Ontology Development and Engineering

Manolis Koubarakis
Knowledge Technologies

Outline

- Ontology development and engineering
- Key modelling ideas of OWL 2
- Steps in developing an ontology
- Creating an ontology with Protégé OWL – useful ontology design patterns

Ontology Engineering

- Ontology engineering is **knowledge engineering**.
- Developing ontology engineering techniques, methodologies and tool support is a **core research problem** and there are today various interesting ontology engineering methodologies (accompanied by relevant tools).
- We will present the **ontology development** methodology championed by the Protégé and CO-ODE groups at Stanford and Manchester.

Goals of this Presentation

- To outline one possible way (a **recipe**) to construct an OWL 2 ontology.
- To emphasize ontology design patterns i.e., known solutions to recurrent modeling problems that have been tested in different applications and are now well documented.
- To demonstrate how to use the **Protégé OWL** to implement these ontology design patterns.

When to Use OWL?

- We need to consider carefully the following features of OWL (and any other DL-based language) to decide whether OWL is the right language for building an ontology for a domain at hand:
  - **Object-centered** (based on individuals with unique identity, classes and properties).
  - **Terminological**: Supports the building of complex terms (noun phrases) in the form of classes. Individuals are asserted to belong to these classes. There is no way to express complex quantifications or disjunctions (as in FOL).
  - **Deductive**: not just a passive repository of assertions.
  - **Incremental**: partial, incomplete descriptions of individuals are acceptable and can be refined later.
  - Based on self-organization of concepts in a **subsumption** hierarchy.
  - Based on **open world assumption**.

Key Modeling Ideas of OWL (and related languages based on DLs)

- OWL 2 allows us to represent knowledge about a domain using the following constructs:
  - **Entities**
    - **Classes**
    - **Individuals**
    - **Properties (object properties and data properties)**
  - **Property restrictions**
  - **Class expressions**
  - **Data ranges**
    - **Data types**
  - **Axioms**
    - **Class axioms**
    - **Property axioms**
    - **Assertions**
  - **Annotations**
  - **Importing of other ontologies**

Classes

- In OWL (as in DLs), we can distinguish two kinds of classes:
  - **Defined classes**
  - **Primitive classes**

Defined Classes

- A **defined class** is like an “if and only if” statement in logic.
- Example: A driver can be defined to be exactly a person who drives a vehicle.
- With a defined class, we give necessary and sufficient conditions for membership in a class.
- Thus a defined class allows **deduction in two directions**. For example:
  - If someone is a driver, then he/she is a person and he/she drives a vehicle.
  - If someone is a person and he/she drives a vehicle, then he/she is a driver.

Defined Classes (cont'd)

- Defined classes in OWL 2 are introduced as follows:
  - In the functional-style syntax, using an equivalent classes axiom:
    `EquivalentClasses(CE1 ... CEn)`
  - Similarly in other syntaxes.

- Example:

```
EquivalentClasses(a:Driver
   ObjectIntersectionOf(
        ObjectSomeValuesFrom(a:drives a:Vehicle)
        a:Person))
```

Primitive Classes

- A **primitive class** includes only necessary (but not sufficient conditions) for membership.

- Example: It is hard to define a dog (or any other natural kind). However, we might want to say:
  - Among other things, a dog is something that eats bones.

- In contrast to defined classes, primitive classes support deductions in only one direction. For example:
  - If something is a dog, then we can infer that it eats bones.

Primitive Classes (cont'd)

- Primitive classes in OWL are introduced as follows:
  - In the functional-style syntax, using a subclass axiom:

    `SubClassOf(CE1 CE2)`
  - Similarly in other syntaxes.

- Example:

```
SubClassOf(a:Dog
    ObjectSomeValuesFrom(a:eats a:Bone))
```

Determining whether a class is defined or primitive

- **Defined**
  - The complete definition of the class is known and relevant.
  - When one wants the system to determine class membership (well, if we do not want to do this, why use OWL?).

- **Primitive concepts** are usually found near the top of a generalization hierarchy and **defined concepts** typically appear as we move further down by specializing general concepts with various restrictions.

Definitional vs. Incidental Properties

- It is important to **distinguish between a class' true definition and any incidental properties**.

- Example: Red Bordeaux wines are always dry. But the property of being dry is certainly not a part of the definition of the class `RedBordeauxWine` (only the color and the region define a wine to be a Red Bordeaux).

Definitional vs. Incidental Properties (cont'd)

- In OWL, incidental properties are asserted using **extra class axioms** (in addition to the axioms that define the class).

Definitional vs. Incidental Properties (cont'd)

- This distinction must be made in OWL for **all classes**, not just defined classes.

- The ontology engineer must decide on ontological grounds whether a **restriction** should be taken as
  - **Part of the meaning** of a class (and thus participate in classification).
  - **Derived property** to be inferred once class membership is known.

Individuals vs. Classes

- Imagine that we are developing a **knowledge base of Greek foods and wines.**

- Consider the following terms:
  - Wine (class)
  - Red Wine (class)
  - Xinomavro (class)
  - Xinomavro Boutari (class or individual?)

Individuals vs. Classes (cont'd)

- For an application that will recommend wine to eaters (e.g., Xinomavro Boutari goes nicely with kontosouvli), Xinomavro Boutari can be an **individual**.

- What if we are also interested in the year the wine was produced?
  - "2004 Xinomavro Boutari" is a better choice for **individual** with Xinomavro Boutari being a **class** of which the former is an instance.

Individuals vs. Classes (cont'd)

- What if the ontology covers the inventory of the restaurant?
  - Individual bottles are the appropriate **individuals** to have.

