University of
Southampton

University of Southampton

Ontology Engineering

COMP6256 Knowledge Graphs for AI Systems

Dr Nicholas Gibbins - nmg@ecs.soton.ac.uk

Ontology Types

Representation ontologies

- Describe low level primitive representations
  e.g. RDFS, OWL

General or upper-level ontologies

- Describe high-level, abstract, concepts; Usually domain independent
  e.g. Cyc, DOLCE, WordNet, SUMO

Domain ontologies

- Describe a particular domain extensively
  e.g. Gene Ontology, CIDOC CRM

Application ontologies

- Designed to answer to the needs of a particular application
  e.g. FOAF, ESWC06

Ontology Building Methodologies

No standard methodology for ontology construction

There are a number of methodologies and best practices

The following life cycle stages are usually shared by the methodologies:

- Specification - scope and purpose
- Conceptualisation - determining the concepts and relations
- Formalisation - axioms, restrictions
- Implementation - using some ontology editing tool
- Evaluation - measure how well you did
- Documentation - document what you did

Specification

Specifying the ontology’s purpose and scope

- Why are you building this ontology?
- What will this ontology be used for?
- What is the domain of interest?
  - An ontology for car sales probably doesn't need to know much about types and prices of engine oil
- How much detail do you need?

Specification: Competency Questions

What are the questions you need the ontology to answer?

- These are competency questions
- Make a list of such questions and use as a check list when designing the ontology
- Helps to define scope, level of detail, evaluation, etc.

Specification: Competency Questions

The questions that you REALLY need

- You probably don’t need to worry about the questions that “perhaps someone might need to ask someday”

The questions that CAN BE answered

- Can you get the necessary data to answer those questions?
- Permanent lack of some data may render parts of the ontology useless!

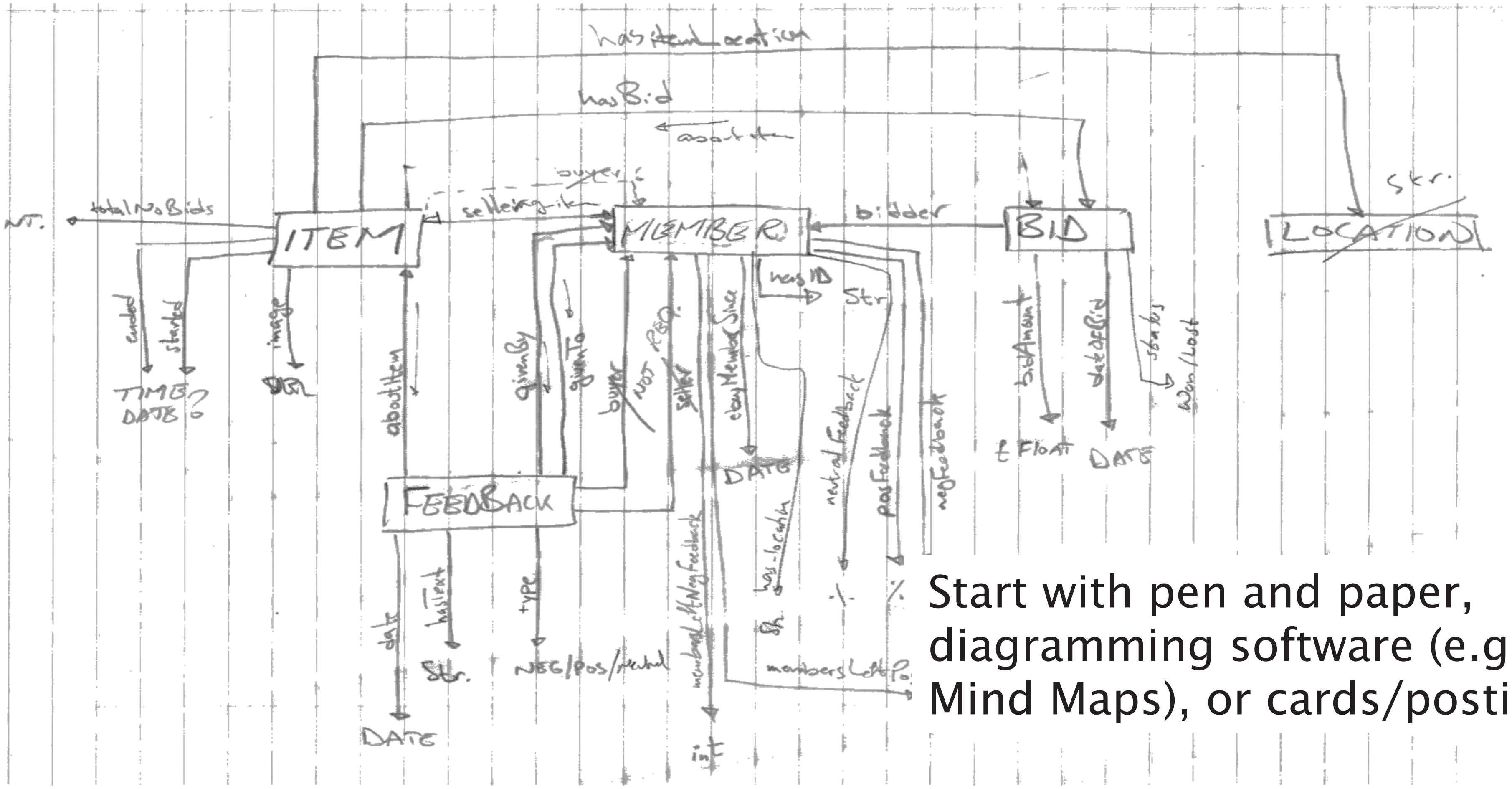
Conceptualisation

Identify the concepts to include in your ontology, and how they relate to each other

- Depends on your defined scope and competency questions
- Define unambiguous names and descriptions for classes and properties (more on this in Documentation)
- Reach agreement (the hard part!)

