Foundations of the Semantic Web: Ontology Engineering

Building Ontologies 1

Alan Rector & colleagues

Developing an Ontology Start at the Beginning

- You now have all you need to implement simple existential ontologies, so let’s go back to the beginning
- The goal for the example ontology is to build an ontology of animals to index a children’s book of animals
- The goal for the lab ontology is for you to build an ontology for the CS department and eventually for the University

Steps in developing an Ontology

1. **Establish the purpose**
   - Without purpose, no scope, requirements, evaluation,
2. **Informal/Semiformal knowledge elicitation**
   - Collect the terms
   - Organise terms informally
   - Paraphrase and clarify terms to produce informal concept definitions
   - Diagram informally
3. **Refine requirements & tests**

Steps in implementing an Ontology

4. **Implementation**
   - ***Paraphrase and comment at each stage <u>before</u> implementing***
   - Develop normalised schema and skeleton
   - Implement prototype recording the *intention as a paraphrase*
     - Keep track of what you meant to do so you can compare with what happens
       - Implementing logic-based ontologies is programming
   - Scale up a bit
     - Check performance
   - Populate
     - Possibly with help of text mining and language technology
5. **Evaluate & quality assure**
   - Against goals
   - Include tests for evolution and change management
   - Design regression tests and “probes”
6. **Monitor use and evolve**
   - ***Process not product!***

Purpose & scope of the animals ontology

- **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.

Collect the concepts

- Card sorting is often the best way
  - Write down each concept/idea on a card
  - Organise them into piles
  - Link the piles together
  - Do it again, and again
  - Works best in a small group

  - In the lab we will provide you with some pre-printed cards and many spare cards
    - Work in pairs or triples

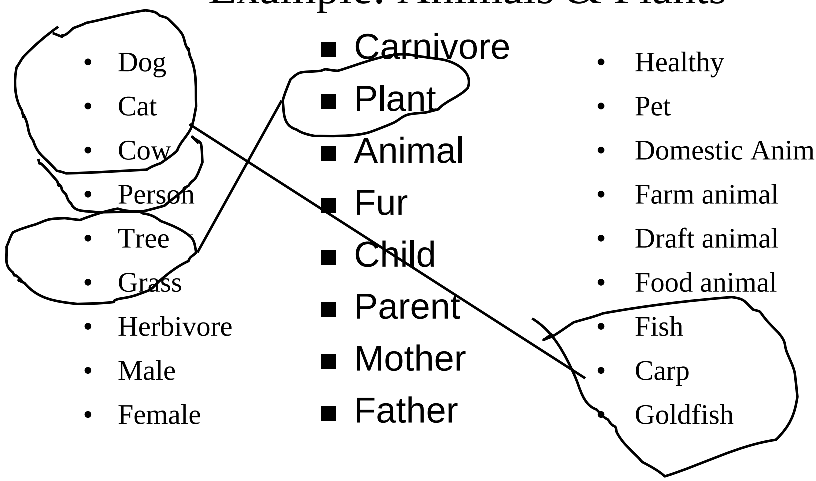
Example: Animals & Plants

- Dog
- Cat
- Cow
- Person
- Tree
- Grass
- Herbivore
- Male
- Female

- ■ Carnivore
- ■ Plant
- ■ Animal
- ■ Fur
- ■ Child
- ■ Parent
- ■ Mother
- ■ Father

- Dangerous
- Pet
- Domestic Animal
- Farm animal
- Draft animal
- Food animal
- Fish
- Carp
- Goldfish

Organise the concepts
Example: Animals & Plants



- 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

Extend the concepts "Laddering"

- 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
  - Cat, Dog, Cow, Person → Mammal
    - Others might be Goat, Sheep, Horse, Rabbit,…
  - Cow, Goat, Sheep, Horse → Hoofed animal ("Ungulate")
    - What others are there? Do they divide amongst themselves?
  - Wild, Domestic → Demoestication
    - What other states – "Feral" (domestic returned to wild)

Vocabulary note:
"Sibling" = "brother or sister"

Choose some main axes

- **Add abstractions where needed**
  - **e.g. "Living thing"**
- **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**
      - **Try to define a dog precisely – very difficult**
        - **A "natural kind"**
- **make names explicit**

Choose some main axes Add abstractions where needed; identify relations; Identify definable things, make names explicit

