Foundations of the Semantic Web: Ontology Engineering

Building Ontologies 3+
Common problems
Ontology Patterns
Re-representing properties and classes
Parts and Wholes

Alan Rector & colleagues
Special acknowledgement to Jeremy Rogers & Chris Wroe

"Elephant Traps"

- 'Some' does not imply only
  'Only' does not imply some'
- Trivial satisfaction of universal restrictions
- Domain and Range Constraints
- What to do when it all turns red

someValuesFrom means "some"

- someValuesFrom means "some" means "at least 1"
- Dog_owner *complete*
  Person *and* hasPet *someValuesFrom* Dog
  - means:
    A Pet_owner is any person who has as a pet some (i.e. at least 1) dog
- Dog_owner partial
  Person *and* hasPet *someValuesFrom* Dog
  - means
    All Pet_owners are people and have as a pet some (i.e. at least 1) dog.

allValuesFrom means "only"

- allValuesFrom means "only" means "no values except"
- First_class_lounge *complete*
  Lounge *and* hasOccupants *allValuesFrom* FirstClassPassengers
  - Means
    "A 'first class lounge' is any lounge where the occupants are *only* first class passengers"
    or
    "A first 'class lounge' is any lounge where there are *no* occupants *except* first class passengers"
- First_class_lounge *partial*
  Lounge *and* hasOccupants *allValuesFrom* FirstClassPassengers
  - Means
    "All first class lounges have *only* occupants who are first class passengers"
    "All first class lounges have *no* occupants *except* first class passengers"
    "All first class lounges have *no* occupants who are *not* first class passengers"

“Some” does not mean “only”

- A “dog owner” might also own cats, and rats, and guinea pigs, and…
  - It is an open world, if we want a closed world we must add a closure restriction or axiom
- Dog_only_owner *complete*
  Person *and* hasPet *someValuesFrom* Dog and
  hasPet *allValuesFrom* Dog

- A “closure restriction” or “closure axiom”
  - The problem in making maguerita pizza a vegie pizza
  - Closure axioms use ‘or’ (disjunction)
  - dog_and_cat_only_owner *complete*
    hasPet *someValuesFrom* Dog and
    hasPet *someValuesFrom* Cat and
    hasPet *allValuesFrom* (Dog or Cat)

“Only” does not mean “some”

- There might be *nobody* in the first class lounge
  - That would still satisfy the definition
  - It would not violate the rules
- A pizza with *no* toppings satisfies the definition of a vegetarian pizza
  - Pizza & has_topping_ingredient *allValuesFrom* Vegetarian_topping
    - It has no toppings which are meat
      - It has not toppings which are not vegetables
        - » It has no toppings which aren’t fish…
  - Analogous to the empty set is a subset of all sets
    - One reason for a surprising subsumption is that you have made it impossible for there to be any toppings
      - *allValuesFrom* (cheese and tomato)

Trivial Satisfiability

- A universal (‘only’) restriction with an unsatisfiable filler is “trivially satisfiable”
  - i.e. it can be satisfied by the case where there is no filler
    - If there is an existential or min-cardinality restriction, inferred or explicit, then the class will be unsatisfiable
      - Can cause surprising ‘late’ bugs

Domain & Range Constraints

- Actually axioms
  - *Property P range(RangeClass*)
    means
    - owl:Thing
      restriction(P *allValuesFrom* RangeClass)
  - *Property P domain(DomainClass)*
    means
    - owl:Thing
      restriction(inverse(P) *allValuesFrom* DomainClass)

What happens if violated

- Actually axioms
  - *Property eats range(LivingThing*)
    means
    - owl:Thing
      restriction(P *allValuesFrom* LivingThing)
  - Bird eats some Rock
    - All StoneEater eats some rocks
      - What does this imply about rocks?
        - » *Some* rocks are living things
        - » because only living things can be eaten
        - » What does this say about “all rocks”?

