ISIT315
Week 11

This lecture will discuss eight steps of ontology engineering and further discussion on description logic

Format of final exam

- 2 hours
- Given 4 questions (each question is worth 15 marks)
  - Answer all four questions
  - Each question has sub-parts (a), (b), (c) or (d)
- Covers all topics discussed in this subject

8 steps of ontology development process

1. Determine scope
2. Consider reuse
3. Enumerate terms
4. Define taxonomy
5. Define properties
6. Define facets
7. Define instances
8. Check for anomalies

Step 1. Determine scope

- Ontology is a model/abstraction of a domain
- Should be determined by the use of which the ontology will be put and by future extensions
- Basic questions to ask
  - What is the domain that the ontology will cover
  - For what we are going to use the ontology
  - For what types of questions should the ontology provide answers
  - Who will use and maintain the ontology

Competency questions

- One of the easy ways to determine scope of ontology is to develop competency questions
- Example: Wine and food domain
  - Which wine characteristics should I consider when choosing a wine?
  - What is the best choice of wine for grilled meat?
  - Which characteristics of a wine affect its appropriateness for a dish?
  - Does Cabernet Sauvignon go well with seafood?

Example: Food and wines (1)

- We plan to use this ontology for the applications that suggest good combinations of wines and food.
  - Naturally, the concepts describing different types of wines, main food types, the notion of a good combination of wine and food and a bad combination will figure into our ontology.
- It is unlikely that the ontology will include concepts for managing inventory in a winery or employees in a restaurant even though these concepts are somewhat related to the notions of wine and food.

Example: Food and wines (2)

- If the ontology will be used to help restaurant customers decide which wine to order, we need to include retail-pricing information.
- If it is used for wine buyers in stocking a wine cellar, wholesale pricing and availability may be necessary.

Step 2. Consider reuse

- Rarely we have to start from scratch when defining ontology
  - Reusing existing ontologies may be a requirement if our system needs to interact with other applications that have already committed to particular ontologies or controlled vocabularies.
- Look for third-party that provides a useful starting point for ontology

Example: Food and wines

- A knowledge base of French wines may already exist
  - If we can import this knowledge base and the ontology on which it is based, we will have not only the classification of French wines but also the first pass at the classification of wine characteristics used t o distinguish and describe the wines.
- Lists of wine properties may already be available from commercial Web sites such as *www.wines.com* that customers consider use to buy wines.

Step 3. Enumerate terms

- Useful first step is to write down in an unstructured list all relevant terms that are expected to appear in the ontology
  - nouns - class names
  - verbs/phrases – property names (e.g. hasComponent, isPartOf)

Example: Food and wines

- Wine-related terms will include
  - Wine, grape, winery, location, a wine’s color, body, flavor and sugar content;
  - Different types of food, such as fish and red meat;
  - Subtypes of wine such as white wine, and so on.

Step 4. Define taxonomy

- Organise terms identified in step 3 into taxonomic hierarchy
  - Can use bottom-up or top-down approach
  - Hierarchy – class/subclass
- Three approaches
  - Top-down
  - Bottom-up
  - Combination

Top-down

- Starts with the definition of the most general concepts in the domain and subsequent specialization of the concepts.
- Example
  - We can start with creating classes for the general concepts of Wine and Food.
  - Then we specialize the Wine class by creating some of its subclasses: White wine*, Red* wine*, Rosé wine.*
  - *We can further categorize the Red wine class, for example,* into Shiraz, Red Burgundy, Cabernet Sauvignon, and so on.

Bottom-up

- Starts with the definition of the most specific classes, the leaves of the hierarchy, with subsequent grouping of these classes into more general concepts.
- Example
  - We start by defining classes for Pauillac and Margaux wines.
  - We then create a common superclass for these two classes—Medoc—which in turn is a subclass of Bordeaux.

Combination

- Is a combination of the top-down and bottom-up approaches
  - We define the more salient concepts first and then generalize and specialize them appropriately
    - may result in *middle-level* concept

Which way?