The best tool to use:

Conceptualisation



Start with pen and paper, diagramming software (e.g. Visio, Mind Maps), or cards/postit notes

Conceptualisation: Reuse

Ontologies are meant to be reusable!

- Technology for reusing ontologies is still limited

Always a good idea to check any existing models or ontologies

- Check your database models or off-the-shelf ontologies

Check existing ontologies

- No need to reinvent the wheel, unless it is easier to do so!
- Ontology search engines
  - Swoogle, Watson, lodlaundromat

What can you reuse?

- Databases
- Vocabularies
- Ontologies
  - Some much re-used ontologies
    - For describing persons: FOAF
    - For describing documents: Dublin Core
    - For describing social media: SIOC
    - For describing vocabulary hierarchies: SKOS
    - For describing e-commerce: Good Relations
    - For Web metadata: schema.org
    - …

Formalisation

- Moving from a list of concepts to a formal model
- Define the hierarchy of concepts and relations
- Also note down any restrictions
  - E.g. NonProfitOrg isDisjoint from ProfitOrg
  - An email address is unique

Formalisation: Building the Class Hierarchy

Top-down

- Start with the most general classes and finish with the most detailed classes

Bottom-up

- Start with the most detailed classes and finish with the most general ones

Middle-out

- Start with the most obvious classes
- Group as required
- Then go upwards and downwards to the more general and more detailed classes respectively
- Good for controlling scope and detail