Domain & Range Constraints

- Actually axioms
  - *Property eats domain(LivingThing)*
    means
    - owl:Thing
      restriction(inverse(eats) *allValuesFrom* LivingThing)
    - “Only living things eat anything”
  - StoneEater eats some Stone
    - All StoneEaters eat some Stone
      - Therefore All StoneEaters are living things
        - » If StoneEaters are not already classified as living things, the classifier will reclassify (‘coerce’) them
        - » If StoneEaters is disjoint from LivingThing it will be found disjoint

Example of Coercion by Domain violation

- has_topping: *domain*(Pizza) *range*(Pizza_topping)

  class Ice_cream_cone
  has_topping *some* Ice_cream

- If Ice_cream_cone and Pizza are *not* disjoint:
  - Ice_cream_cone is classified as a kind of Pizza
    …but: Ice_cream is *not* classified as a kind of Pizza_topping
    - Have shown that:
      all Ice_cream_cones are a kinds of Pizzas,
      but only that:
      some Ice_cream is a kind of Pizza_topping
      - » *Only domain constraints can cause reclassification*
        … by now most people are very confused - need lots of examples & back to basics

Reminder
Subsumption means necessary implication

- *“B is a kind of A”*
  means
  “All Bs are As”
  - *“Ice_cream_cone is a kind of Pizza”*
    means
    “All ice_cream_cones are pizzas”
    - From “Some Bs are As” we can deduce very little of interest in DL terms
      - » “some ice_creams are pizza_toppings”
        says nothing about “all ice creams”

Summary:Domain & Range Constraints Non-Obvious Consequences

- Range constraint violations – *unsatisfiable or ignored*
  - If filler and RangeClass are disjoint: *unsatisfiable*
  - *Otherwise nothing happens!*
- Domain constraint violations – *unsatisfiable or coerced*
  - If subject and DomainClass are disjoint: *unsatisfiable*
  - *Otherwise, subject reclassified (coerced) to kind of DomainClass!*
- *Furthermore cannot be fully checked before classification*
  - *although tools can issue warnings.*

What to do when "Its all turned red"

Don't Panic!

- Unsatisfiability propagates – so trace it to its source
  - Any class with an unsatisfiable filler in a *someValuesFor* (existential) restriction is unsatisfiable
  - Any *subclass* of an unsatisfiable class is unsatisfiable
  - Therefore errors propagate, trace them back to their source
- Only a few possible sources
  - Violation of disjoint axioms
  - Unsatisfiable expressions in some restrictions
    - Confusion of "and" and "or"
  - Violation of a universal (allValuesFrom) constraint (including range and domain constraints)
    - Unsatisfiable domain or range constraints
- Tools coming RSN

Part IV – Patterns: n-ary relations

- 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

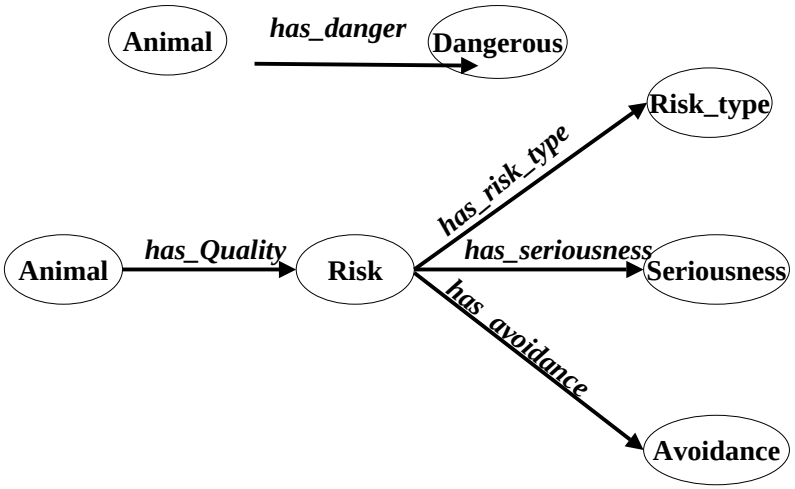
Saying something about a restriction

- Not just
  - that an animal is dangerous,
  - but why
  - And how dangerous
  - And how to avoid
- But can say nothing about properties
  - except special thing
    - Super and subproperties
    - Functional, transitive, symmetric

Re-representing properties as classes

- To say something about a property it must be re-represented as a class
  - property:has_danger → Class: Risk
    - plus property: Thing has_quality Risk
    - plus properties: Risk has_reason
      has_risk_type
      has_avoidance_measure
  - Sometimes called "reification"
    - But "reification" is used differently in different communities

Re-representing the property has_danger as the class Risk



Lions are dangerous

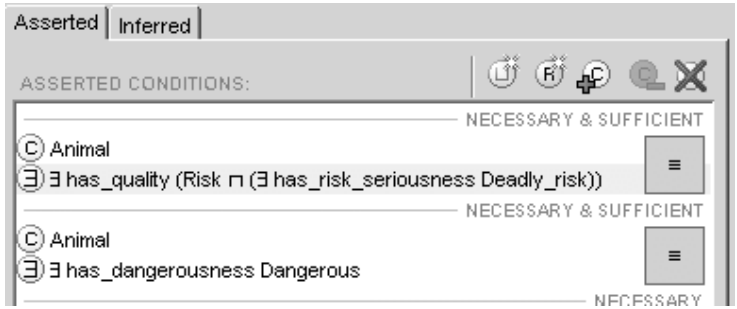
- All lions pose a deadly risk of physical attack that can be avoided by physical separation
- All lions have the quality risk that is
  - of type some physical attack
  - of seriousness some deadly
  - has avoidance means some physical separation