- depends on personal view of the domain

Step 5. Define properties

- Often interleaved with step 4
- Organise properties that link the classes while organising these classes in a hierarchy
- Provide domain and range to properties
  - need to balance between generality and specificity
    - e.g. broad/general enable properties to be used by subclasses (through inheritance)
    - too narrow/specific → may create inconsistencies through domain and range violations

Different types of object properties

- Intrinsic property
- Extrinsic property
- Parts (can be both physical and abstract parts)
- Relationships to other individuals
- Example:
  - Intrinsic – *flavour* of wine
  - Extrinsic – wine's *name*, *area* where it comes from
  - Parts – *courses of a meal*
  - Relationship*: maker* of the wine

Step 6. Define facets

- Consider
  - Cardinality
    - Minimum and maximum cardinalities
    - E.G. Allowed to have a certain number of different values (e.G. At least one pizzatopping; at most one pizzabase)
  - Value types: string, number, boolean, enumerated
  - Required values
  - Relational characteristics
    - Symmetry, transitivity, inverse properties, functional values

Step 7. Define instances

- Defining an individual instance of a class requires
  - (1) Choosing a class,
  - (2) Creating an individual instance of that class, and
  - (3) Filling in the values.
- Fill in instances
  - – can retrieve from databases
  - – often is not done manually due to the large number of instances

Step 8. Check for anomalies

- Check for consistencies
- Examples of anomalies
  - Incompatible domain and range definitions for transitive, symmetric or inverse properties
  - Inconsistent cardinalities
  - Property values conflict with domain and restriction

References for this topic

- Antoniou, G. and Van Harmelen, F. *A Semantic Web Primer*, MIT Press
  - Chapter 7
- Natalya F. Noy and Deborah L. McGuinness. ``Ontology Development 101: A Guide to Creating Your First Ontology''. March 2001 (http://www-ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness-abstract.html)

Reasoning and Logic

What is Reasoning?

- **Reasoning** is the cognitive process of looking for reasons for beliefs, conclusions, actions or feelings
- The world does not give us complete information
- Reasoning is the set of processes that enables us to go beyond the information given
- Reasoning in most of the cases is based on Logic

Three species of OWL

- OWL Lite
  - Classification hierarchies with simple constraints
  - Reasoning is computational simple & efficient
- OWL DL
  - Computational complete and decidable (computation in finite time)
  - Corresponds to description logic
- OWL Full
  - Maximum expressiveness
  - No computational guarantee

The “DL” in OWL DL

- Description Logics
- Goal: to be able to reason (i.e. infer information) about a knowledge base
  - A knowledge base consists of both meta information (classes) and instance information (individuals)