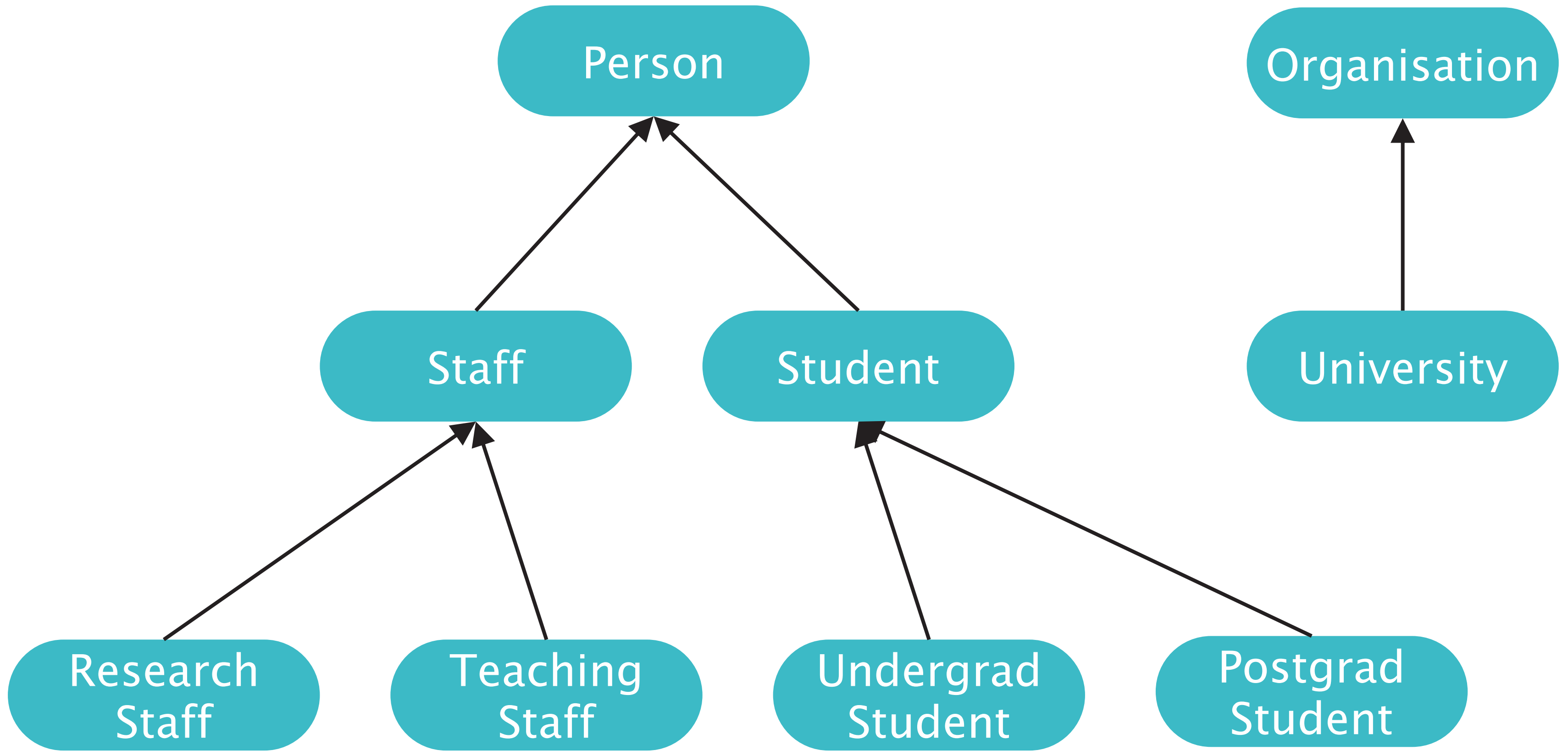
Formalisation: Middle-Out Approach

Staff

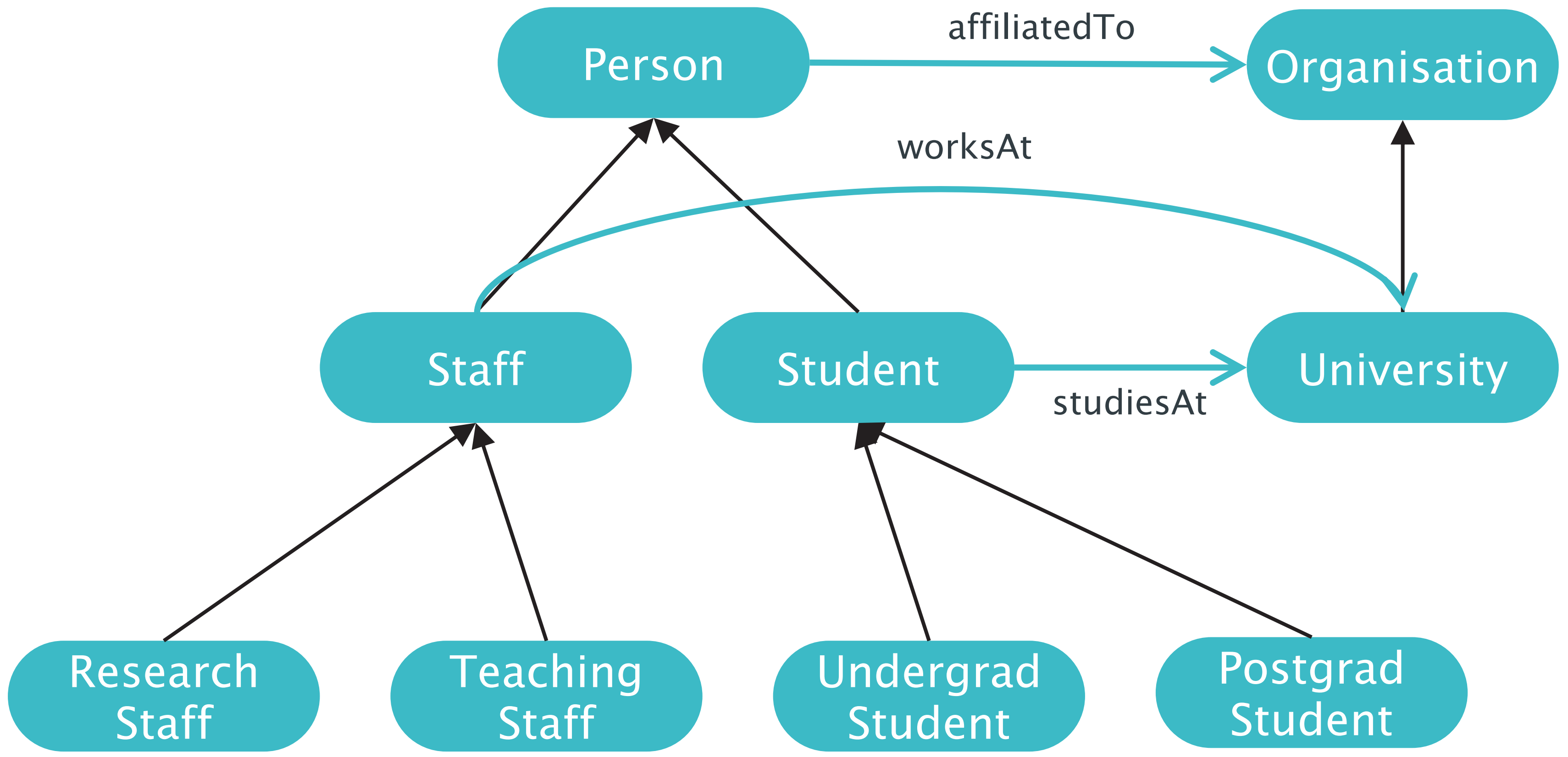
Student

University

Formalisation: Middle-Out Approach



Formalisation: Middle-Out Approach

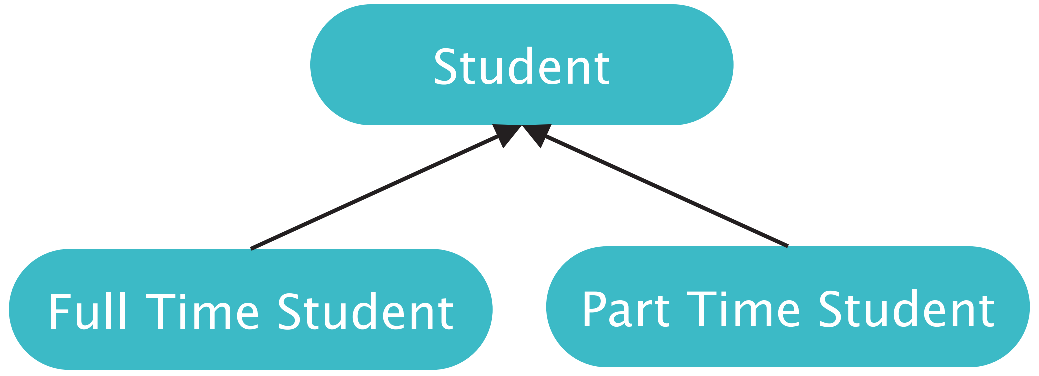


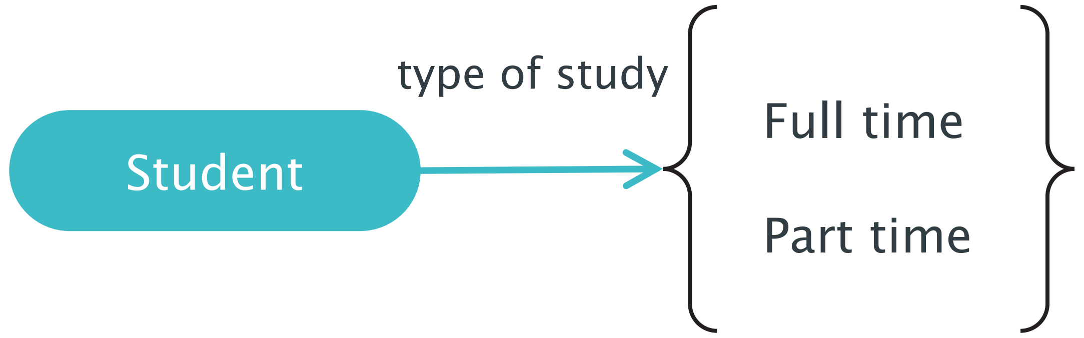
Formalisation: Naming Conventions

- Not rules, but conventions
- Avoid spaces and uncommon delimiters in class and relation names
  - e.g. use PetFood or Pet_Food instead of Pet Food or Pet*Food
- Capitalise each word in a class name
  - e.g. PetFood instead of Petfood or even petfood
- Start names of relations with a lowercase letter
  - e.g. pet_owner, petOwner
- Use singular nouns for classes
  - e.g. Pet, Person, Shop