- 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
  - Sex
    - Male
    - Female
  - Age
    - Adult
    - Child

- Relations
  - eats
  - owns
  - parent-of
  - …
- Definable
  - Carinvore
  - Herbivore
  - Child
  - Parent
  - Mother
  - Father
  - Food Animal
  - Draft Animal

Self_standing_entities

- Things that can exist on there own nouns
  - People, animals, houses, actions, processes, …
    - Roughly nouns
- Modifiers
  - Things that modify ("inhere") in other things
    - Roughly adjectives and adverbs

Reorganise everything but “definable” things into pure trees – these will be the “primitives”

- Self_standing
  - Living Thing
    - Animal
      - Mammal
        - Cat
        - Dog
        - Cow
        - Person
        - Pig
      - Fish
        - Carp
          Goldfish
    - Plant
      - Tree
      - Grass
      - Fruit

- Modifiers
  - **Domestication**
    - **Domestic**
    - **Wild**
  - **Use**
    - **Draft**
    - **Food**
    - **pet**
  - **Risk**
    - **Dangerous**
    - **Safe**
  - **Sex**
    - **Male**
    - **Female**
  - **Age**
    - **Adult**
    - **Child**

- Relations
  - eats
  - owns
  - parent-of
  - …
- Definables
  - Carnivore
  - Herbivore
  - Child
  - Parent
  - Mother
  - Father
  - Food Animal
  - Draft Animal

If anything needs clarifying, add a text comment

- Self_standing
  - Living Thing
    - Animal
      - Mammal
        - Cat
        - Dog
        - Cow
        - Person
        - Pig
      - Fish
        - Carp

          Goldfish
    - Plant
      - Tree
      - Grass
      - Fruit

– ***The abstract ancestor concept including all living things – restrict to plants and animals for now”***

Identify the domain and range constraints for properties

- Animal *eats* Living_thing
  - *eats* domain: Animal;
    range: Living_thing
- Person *owns* Living_thing except person
  - *owns* domain: Person
    range: Living_thing & not Person
- Living_thing *parent_of* Living_thing
  - *parent_of:* domain: Animal
    range: Animal

If anything is used in a special way, add a text comment

- Animal *eats* Living_thing
  - *eats* domain: Animal;
    range: Living_thing

— ***ignore difference between parts of living things and living things also derived from living things***

For definable things

- Paraphrase and formalise the definitions in terms of the primitives, relations and other definables.
- Note any assumptions to be represented elsewhere.
  - Add as comments when implementing
- *“A ‘Parent’ is an animal that is the parent of some other animal” (Ignore plants for now)*
  - Parent =
    Animal and *parent_of* some Animal
- *“A ‘Herbivore’ is an animal that eats only plants” (NB All animals eat some living thing)*
  - Herbivore=
    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

Paraphrases and Comments

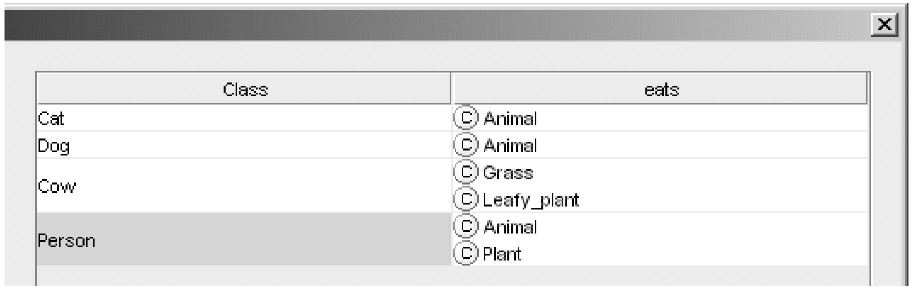
- Write down the paraphrases and put them in the comment space.
  - We can show you how to make the comment space bigger to make it easier.
- Without a paraphrase, we cannot tell if we disagree on
  - What you meant to represent
  - How you represented it
- ***Without a paraphrase we will mark down by at least half and give no partial credit***
  - We will try to understand what you are doing, but we cannot read your minds.

Which properties can be filled in at the class level now?