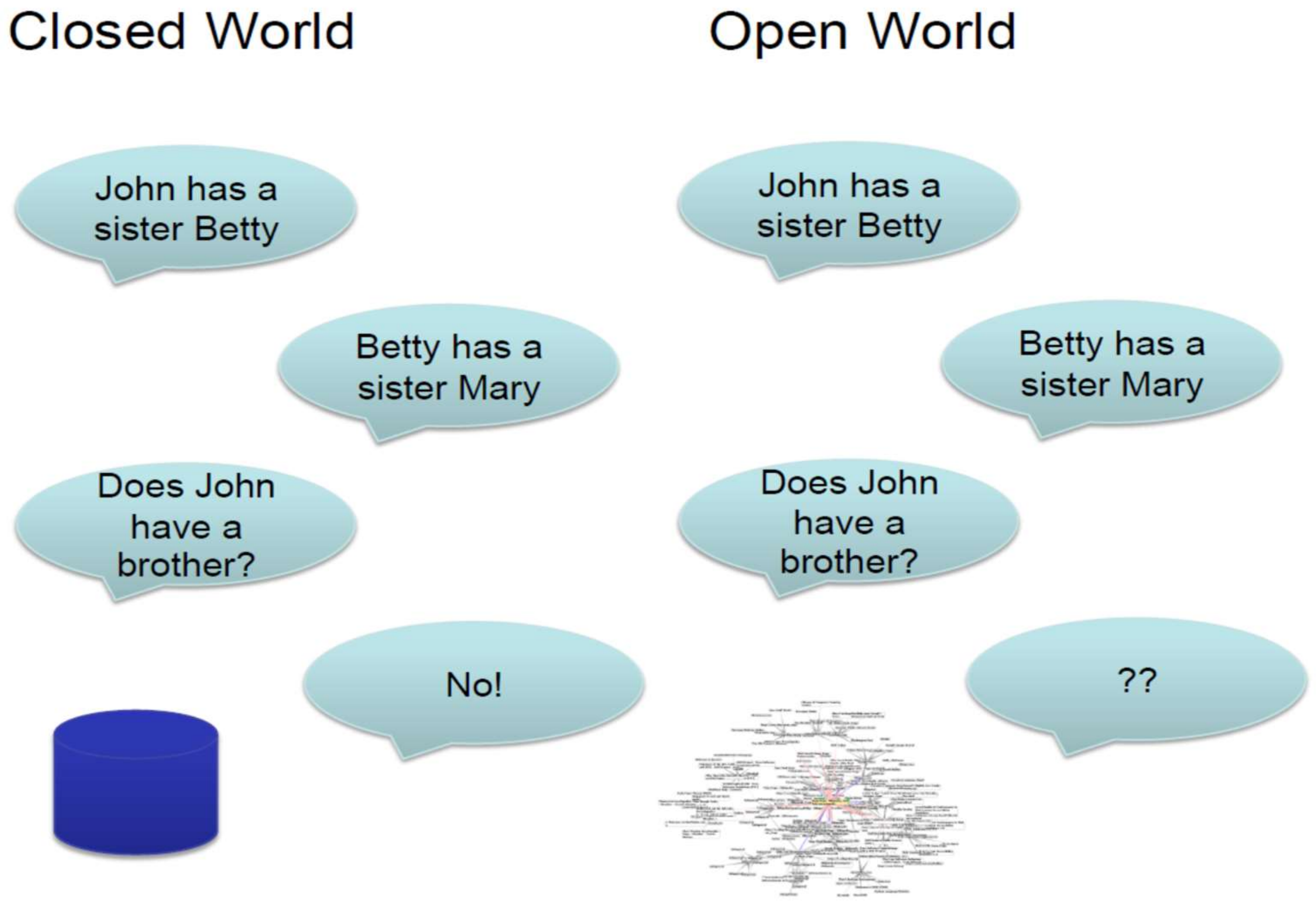
Open world assumption

- Consider the following example:

```
Statement: "Mary" "is a citizen of" "France"
Question: Is Paul a citizen of France?
```

- "Closed world" (for example SQL) answer: No.
- "Open world" answer: Unknown.

Closed world vs. Open World



Description Logics

- Highly expressible fragment of first-order logic with
  - Decidability
    - Guaranteed that computation can be done in finite time
  - Completeness
    - Every question within the logical system can be answered

Description Logic

- Designed for logical representation of object-oriented formalisms
  - Frames/classes/concepts
    - Sets of objects
  - Roles/properties
    - Binary relations on objects
  - Individuals
- Represented as a collection of statements, with unary & binary predicates that stand for concepts and roles, from which deductions can be made

Description Logic Syntax - Concepts

- Classes/concepts are actually a set of individuals
- We can distinguish different types of concepts:
  - Atomic concepts: Cannot be further decomposed (i.e. Person)
  - Incomplete concepts (defined by $\sqsubseteq$)
    - necessary condition
  - Complete concepts (defined by $\equiv$)
    - necessary and sufficient condition

Incomplete concept defintion

- Man $\sqsubseteq$ Person $\sqcap$ Male
  - Intended meaning: If an individual is a man, we can conclude that it is a person and male.
- Man(x) $\Rightarrow$ Person(x) $\land$ Male(x)
- In Protege
  - necessary conditions are called primitive class

Complete concept definition

- $\text{Man} \equiv \text{Person} \sqcap \text{Male}$
  - Intended meaning: Every individual which is a male person is a man, **and** every man is a male person.
- $\text{Man}(x) \Leftrightarrow \text{Person}(x) \wedge \text{Male}(x)$
- In Protégé
  - necessary and sufficient conditions are called Equivalent classes (defined class)

Description Logic Syntax- Roles

- Roles relate individuals to each other
  - I.e. directedBy(Pool Sharks, Edwin Middleton), hasChild(Jonny, Sue)
- Roles have a **domain** and a **range**
- Example:
  - Domain(directedBy, Movie)
  - Range(directedBy, Person)
    - Given the above definitions we can conclude that Pool Sharks is a movie and that Edwin Middleton is a person.