Formalisation: Class or Relation?

Is it a class or a relation?



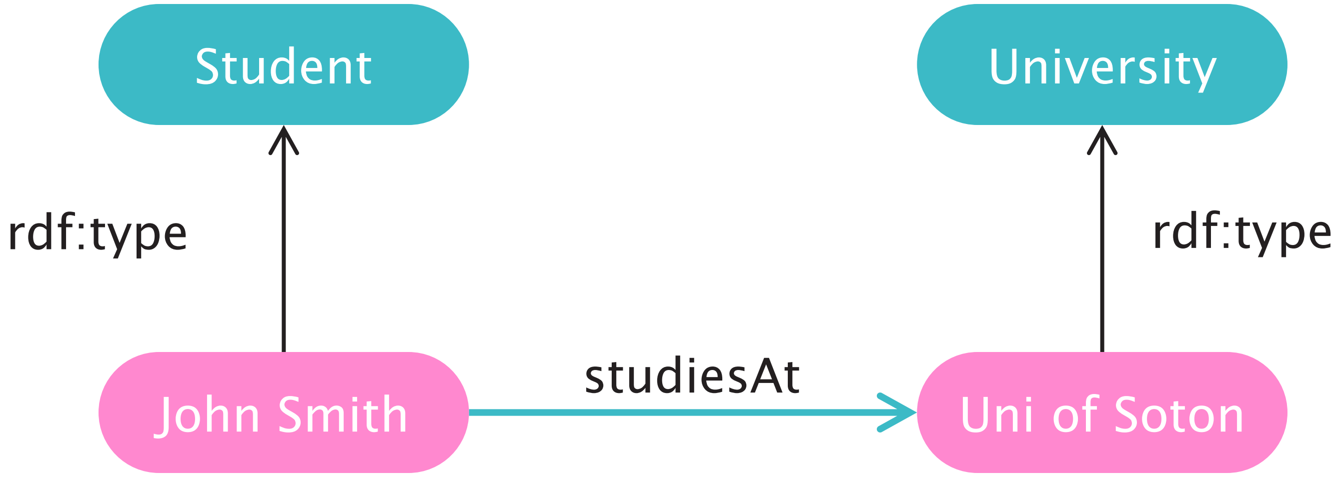


It depends!

If the subclass doesn’t need any new relations (or restrictions), then consider making it a relation

Formalisation: Class or Instance?

Is it a class or an instance?

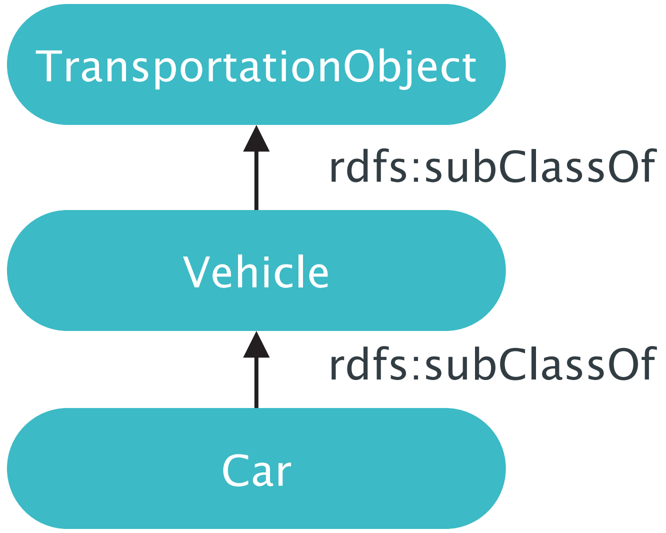


- If it can have its own instances, then it should be a class
- If it can have its own subclasses, then it should be a class

Formalisation: Transitivity of Class Hierarchy

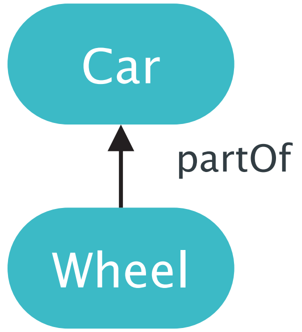
subClassOf relation is always transitive

- Car is a subclass of Vehicle
- Vehicle is a subclass of TransportationObject
- Any instance of Car is also a TransportationObject



subClassOf is not the same as "part of"