- What can we say about *all* members of a class?
  - *eats*
    - *All cows eat some plants*
    - *All cats eat some animals*
    - *All pigs eat some animals & eat some plants*

Fill in the details

(can use property matrix wizard)



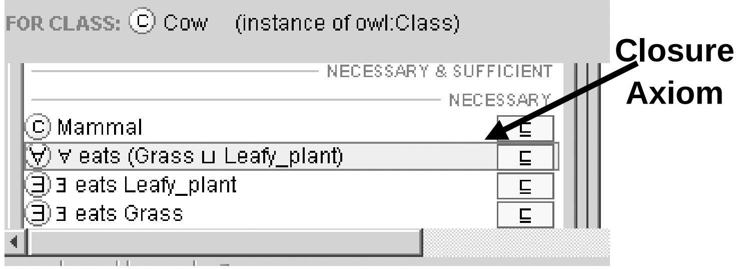
Class	eats
Cat	Ⓒ Animal
Dog	Ⓒ Animal
Cow	Ⓒ Grass Ⓒ Leafy_plant
Person	Ⓒ Animal Ⓒ Plant

Check with classifier

- Cows should be Herbivores
  - Are they? why not?
    - What have we said?
      - Cows are animals and, *amongst other things*, eat *some* grass and eat some leafy_plants
    - What do we need to say: Closure axiom
      - Cows are animals and, *amongst other things*, eat *some* plants and eat *only* plants
        - » (See "Vegetarian Pizzas" in OWL tutorial)

Closure Axiom

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



FOR CLASS: Ⓒ Cow (instance of owl:Class)

NECESSARY & SUFFICIENT

NECESSARY

Ⓒ Mammal ⊑

∀ eats (Grass ⊔ Leafy_plant) ⊑

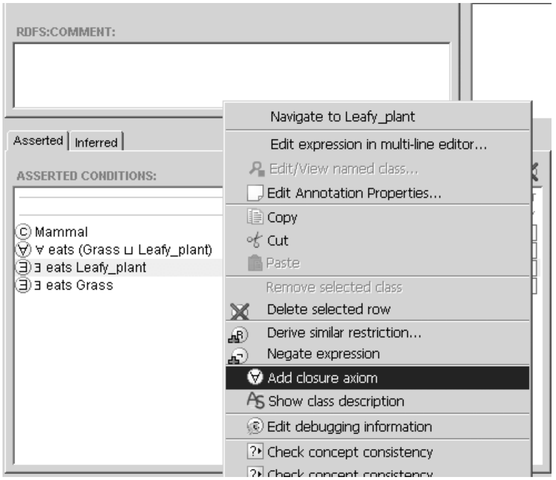
∃ eats Leafy_plant ⊑

∃ eats Grass ⊑

Closure Axiom

In the tool

- Right mouse button short cut for closure axioms
  - for any existential restriction



RDFS:COMMENT:

ASSERTED CONDITIONS:

Ⓒ Mammal

∀ eats (Grass ⊔ Leafy_plant)

∃ eats Leafy_plant

∃ eats Grass

Asserted
Inferred
Navigate to Leafy_plant
Edit expression in multi-line editor...
Edit/View named class...
Edit Annotation Properties...
Copy
Cut
Paste
Remove selected class
Delete selected row
Derive similar restriction...
Negate expression
Add closure axiom
Show class description
Edit debugging information
Check concept consistency
Check concept consistency

Open vs Closed World reasoning

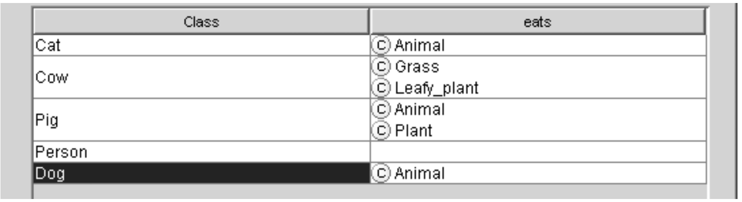
- **Open world reasoning**
  - Negation as contradiction
    - Anything might be true unless it can be proven false
      - Reasoning about *any world consistent with this one*
- **Closed world reasoning**
  - Negation as failure
    - Anything that cannot be found is false
      - Reasoning about *this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

Normalisation and Untangling

Let the reasoner do multiple classification

- **Tree**
  - Everything has just one parent
    - A ‘strict hierarchy’
- **Directed Acyclic Graph (DAG)**
  - Things can have multiple parents
    - A ‘Polyhierarchy’
- **Normalisation**
  - Separate *primitives* into disjoint trees
  - Link the trees with definitions & restrictions
    - Fill in the values
  - Let the classifier produce the DAG