Description Logic Syntax- Roles

- Functional Roles
  - Roles which have exactly one value
  - Usually used with primitive data values
- Transitive Roles
  - Example: hasAncestor

    Simple in a rule language:

hasAncestor(X,Z) $\Rightarrow$ hasAncestor(X,Y), hasAncestor(Y,Z).

Description Logic Syntax- Roles

- Symmetric Roles
  - Roles which hold in both directions
  - I.e. hasSpouse, hasSibling

- Inverse Roles
  - Roles are directed, but each role can have an inverse
  - I.e. hasParent $\equiv^{-}$ hasChild

    hasParent(X,Y) $\Leftrightarrow$ hasChild(Y,X)

Description Logic Knowledge Bases

- Typically a DL knowledge base (KB) consists of two components
  - Tbox (terminology/terminological knowledge): A set of inclusion/equivalence axioms denoting the conceptual schema/vocabulary of a domain
    - Bear $\sqsubseteq$ Animal $\sqcap$ Large
    - transitive(hasAncestor)
  - Abox (assertions/assertional knowledge): concept assertions and role assertions
    - Bear(WinniPooh)
    - hasAncestor(Susan, Granny)

Example TBox

- man $\sqsubseteq$ person
- woman $\sqsubseteq$ person
- city $\sqsubseteq$ location

Example: ABox

- person(mary)
- person(john)
- loves(mary, john)
- loves(john, mary)
- works_for(mary, c1)
- located_in(NY, c1)
- woman(mary)
- man(john)

DL and reasoning

- Modern DL systems provide their users with reasoning services that can automatically deduce implicit knowledge from the explicitly represented knowledge, and always yield a correct answer in finite time.

The Subsumption Algorithm

- The subsumption algorithm determines subconcept-superconcept relationships:
- C is subsumed by D if all instances of C are necessarily instances of D
  – the first description is always interpreted as a subset of the second description.

The Instance Algorithm

- The instance algorithm determines instance relationships:
  - the individual *i* is an instance of the concept description C if *i* is always interpreted as an element of the interpretation of C.

The Consistency Algorithm

- The consistency algorithm determines whether a knowledge base (consisting of a set of assertions and a set of terminological axioms) is non-contradictory

Reasoning Steps - 1

- In a typical application, one would start building the Tbox
- Ensure that all concepts in it are satisfiable,
  - i.e., Are not subsumed by the bottom concept, which is always interpreted as the empty set.
- One would use the subsumption algorithm to compute the subsumption hierarchy
  - i.e., to check, for each pair of concept names, whether one is subsumed by the other.

Reasoning Steps - 2

- Check for its consistency Abox with Tbox
  - Compute the most specific concept(s) that each individual is an instance of
  - Could also use a concept description as a query, i.e., We could ask the DL system to identify all those individuals that are instances of the given, possibly complex, concept description.

OWL Reasoning

- In the context of OWL ontologies, reasoning is the process of making explicit that which is implicit in an ontology
- Consider the following example

```
<owl:Class rdf:ID="FemalePerson"/>
  <owl:Class rdf:ID="MalePerson"/>
  <owl:ObjectProperty rdf:ID="hasWife">
    <rdfs:subPropertyOf>
      <owl:ObjectProperty rdf:ID="hasSpouse"/>
    </rdfs:subPropertyOf>
    <owl:inverseOf>
      <owl:FunctionalProperty rdf:about="#hasHusband"/>
    </owl:inverseOf>
    <rdfs:range rdf:resource="#FemalePerson"/>
    <rdfs:domain rdf:resource="#MalePerson"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  </owl:ObjectProperty>
  <owl:FunctionalProperty rdf:ID="hasHusband">
    <rdfs:subPropertyOf rdf:resource="#hasSpouse"/>
    <rdfs:domain rdf:resource="#FemalePerson"/>
    <rdfs:range rdf:resource="#MalePerson"/>
    <owl:inverseOf rdf:resource="#hasWife"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </owl:FunctionalProperty>
  <FemalePerson rdf:ID="Martha">
    <hasHusband>
      <MalePerson rdf:ID="George"/>
    </hasHusband>
  </FemalePerson>
</rdf:RDF>
```