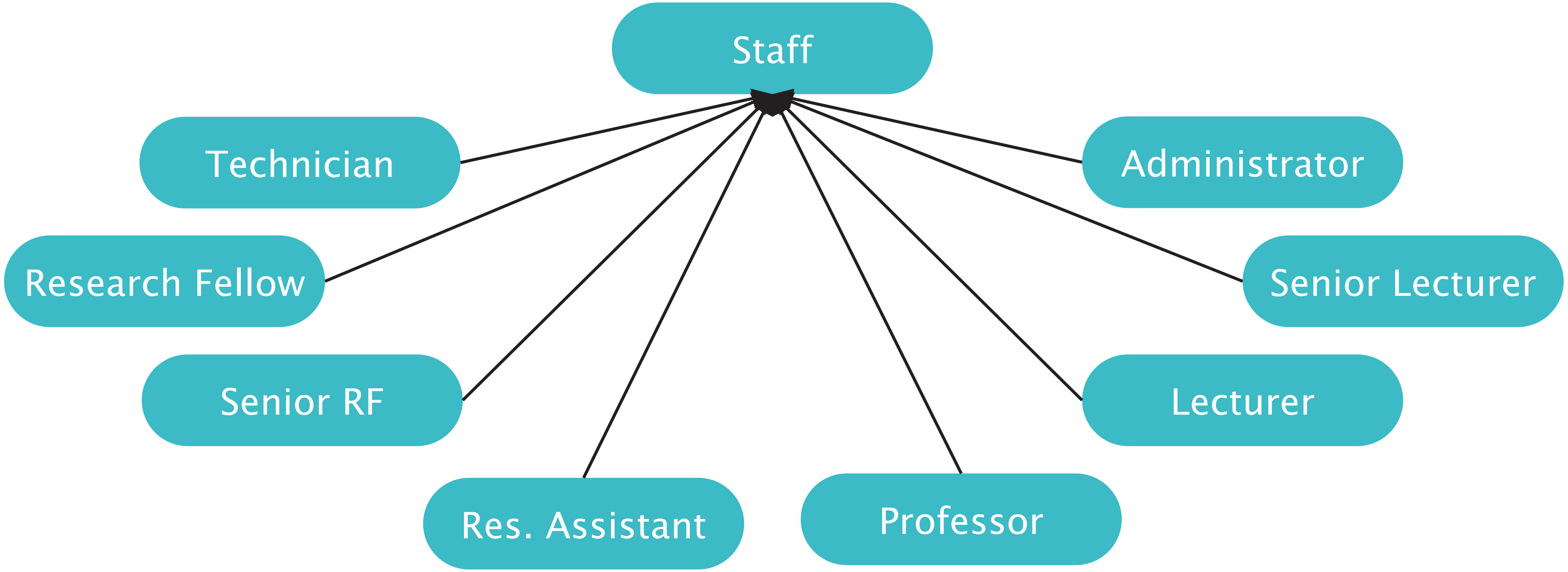
- (see meronymy pattern later this lecture)



Formalisation: Tidy Your Hierarchy

Avoid subClassOf clutter!

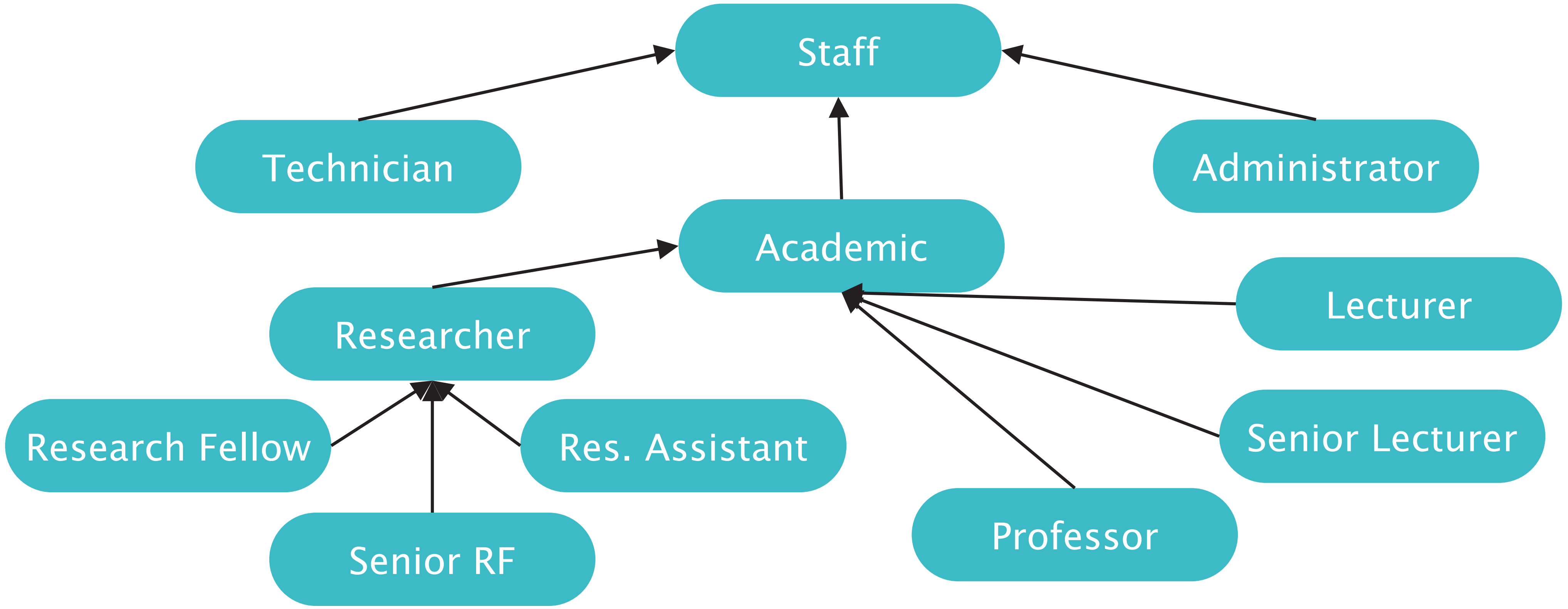
- Break down your hierarchy further if you have too many direct subclasses of a class



Formalisation: Tidy Your Hierarchy

Avoid subClassOf clutter!

