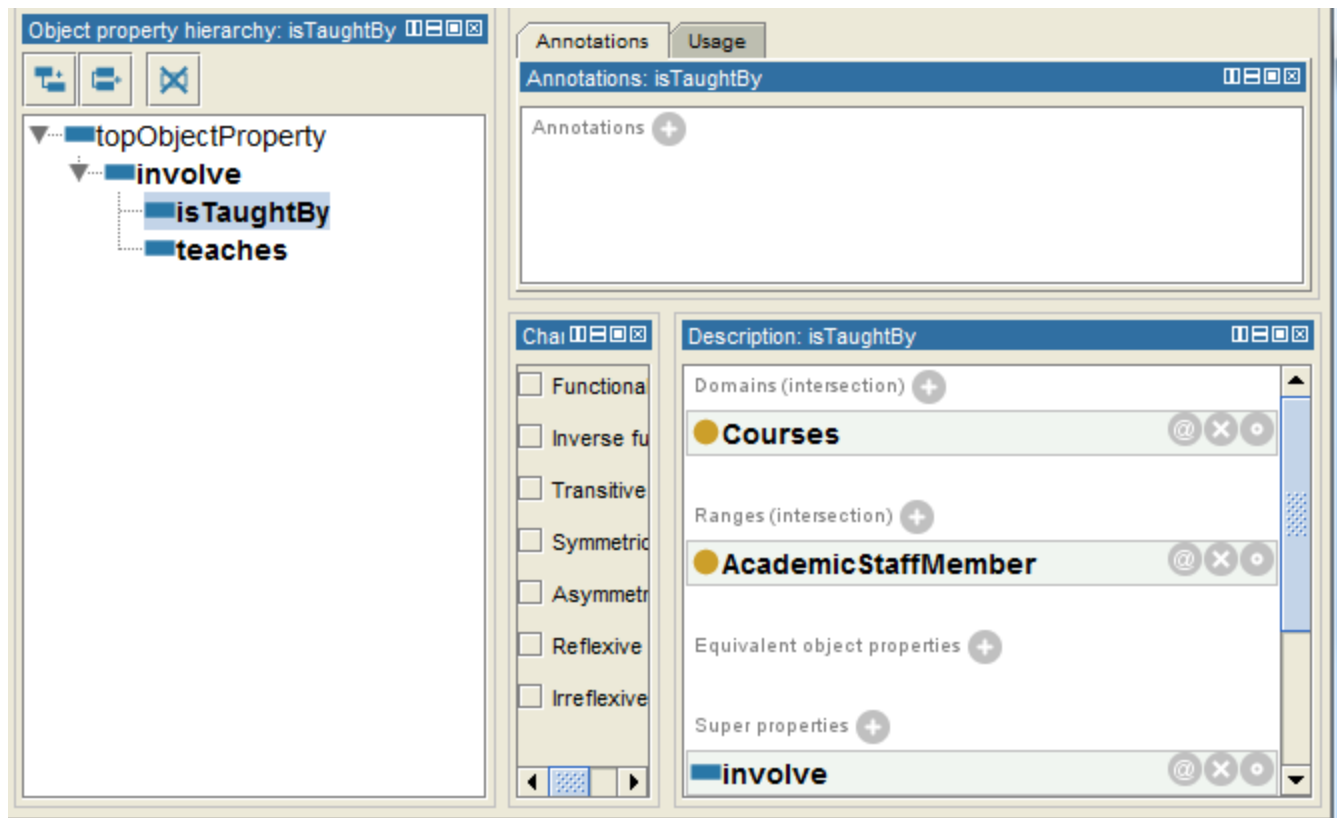
# Inverse Slot

- Each object property may have a corresponding inverse property.
- If some property links individual a to individual b, then its inverse property will link individual b to individual a.