Can add a second definition of Dangerous Animal

- A dangerous animal is any animal that has the quality Risk that is Deadly
  - or
- Dangerous_animal =
  - Animal
    has_quality *some*
    (Risk AND has_seriousness *some* Deadly)
  - [NB: “that” paraphrases as “AND”]

In the tool

- Dangerous_animal =
  - Animal
    has_quality *some*
    (Risk AND has_seriousness *some* Deadly)



This says that

- Any animal that is Dangerous

  is also

  An animal that has the quality Risk
  with the seriousness Deadly

Anopheles Mosquitos now count as dangerous

- Because they have a deadly risk of carrying disease

Multiple definitions are dangerous

- Better to use one way or the other
  - Otherwise keeping the two ways consistent is difficult
  - … but ontologies often evolve so that simple Properties are re-represented as Qualities
    - Then throw away the simple property

Often have to re-analyse

- What do we mean by "Dangerous"
  - How serious the danger?
  - How probable the danger?
  - Whether from individuals (Lions) or the presence or many (Mosquitos)?
- Moves to serious questions of "ontology"
  - The information we really want to convey
    - Often a sign that we have gone to far
      - So we will stop

Parts and Wholes: The Basics

Part VI – Patterns: Part-whole relations

- 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
- *Part-whole relations*

Part-whole relations

One method: NOT a SWBP draft

- How to represent part-whole relations in OWL is a commonly asked question
- SWBP will put out a draft.
- This is one approach that will be proposed
  - It has been used in teaching
  - It has no official standing
  - It is presented for information only

Part Whole relations

- OWL has no special constructs
  - But provides the building blocks
- Transitive relations
  - Finger is_part_of Hand
    Hand is_part_of Arm
    Arm is_part_of Body
    - →
  - Finger is_part_of Body

Implementation Pattern
Transitive properties with non-transitive "direct" subproperties

- Transitive properties should have non-transitive children
  - isPartOf : transitive
    isPartOfDirectly : non-transitive
- Split which is used in partial descriptions and complete definitions
  - Necessary conditions use non-transitive version
  - Definitions use transitive version
- Benefits
  - Allows more restrictions in domain/range constraints and cardinality
    - Allows the hierarchy along that axis to be traced one step at a time
    - Allow a good approximation of pure trees
      - Make the nontransitive subproperty functional
        - » Transitive properties can (almost) never be functional (by definition, a transitive property has more than one value in any non-trivial system)
    - Constraints on transitive properties easily lead to unsatisfiability

Many kinds of part-whole relations

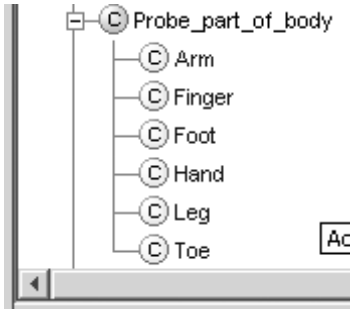
- Physical parts
  - hand-arm
- Geographic regions
  - Hiroshima - Japan
- Functional parts
  - cpu – computer
- See Winston & Odell
  Artale
  Rosse

Simple version

- One property *is_part_of*
  - transitive
    - Finger is_part_of *some* Hand
      Hand is_part_of *some* Arm
      Arm is_part_of some Body

Get a simple list

- Probe_part_of_body =
  Domain_category
  is_part_of some Body



- Logically correct
  - But may not be what we want to see

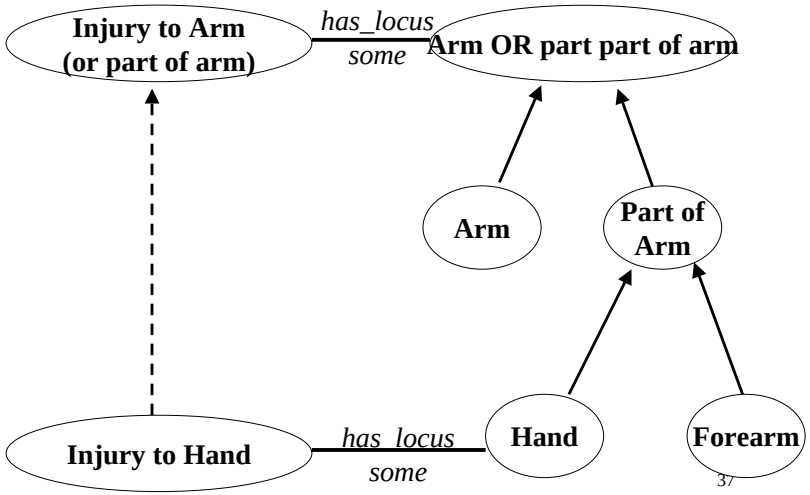
Injuries, Faults, Diseases, Etc.