- Break down your hierarchy further if you have too many direct subclasses of a class



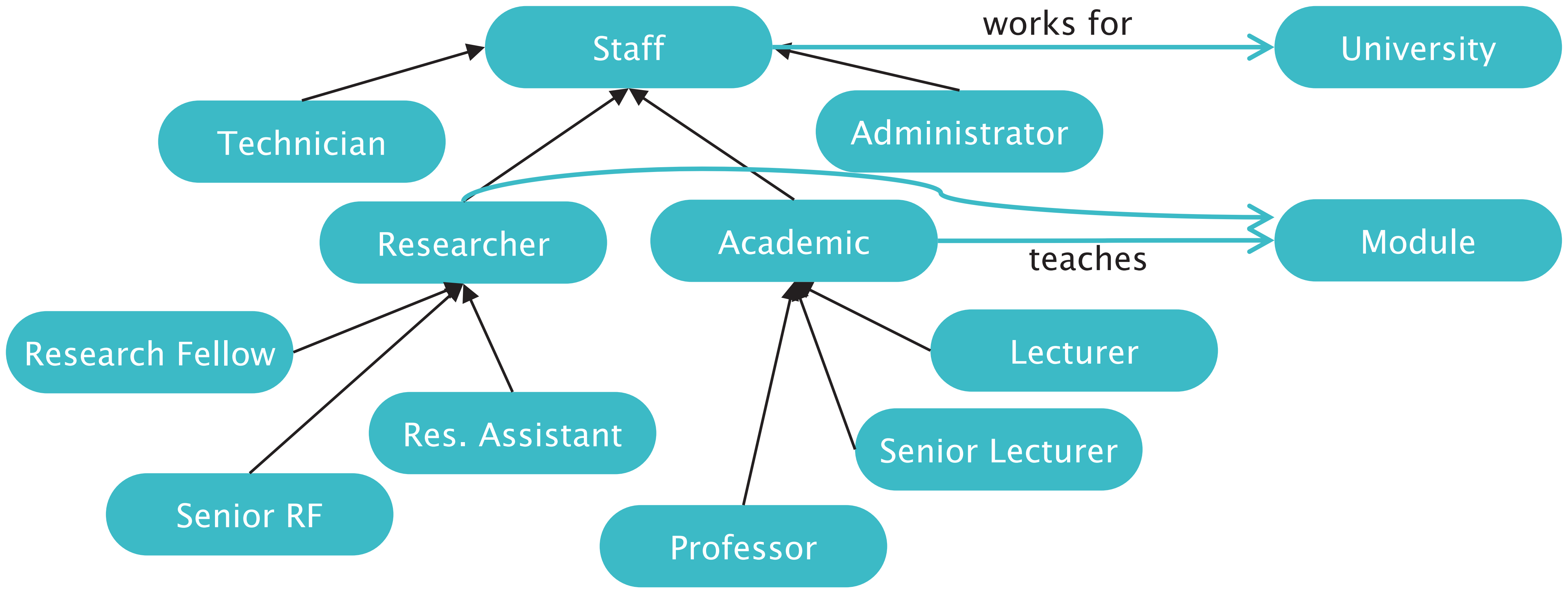
Formalisation: Where to Point my Relation?

Relations should point to the most general class

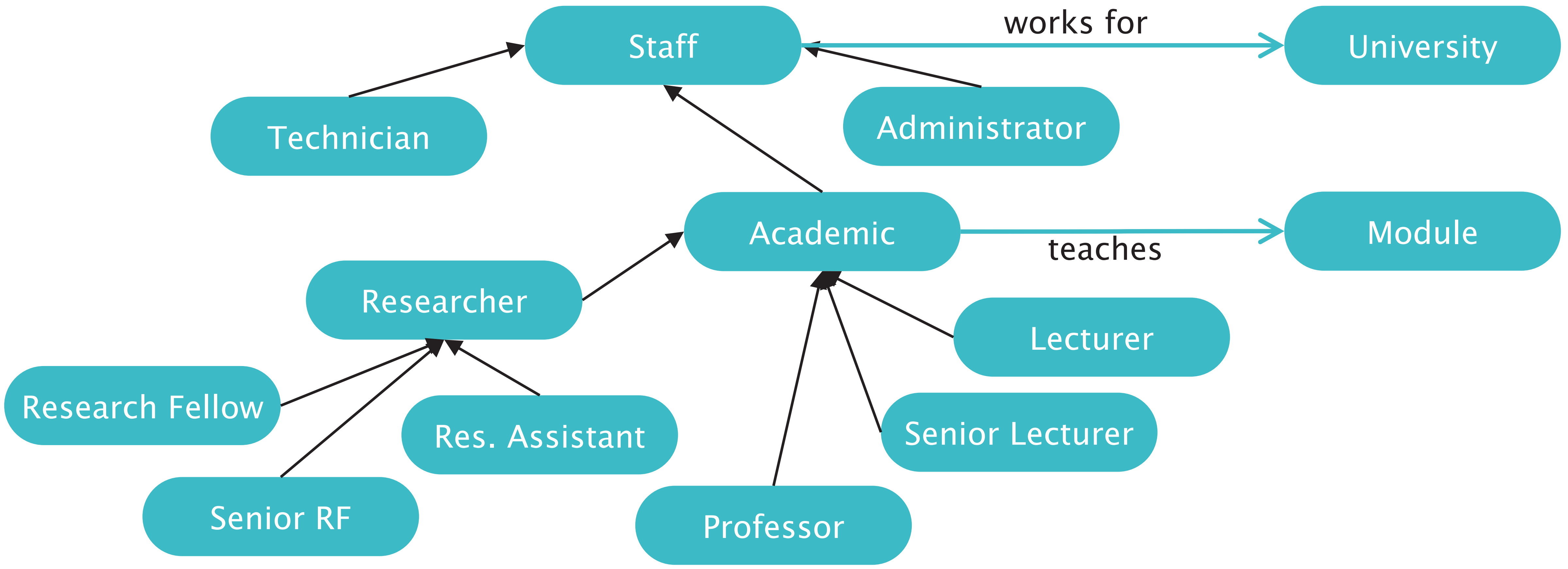
- But not too general
  - e.g relations pointing to Thing!
- And not too specific
  - e.g. relations pointing to the bottom of the hierarchy