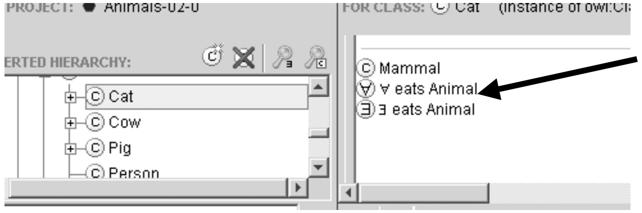
Tables are easier to manage than DAGs / Polyhierarchies



Class	eats
Cat	Ⓒ Animal
Cow	Ⓒ Grass Ⓒ Leafy_plant
Pig	Ⓒ Animal Ⓒ Plant
Person	
Dog	Ⓒ Animal

…and get the benefit of inference:
Grass and Leafy_plants are both kinds of Plant

Remember to add any closure axioms



PROJECT: ● Animals-02-0

ERTED HIERARCHY:

- Cat
- Cow
- Pig
- Person

FOR CLASS: Ⓒ Cat (instance of owl:Cl

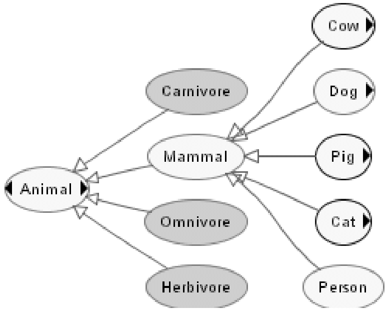
- Ⓒ Mammal
- ∀ eats Animal
- ∃ eats Animal

Closure Axiom

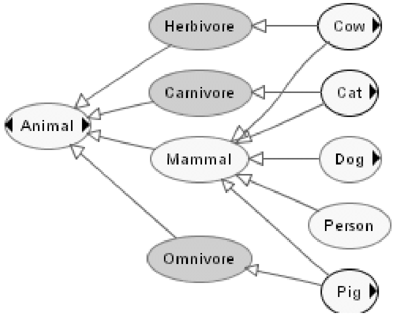
Then let the reasoner do the work

Normalisation: From Trees to DAGs

- Before classification
  - A tree



- After classification
  - A DAG
    - **D**irected **A**cyclic **G**raph



Summary: Normalised Ontology Development

- **Identify the self-standing primitives**
  - Comment any that are not self-evident
- **Separate them into trees**
  - You may have to create some ‘roles’ or other auxiliary concepts to do so
- **Identify the relations**
  - Comment any that are not self evident
- **Create the descriptions and definitions**
  - Provide a paraphrase for each
- **Identify how key items should be classified –**
  - Define regression tests
- **Use classifier to form a DAG**
- **Check if tests are satisfied**

Part II – Useful Patterns (continued)

- Upper ontologies & Domain ontologies
- Building from trees and untangling
- Using a classifier
- Closure axioms & Open World Reasoning
- ***Specifying Values***
- n-ary relations
- Classes as values – using the ontology

Examine the modifier list

- Modifiers
  - **Domestication**
    - **Domestic**
    - **Wild**
  - **Use**
    - **Draft**
    - **Food**
  - **Risk**
    - **Dangerous**
    - **Safe**
  - **Sex**
    - **Male**
    - **Female**
  - **Age**
    - **Adult**
    - **Child**

- Identify modifiers that have mutually exclusive values
  - Domestication
  - Risk
  - Sex
  - Age
- Make meaning precise
  - Age → Age_group
- NB Uses are not mutually exclusive
  - Can be both a draft (pulling) and a food animal

Extend and complete lists of values

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

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

Note any hierarchies of values

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

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

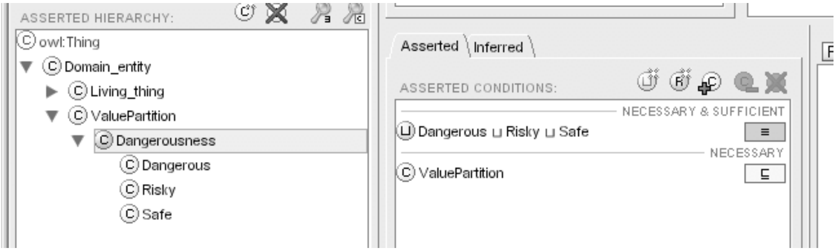
Specify Values for each:
Two methods

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

Method 1: Value Partitions- example “Dangerousness”