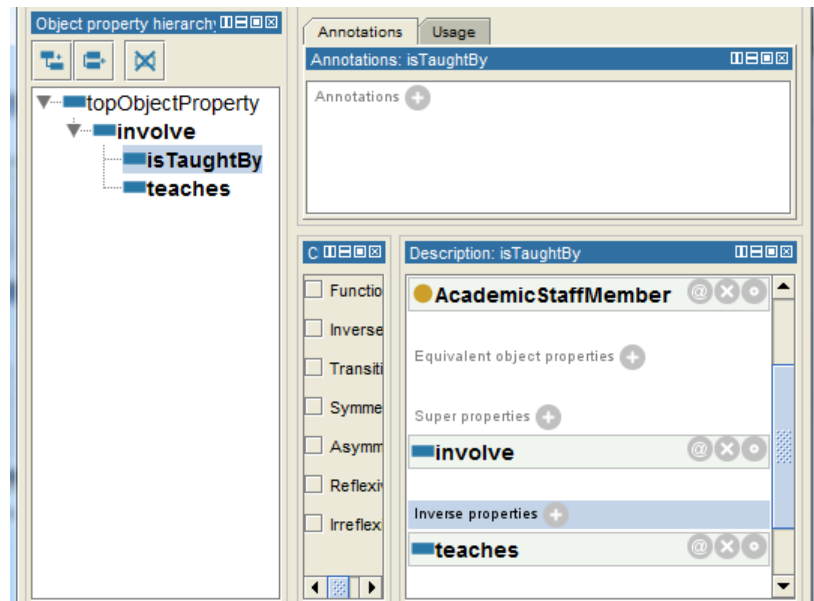
# Object Properties

- Object properties will have domain/ranges assigned to classes
  - Involve
    - isTaughtBy
    - teaches



# Inverse object properties

- teaches is inverse of isTaughtBy



# Ontology Engineering

## Steps to develop An ontology

### 1. determine domain and scope

- what is the domain that the ontology will cover?
- what we are going to use the ontology for?
- what types of questions the information in the ontology should provide answers for (competence questions)?

### 2 informal/semiformal knowledge acquisition

- Collect the terms
- Organize them informally
- Clarify terms to produce informal concept definitions
- Diagram informally

### 3 refine requirements and tests

# Steps to develop An ontology(cont.)

## 4. implementation

- implement prototype recording the intension as a summary
- scale up a bit and check performance

## 5 evaluation and quality assurance

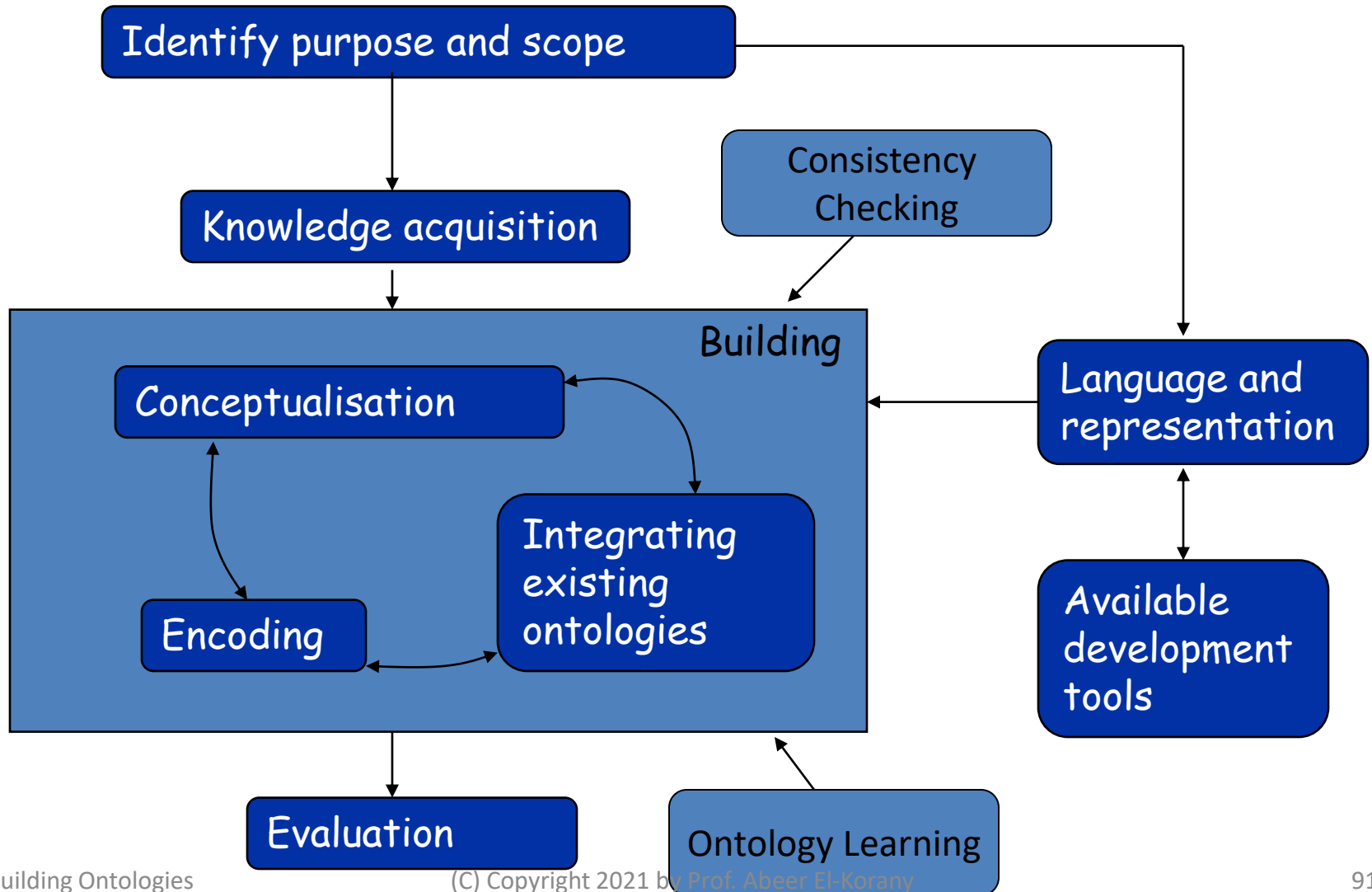
- against goals (ontology design is subjective!)
- include tests for evolution and change management
- design regression tests and ‘probes’