As a rule of thumb, if the domain or range of a relation is a disjunction (union) of classes, some refactoring is probably required

Formalisation: Where to Point my Relation?



Formalisation: Where to Point my Relation?



Implementation

- Choose a language
    - e.g. RDFS, OWL…
- Implement it with an ontology editor
    - e.g. Protégé, SWOOP, TopQuadrant
- Edit the class hierarchy
- Add relationships
- Add restrictions
- Select appropriate value types, cardinality, etc
- Use a reasoner to check the consistency of your ontology
    - e.g. Racer, Pellet, Fact++, HermiT
    - Best to do this as you go along – easier to trace bugs in your modelling

Evaluation: Verification

Is your ontology correct?

- Is it syntactically correct?
- Is it consistent?

Implementing the ontology in an ontology editor helps to get the syntax correct

Using a reasoner helps you check that it's consistent

You can also validate your OWL ontology online:

- http://visualdataweb.de/validator/

Evaluation: Validation

Does your ontology successfully do what you set out to do?

Check the ontology against your competency questions

- Write the questions in SPARQL or in similar query languages
- Can you get the answers you need?
- Is it quick enough?
- Add additional properties or restructure the ontology to increase efficiency?

Documentation

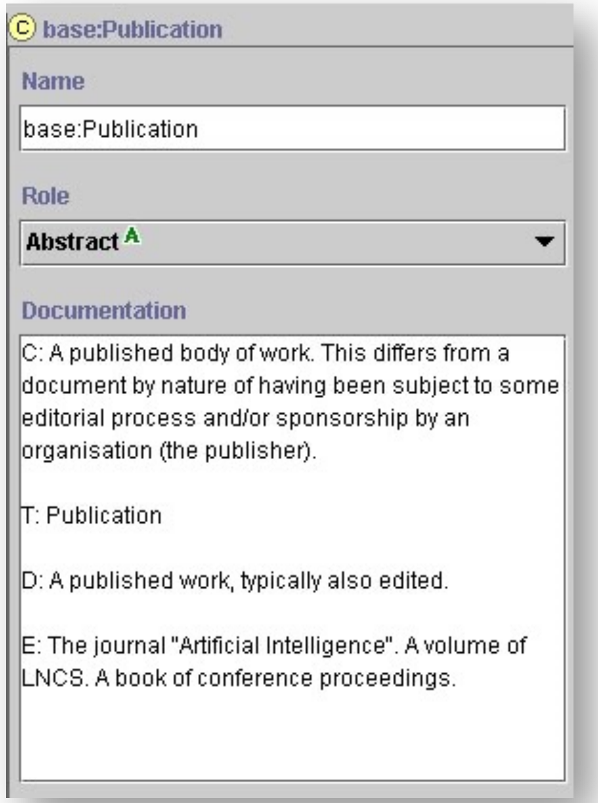
Documenting the design and implementation rational is crucial for future usability and understanding of the ontology

- Rational, design options, assumptions, decisions, examples, etc.

Structured documentation may clarify these assumptions

Douglas Skuce proposed a convention for structured documentation of ontological assumptions in 1995

- Conceptual assumptions (C)
  (long definition, comparing with other classes/properties)
- Terminological assumptions (T) (alternative terms used)
- Definitional assumption (D) (short definition)
- Examples (E)



Name: base:Publication

Role: Abstract

Documentation:

C: A published body of work. This differs from a document by nature of having been subject to some editorial process and/or sponsorship by an organisation (the publisher).

T: Publication

D: A published work, typically also edited.

E: The journal "Artificial Intelligence". A volume of LNCS. A book of conference proceedings.