Classes vs. Properties

- In a natural language description of the domain, usually **nouns** (or noun phrases) suggest the use of classes while **verbs** (or verb phrases) suggest the use of properties.

- Example: A Bordeaux wine is any wine produced in the Bordeaux region of France.

Classes vs. Properties (cont'd)

- But there might be cases when it might be difficult to decide whether a **term** (in this case a **noun**) should be a concept or a property.
- Examples:
  - Father (e.g., "George is a new father" vs. "George is the father of Mary").
  - Grape (e.g., grape as a kind of food vs. grape used to make some kind of wine).

Classes vs. Properties (cont'd)

- The question to ask is:
  - Can the description **stand on its own** without implying an unmentioned object related to the object in question?
- If the answer is yes, then it should be a class otherwise it should be a property.

- If the term should play **both roles** then we can use the prefix "has" in the name of the property (as in "has-grape") to solve the problem.

Things to Remember (from Ontology 101 tutorial)

- There is no one correct way to model a domain— there are always **viable alternatives.**
- The best solution almost always depends on the **application** that you have in mind and the **extensions** that you anticipate.
- Ontology development is necessarily an **iterative process**.

Steps in Developing an Ontology

The **ontology development methodology** championed by Alan Rector and the CO-ODE group at the University of Manchester has the following steps:

1. **Establish the purpose of the ontology**
   - Without purpose, no scope, requirements, evaluation
   - Competency questions

2. **Consider re-using existing ontologies.** (but the rest of the steps apply for the "build the ontology from scratch" scenario).

3. **Informal/semi-formal knowledge elicitation**
   - Collect the terms
   - Organise terms informally
   - Paraphrase and clarify terms to produce informal concept definitions
   - Diagram informally

   Card sorting and **laddering** are two useful knowledge elicitation techniques that can be used here.

4. **Refine requirements and tests**

Steps in Developing an Ontology (cont'd)

5. **Implementation**
   - Paraphrase and comment at each stage before 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
6. **Evaluate and quality assure**
   - Against goals
   - Include tests for evolution and change management
   - Design regression tests and "probes"
7. **Monitor use and evolve**
   - Process not product!

The Steps in Detail

- See any of the following sources:
  - The ISWC2005 tutorial "Ontology Design Patterns and Problems: Practical Ontology Engineering using Protege-OWL" by Alan Rector, Natasha Noy, Nick Drummond and Mark Musen. Available at http://www.co-ode.org/resources/tutorials/iswc2005.
  - The Ontology Building slides of Alan Rector given as part of the course CS646 (Dept. of Computer Science, University of Manchester). Available at http://www.cs.man.ac.uk/~rector/modules/CS646/.

    I will use the presentation http://www.cs.man.ac.uk/~rector/modules/CS646/Lecture-Handouts/Lect-2-Ontology-building-2007.pdf .
  - The tutorial of the CO-ODE group at the University of Manchester on "Ontologies and OWL". Available at http://www.co-ode.org/resources/tutorials/intro/.

Readings

- Natalya F. Noy and Deborah L. McGuinness. Ontology Development 101: A Guide to Creating Your First Ontology. Available from http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html

  This is an excellent introductory paper.

- Ronald J. Brachman, Deborah L. Mcguinness, Peter F. Patel-Schneider, Lori Alperin Resnick and Alexander Borgida. Living With Classic: When and How to Use a KL-ONE-like language. In J. Sowa (eds.) Principles of Semantic Networks: Explorations in the Representation of Knowledge. Morgan Kaufmann 1991. Available from http://www-out.bell-labs.com/project/classic/papers/sowabook.ps.gz

  This paper uses the language Classic which is one of the early DL-based systems. It is a must if you want to understand DL-based systems and use them! It can be read very easily even if you have never seen Classic before.

Readings (cont'd)

- Recent tutorial by Robert Stevens concentrating on OWL2 using "family history" as an example. See http://www.cs.man.ac.uk/~stevensr/menupages/fhkb.php.

- See lots of other OWL related material produced at the University of Manchester at http://owl.cs.manchester.ac.uk/.

Readings (cont'd)

- Yimin Wang, York Sure, Robert Stevens, Alan L. Rector: Knowledge Elicitation Plug-In for Protégé: Card Sorting and Laddering. ASWC 2006: 552-565.

- A Rector. Modularisation of Domain Ontologies Implemented in Description Logics and related formalisms including OWL. in Knowledge Capture 2003, (Sanibel Island, FL, 2003), ACM, 121-128.
  Available from http://www.cs.man.ac.uk/~rector/home_page_rector/alr-papers.html

  This paper explains some of the rationale of various steps in the ontology development method we presented.

- Alan L. Rector, Chris Wroe, Jeremy Rogers, Angus Roberts: Untangling taxonomies and relationships: personal and practical problems in loosely coupled development of large ontologies. K-CAP 2001: 139-146
  Available from http://portal.acm.org/citation.cfm?id=500760

Readings (cont'd)

- Aldo Gangemi: Ontology Design Patterns for Semantic Web Content. International Semantic Web Conference 2005: 262-276.
- Valentina Presutti, Aldo Gangemi: Content Ontology Design Patterns as Practical Building Blocks for Web Ontologies. ER 2008: 128-141.

- Mikel Egaña, Alan Rector, Robert Stevens, Erick Antezana. Applying Ontology Design Patterns in bio-ontologies. EKAW 2008, LNCS 5268, pp. 7-16. Available from http://www.springerlink.com/content/d2lp476v0p281q73/?p=f9d5500ce8b24589b2baf5eef213b0f5&pi=3

- Portals for ontology design patterns:
  - http://ontologydesignpatterns.org/wiki/Main_Page
  - http://www.gong.manchester.ac.uk/odp/html/index.html