- A hand is not a *kind of* a body
  - … but an injury to a hand is a kind of injury to a body
- A motor is not a *kind of* automobile
  - … but a fault in the motor is a kind of fault in the automobile
- And people often expect to see partonomy hierarchies

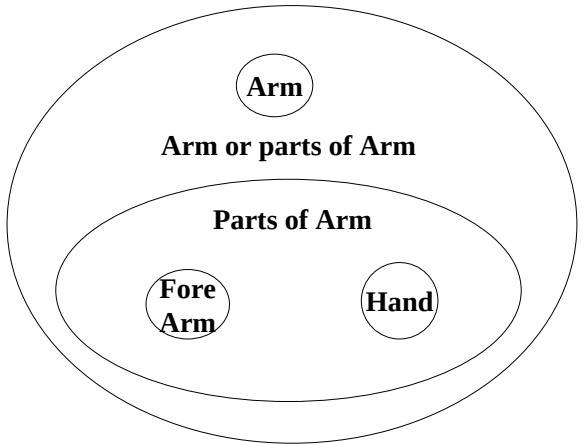
Being more precise: “Adapted SEP Triples”

- Body (‘as a whole’)
  - Body
- The Body’s parts
  - is_part_of *some* Body
- The Body and it’s parts
  - Body OR is_part_of *some* Body
- Repeat for all parts
  - Use ‘Clone class’ or
  - NB: ‘JOT’ Python plugin is good for this

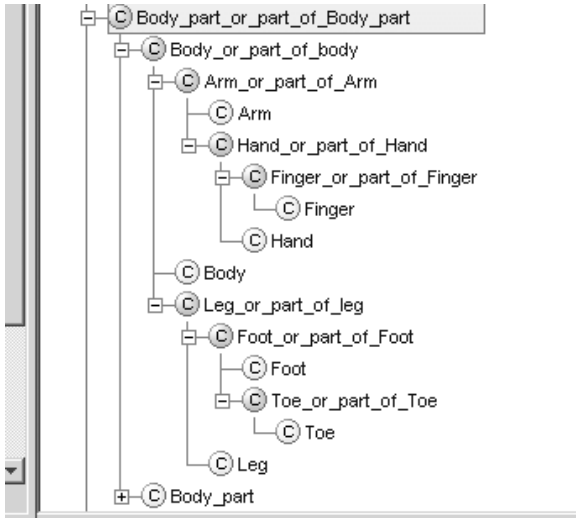
Adapted SEP triples: UML like view



Adapted SEP triples: Venn style view



Resulting classification: Ugly to look at, but correct



Using part-whole relations: Defining injuries or faults

- Injury_to_Hand =
  Injury has_locus *some* Hand_or_part_of_hand
- Injury_to_Arm =
  Injury has_locus *some* Arm_or_part_of_Arm
- Injury_to_Body =
  Injury has_locus *some* Body_or_part_of_Body

- The expected hierarchy from point of view of anatomy

Parts & Wholes in More Detail

Parts & Wholes, containment, connection and adjacency – common sense merology

- Standard lexical semantic versions motivated by history
  Many philosophical versions motivated by topology
  - This version motivated primarily by anatomy and engineering
- Classic knowledge representation work is
  - Odell, J. J. (1994). "Six different kinds of composition." Journal of Object Oriented Programming **5**(8): 10-15.
    - A short readable summary
      - Not complete nor completely up to date
  - Winston, M., R. Chaffin, et al. (1987). "A taxonomy of part-whole relations." Cognitive Science **11**: 417-444.
- Merology – the study of parts and wholes
  - A quick glance at Google…

Parts & wholes: Some examples

- The leg is part of the chair
- The left side of the body is part of the body
- The liver cells are part of the liver
- The ignition of part of the electrical system of the car
- The goose is part of the flock
- Manchester is part of England
- Computer science is part of the University