Structured documentation

Instead of putting C/T/D/E into a single rdfs:comment, structure the metadata using appropriate properties from RDFS and SKOS (import SKOS into your ontology)

Conceptual assumptions (C)

- skos:scopeNote, rdfs:comment

Terminological assumptions (T)

- skos:prefLabel, skos:altLabel, rdfs:label

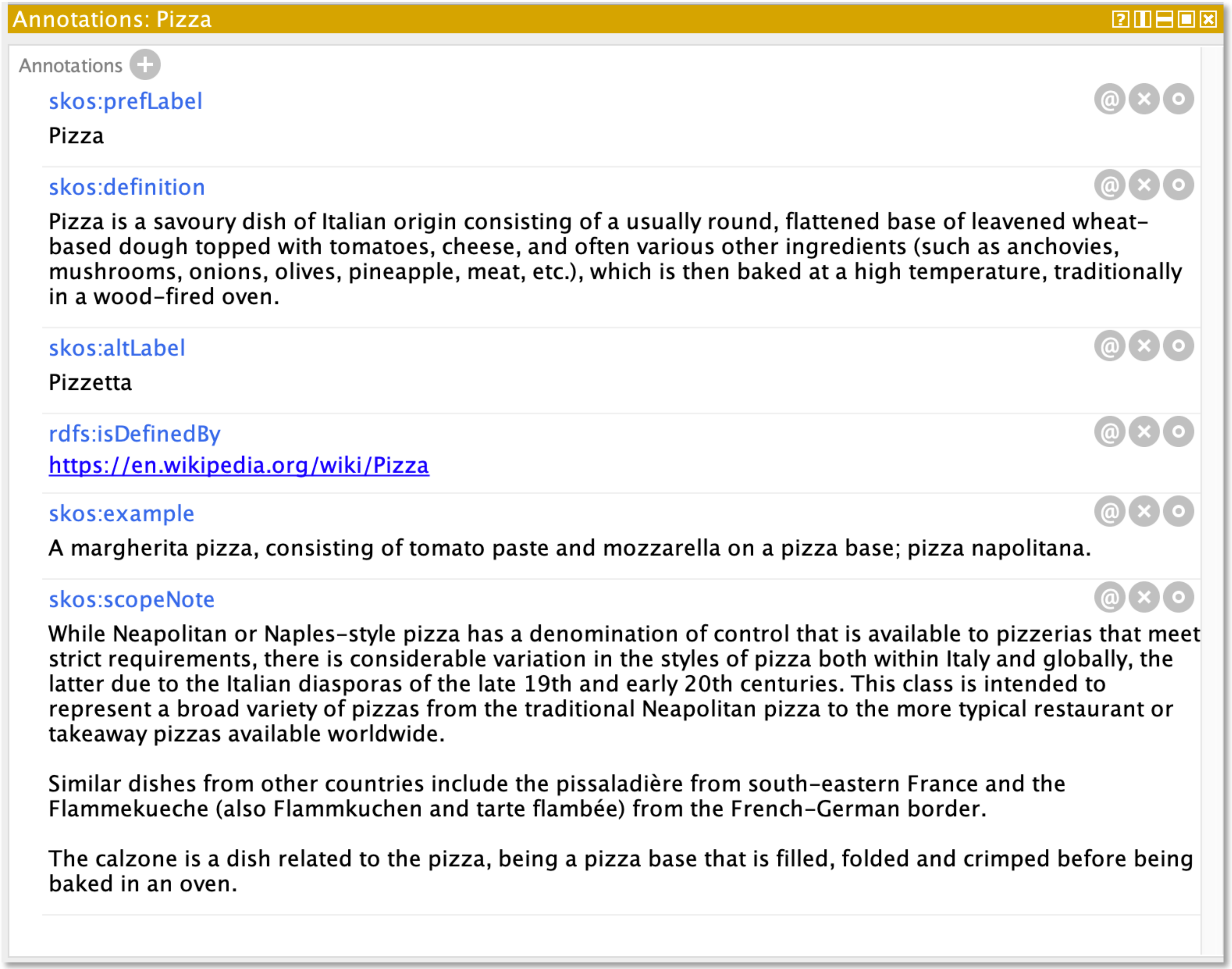
Definitional assumptions (D)

- skos:definition

Examples (E)

- skos:example

Use rdfs:isDefinedBy to indicate if definition is taken from an external source


Annotations: Pizza

Annotations

skos:prefLabel

Pizza

skos:definition

Pizza is a savoury dish of Italian origin consisting of a usually round, flattened base of leavened wheat-based dough topped with tomatoes, cheese, and often various other ingredients (such as anchovies, mushrooms, onions, olives, pineapple, meat, etc.), which is then baked at a high temperature, traditionally in a wood-fired oven.

skos:altLabel

Pizzetta

rdfs:isDefinedBy

https://en.wikipedia.org/wiki/Pizza

skos:example

A margherita pizza, consisting of tomato paste and mozzarella on a pizza base; pizza napolitana.

skos:scopeNote

While Neapolitan or Naples-style pizza has a denomination of control that is available to pizzerias that meet strict requirements, there is considerable variation in the styles of pizza both within Italy and globally, the latter due to the Italian diasporas of the late 19th and early 20th centuries. This class is intended to represent a broad variety of pizzas from the traditional Neapolitan pizza to the more typical restaurant or takeaway pizzas available worldwide.

Similar dishes from other countries include the pissaladière from south-eastern France and the Flammekueche (also Flammkuchen and tarte flambée) from the French-German border.

The calzone is a dish related to the pizza, being a pizza base that is filled, folded and crimped before being baked in an oven.

Summary

Ontology construction is an iterative process

- Build ontology, try to use it, fix errors, extend, use again, and repeat

There is no single correct model for your domain

- The same domain may be modelled in several ways

Following best practices helps to build good ontologies

- Well scoped, documented, structured

Reuse existing ontologies if possible

- Check your database models and existing ontologies
- Reuse or learn from existing representations
- (most ontology editing tools don’t yet provide good support for reuse)

Common Pitfalls

Over scaling and complicating your ontology

- Need to learn when to stop expanding the ontology

Lack of documentation

- For the design rationale, vocabulary and structure decisions, intended use, etc.

Redundancy

- Increase chances of inconsistencies and maintenance cost

Using ambiguous terminology

- Others might misinterpret your ontology
- Mapping to other ontologies will be more difficult

University of Southampton

Next Lecture:
Ontology Design Patterns