- A parent quality – Dangerousness
- Subqualities for each degree
  - Dangerous, Risky, Safe
- All subqualities disjoint
- Subqualities ‘cover’ parent quality
  - Dangerousness = Dangerous OR Risky OR Safe
- A functional property has_dangerousness
  - Range is parent quality, e.g. Dangerousness
  - Domain must be specified separately
- Dangerous_animal =
  Animal *and* has_dangerousness *some* Dangerous

as created by Value Partition wizard

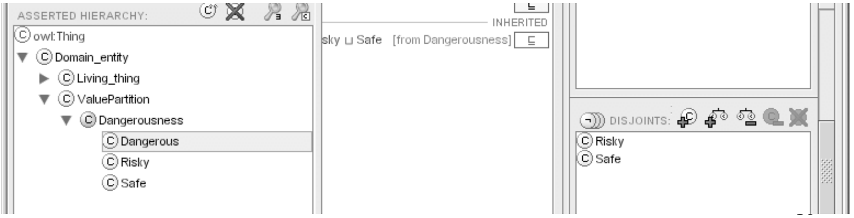

ASSERTED HIERARCHY:
- owl:Thing
- Domain_entity
  - Living_thing
  - ValuePartition
    - Dangerousness
      - Dangerous
      - Risky
      - Safe

Asserted | Inferred

ASSERTED CONDITIONS:

NECESSARY & SUFFICIENT
Dangerous ⊔ Risky ⊔ Safe ≡

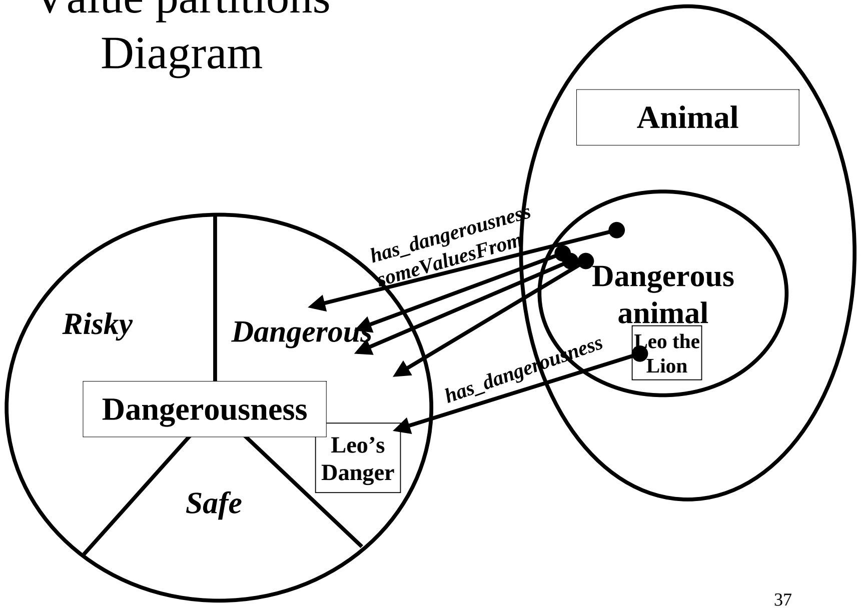
NECESSARY
ValuePartition ⊑


ASSERTED HIERARCHY:
- owl:Thing
- Domain_entity
  - Living_thing
  - ValuePartition
    - Dangerousness
      - Dangerous
      - Risky
      - Safe