Five families of relations

- Partonomic
  - Parts and wholes
    - The lid is part of the box
  - Constitution
    - The box is made of cardboard
  - Membership
    - The box is part of the shipment
- Nonpartonomic
  - Containment
    - The gift is contained in the box
  - Connection/branching/Adjacency
    - The box is connected to the container by a strap

Some tests

- True kinds of *part-of* are transitive and
  A fault to the part is a fault in the whole
  - The finger nail is part of the finger is part of the hand is part of the upper extremity is part of the body
    - Injury to the fingernail is injury to the body
  - The tail-light is part of the electrical system is part of the car
    - A fault in the tail light is a fault in the car
- Some similar relations are not transitive
  - The foot of the goose is part of the goose but not part of the flock of geese
    - Damage to the foot of the goose is not damage ot the flock of geese
- *Containment* is transitive but things contained are not necessarily parts
  - A fault (e.g. souring) to the milk *contained in* the bottle is not damage to the bottle
- Some kinds of part-whole relation are questionably transitive
  - Is the cell that is part of the finger a part of the body?
    - Is damage to the cell that is part of the finger damage to the body?
      - Not necessarily, since the cells in my body die and regrow constantly

Structural parts

- The leg is *a component of* of the table
  - Discrete
  - connected,
  - clear boundary,
  - specifically named
  - may be differently constituted
  - Can have metal legs on a wooden table or vice versa
- The left side is *a subdivision of* the table
  - ‘Side’, ‘Lobe’, ‘segment’, ‘region’,…
    - Arbitrary, similarly constituted,
    - components typically fall into one or another subdivision;
    - defined in relation to something else;
    - sensible to talk about what fraction it is: half the table, a third of the table, etc.

Propagates_via / transitive_across

- Components of subdivisions are components of the whole, *but*
  subdivisions of components are not subdivisions of the whole
  - A the left side of the steering wheel of the car is not a subdivision of the car
    - and certainly not a subdivision of the left side of the car
      - (at least not in the UK)
- No consistent name for this relation between properties
  - We shall call it propagates_via or transitive_across
    - Also known as “right identities”
  - Not supported in most DLs or OWL directly
    - Although an extension to FaCT to support it exists
    - Heavily used in medical ontologies (GRAIL and SNOMED-CT)

No simple solution: Here’s one of several nasty kluges

- Component_of_table is defined as a component of table or any subdivision of table
  - Must do it for each concept
    - A Schema rather than an axiom
      - No way to say “same as”
      - No variables in OWL
        - » or most DLs
- SCHEMA:
  Components_of_X ≙
  isComponentOf someValuesFrom
  (X or (someValuesfrom isSubDivisionOf X))
  - Tedious to do with OilEd Expression editor
    - Schemas to be built into new tools

Functional parts

- Structural parts form a contiguous whole
  - May or may not contribute to function
    e.g. decorative parts, vestiges such as the human appendix, "spandrels"[1], accidental lumps and bumps
- The remote control is part of the projection system
  - May or may not be physically connected to it
    - Part of a common function
- Biology examples:
  - The endocrine system
    - The glands are not connected, but form part of a functioning system communicating via hormones and transmitters
    - The blood-forming system
      - Bone marrow in various places, the spleen, etc.

[1] See Stephen J Gould

If something is both a structural and functional part…

- Must put in both restrictions explicitly
  - Can create a common child property but this gets complicated with the different kinds of structural parts
  - Better to put syntactic sugar in tools
    - But syntactic sugar has not arrived, so for this course you have to do it by hand!
      - Coming Real Soon Now (RSN)

So far we have

- isPartOf
  isStructuralPartOf
  isSubdivisionOf
  isComponentOf
  isFunctionalPartOf
- Many other varieties
  - Layers, surfaces, …
- Many other constraints, e.g.
  - Dimensions must match
    - 3-D things can only be structural parts of 3-D things
  - boundaries have one less dimension than the things they bound
    - surfaces bound volumes, lines bound areas
  - layers of subdivisions are subdivisions of layers of the whole
    - the skin of the finger is a subdivision of the skin of the upper hand
- Can add isSubprocessOf –
  - similar to isComponentOf

What about containment

- X **isContainedIn** Y **isStructuralPartOf** Z →
  X **isContainedIn** Z
- Rigorous version needs analogous schema to subdivision
  - **contained_in**_X ≅
    contained_in *someValuesFor*
    (X *or* (*someValuesFor* **is_structural_part_of** X))
- Weak approximation
  - make contained_in a parent of is_structural_part
    - Not right – implies all structural parts are contained in the whole
      » A "kluge"