Example 1: Questions can be answered directly by the assertions in the ontology

- **Question**: Are there any instances of MalePerson?
- **Answer**: George.

```
<owl:Class rdf:ID="FemalePerson"/>
  <owl:Class rdf:ID="MalePerson"/>
  <owl:ObjectProperty rdf:ID="hasWife">
    <rdfs:subPropertyOf>
      <owl:ObjectProperty rdf:ID="hasSpouse"/>
    </rdfs:subPropertyOf>
    <owl:inverseOf>
      <owl:FunctionalProperty rdf:about="#hasHusband"/>
    </owl:inverseOf>
    <rdfs:range rdf:resource="#FemalePerson"/>
    <rdfs:domain rdf:resource="#MalePerson"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  </owl:ObjectProperty>
  <owl:FunctionalProperty rdf:ID="hasHusband">
    <rdfs:subPropertyOf rdf:resource="#hasSpouse"/>
    <rdfs:domain rdf:resource="#FemalePerson"/>
    <rdfs:range rdf:resource="#MalePerson"/>
    <owl:inverseOf rdf:resource="#hasWife"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </owl:FunctionalProperty>
  <FemalePerson rdf:ID="Martha">
    <hasHusband>
      <MalePerson rdf:ID="George"/>
    </hasHusband>
  </FemalePerson>
</rdf:RDF>
```

Example 2: Use "reasoning" to discover the initially implicit answer in the ontology

- **Question**: Who is George's wife?
- **Answer**: Martha.

- *the owl:inverseOf relation between hasHusband and hasWife was exploited to deduce that George hasWife Martha.*

```
<owl:Class rdf:ID="FemalePerson"/>
   <owl:Class rdf:ID="MalePerson"/>
   <owl:ObjectProperty rdf:ID="hasWife">
      <rdfs:subPropertyOf>
         <owl:ObjectProperty rdf:ID="hasSpouse"/>
      </rdfs:subPropertyOf>
      <owl:inverseOf>
         <owl:FunctionalProperty rdf:about="#hasHusband"/>
      </owl:inverseOf>
      <rdfs:range rdf:resource="#FemalePerson"/>
      <rdfs:domain rdf:resource="#MalePerson"/>
      <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
   </owl:ObjectProperty>
   <owl:FunctionalProperty rdf:ID="hasHusband">
      <rdfs:subPropertyOf rdf:resource="#hasSpouse"/>
      <rdfs:domain rdf:resource="#FemalePerson"/>
      <rdfs:range rdf:resource="#MalePerson"/>
      <owl:inverseOf rdf:resource="#hasWife"/>
      <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
   </owl:FunctionalProperty>
   <FemalePerson rdf:ID="Martha">
      <hasHusband>
         <MalePerson rdf:ID="George"/>
      </hasHusband>
   </FemalePerson>
</rdf:RDF>
```

Example 3: Use "reasoning" to discover the initially implicit answer in the ontology

- **Question**: Who has a spouse?
- **Answer**: Martha has spouse George and George has spouse Martha.
- *the owl:inverseOf relation between hasHusband and hasWife was combined with the rdfs:subPropertyOf relation of hasHusband and hasWife with hasSpouse to deduce that George and Martha have each other as spouse*

Some examples of reasoners in OWL

- FACT++
  - http://owl.cs.manchester.ac.uk/tools/fact/
- HermiT
  - http://www.hermit-reasoner.com/RacerPro
- Pellet
  - https://www.w3.org/2001/sw/wiki/Pellet