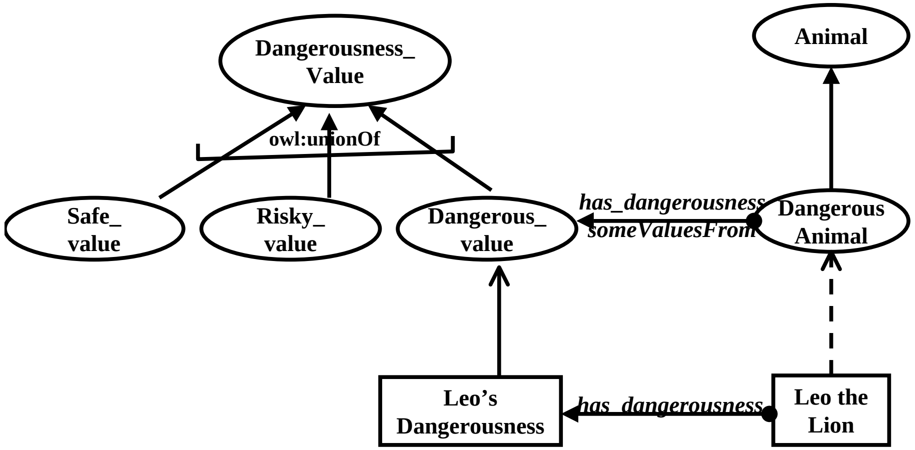
INHERITED
sky ⊔ Safe [from Dangerousness] ⊑

DISJOINTS:
Risky
Safe

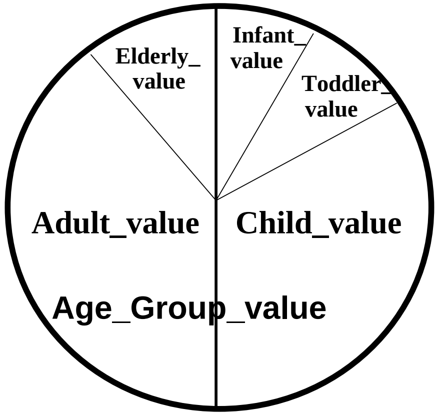
Value partitions Diagram



Value partitions UML style



Picture of subdivided value partition



Method 2: Value sets –

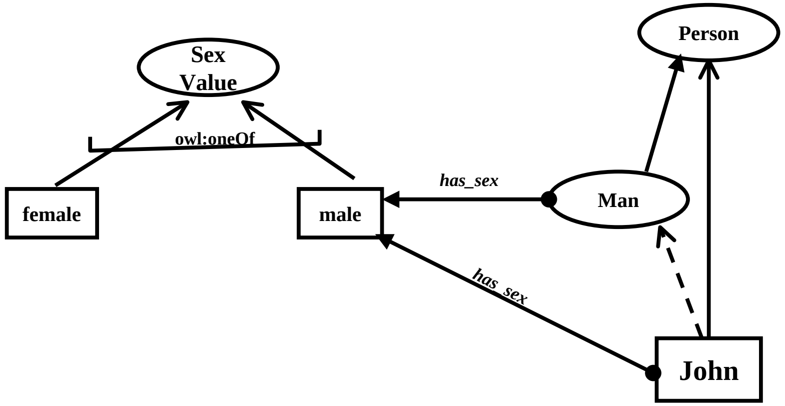
Example Sex

- There are only two sexes
  - Can argue that they are things
    - "Administrative sex" definitely a thing
    - "Biological sex" is more complicated

Method 2: Value sets-
example “Sex”

- A parent quality – Sex_value
- Individuals for each value
  - male, female
- Values all different (NOT assumed by OWL)
- Value type is enumeration of values
  - Sex_value = {male, female}
- A functional property has_sex
  - Range is parent quality, e.g. Sex_value
  - Domain must be specified separately
- Male_animal =
  Animal *and* has_sex *is* male

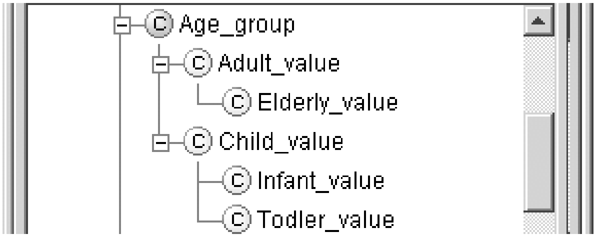
Value sets UML style



Issues in specifying values

- Value Partitions
  - Can be subdivided and specialised
  - Fit with philosophical notion of a quality space
  - Require interpretation to go in databases as values
    - **in theory but rarely considered in practice**
  - Work better with existing classifiers in OWL-DL
- **Value Sets**
  - Cannot be subdivided
  - Fit with intuitions
  - More similar to data bases – no interpretation
  - Work less well with existing classifiers

Value partitions – practical reasons for subdivisions



- Age_group
  - Adult_value
    - Elderly_value
  - Child_value
    - Infant_value
    - Todler_value

- “All elderly are adults”
- “All infants are children”
- etc.