## 6 maintenance: usage monitoring and evolution

- – compatibility between different versions of the same ontology and between versions of an ontology and instance data



# An Ontology Development Life-cycle



# Ontology Engineering versus Object-Oriented Modeling

## An ontology

- reflects the structure of the **world**
- is often about **structure** of concepts
- actual physical representation is **not** an issue

## An OO class structure

- reflects the structure of **the data and code**
- is usually about **behavior** (methods)
- describes **the physical representation of data** (long int, char, etc.)

# Open vs Closed World reasoning

- **Open world reasoning**
  - Negation as contradiction
    - Anything might be true unless it can be proven false
      - Reasoning about *any world consistent with this one*
- **Closed world reasoning**
  - Negation as failure
    - Anything that cannot be found is false
      - Reasoning about *this world*
- ***Ontologies are not databases***

# Database -v- Ontology

## Database:

- Closed world assumption (**CWA**)
  - Missing information treated as false.
- Unique name assumption (**UNA**)
  - Each individual has a single, unique name.
- Schema behaves as **constraints** on structure of data
  - Define legal database states

## Ontology:

- Open world assumption (**OWA**)
  - Missing information treated as unknown
- **No UNA**
  - Individuals may have more than one name
- Ontology axioms behave like **implications** (inference rules)
  - Entail implicit information

# Exercise

Convert the following Statements into ontology component

|                                      |   |                                                                                                                  |
|--------------------------------------|---|------------------------------------------------------------------------------------------------------------------|
| George is an employee.               | → | An object is an instance of the Employee class.                                                                  |
| George works for Sony.               | → | An object in the Employee class is linked with an object in the Company class via the works_for relationship.    |
| George reports to Adam.              | → | An object in the Employee class is linked with another object in the same class via the reports_to relationship. |
| Fred works for a company.            |   |                                                                                                                  |
| Fred reports to two other employees. |   |                                                                                                                  |

# homework

- Use the Proteg´e editor to define a normalised ontology for use by a travel
- agency covering the following:
- Hotel, restaurant, sports, luxury hotel, bed and breakfast, safari, activity, hiking, spa treatment, sunbathing, sightseeing, accommodation rating (three stars, etc.), campground, surfing.

Build a class hierarchy and indicate which classes in it are primitive and which are definable.

Define the required relations, their properties, domains and ranges as well as individuals.

- Define the following classes:
- 1. A two star hotel.
- 2. A spa resort (i.e., a destination offering a spa treatment).
- 3. A destination with sport activities but without safari.
- 4. A destination where all hotels have three star rating.
- 5. A destinations with at least three restaurants and at least four hotels.