- See also “Normality_status” in http://www.cs.man.ac.uk/~rector/ontologies/mini-top-bio
  - One can have complicated value partitions if needed.

Summary of Specifying Values

- Principles
  - Values distinct
    - Disjoint if value partition/classes
    - allDifferent if value sets/individuals
  - Values “cover” type
    - Covering axiom if value partition/classes
      - Quality = VP$_1$ OR VP$_2$ OR VP$_3$ OR…OR VP$_n$
    - Enumeration if value sets/individuals
      - Quality = {v$_1$ v$_2$ v$_3$ … v$_n$}
  - Property usually functional
    - But can have multi-valued cases occasionally
- Practice
  - In this module we recommend you use Value Partitions in all cases for specifying values
    - Works better with the reasoner
    - We have a Wizard to make it quick

“Roles”

- To keep primitives in disjoint
  - need to distinguish the roles things play in different situations from what they are
  - e.g. “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

Roles usually summarise relations

- “to play the role of pet” is to say that there is somebody for whom the animal is a pet
- “to play the role of doctor” is to say that there is somebody for whom the person is acting as the “doctor” – or some “situation” in which they play that role

But we often do not want to explain the situation or relation completely

“Roles” and “Untangling”

- Animal
  - Draft_animal
    - Cow
    - Horse
    - Dog
  - Food_animal
    - Cow
    - Horse
  - Pet_animal
    - Horse
    - Dog

- Animal
  - Mammal
    - Cow
    - Horse
    - Dog
- Animal_use_role
  - Food_role
  - Pet_role
  - Draft_role

Draft_animal =
Animal &
has_role some Daft_role

Food_animal =
Animal &
has_role some Food_role

Pet_animal =
Animal &
has_role some Pet_role

Vocabulary note:
“Draft” means pulling – as in pulling a cart or plough

Logical approximations for roles

- Cow plays_role some Draft_role
  - Means *all* cows play some draft role
    - Too strong a statement
- Solutions
  - Ignore the problem for purposes of the ontology
  - Replace *has_role* by *may_have_role*
    - Still to strong but probably the a pragmatic answer
- If classifying instances need both
  - All cows may have some draft role
    Cow → may_have Draft_role
    - Just those that actually do are known as Draft_cows
      - Draft_Cow = Cow & has_role Draft_role

Example of language problems

- "DraftHorse" and "Draft_horse"
  - Some breeds of horses were bred for draft work
    - Known officialy as "Draft horses"
      - The words have taken on a "idiomatic" meaning
        - » No longer mean what they say
        - » Other examples "Blue bird" vs "Bluebird"
          "Black berry" vs "Blackberry"
          …
- Horse → *may_have_role* some Draft_role
  - DraftHorse rdf:comment "Draft breed horse"
  - Draft_horse = Horse AND *has_role* some Draft_role
    rdf:comment: "Horse actually used for draft work"

Separate Language Labels from Ontology

- OWL/RDF mechanisms weak
  - rdf:label
    - Allows a language or sublanguage tag, but merely an annotation
- Better to be maximally explicit in internal names for concepts
  - Better to be *not understood* than to be *misunderstood*
- Change DraftHorse to Draft_breed_horse
  - rdf:label "Draft horse"

Ontology engineering

- Provide paraphrases and comments for all classes
- Provide probe classes and testing framework
  - Probe classes: extra classes that either should or should not be satisfiable or classified in a particular place
    - The tool lets you hide probe classes from user and delete them from final export
  - Can also put debugging information on other classes
    - Testing framework will report violations
- This is still new software, so let us know if it doesn't work or how it could be improved.

Lab Exercise

- Take cards for University ontology to produce an ontology for the university including the personnel department’s equal opportunities officer
- Group the cards and form initial hierarchies
  - Separate likely primitives, modifiers, roles, defined concepts and properties, classes and individuals
  - Ladder up to provide abstractions as needed
    - And fill in siblings
  - Propose a normalised ontology
    - Classify it to see that it works correctly
      - Provide probe classes to check both classification and unsatisfiability
        - One file to turn in
  - Download the tangled ontology proposed by the personnel department
    - Untangle it
      - A second file to turn in