



Aalto University
School of Science



Ontology engineering

How to develop an ontology?

CS-E4410 Semantic Web – additional material

Jouni Tuominen

Department of Computer Science

Semantic Computing Research Group (SeCo), <http://seco.cs.aalto.fi>

jouni.tuominen@aalto.fi

Methodology for ontology engineering

- Simple structures can be developed fairly easily
- Be aware of too complex structures
 - *Focus on what is actually needed in your application*
 - *Ontology Engineering rather art than science*
- A good starting point:
 - *Natasha Noy, Deborah McGuinness:*
Ontology Development 101: A Guide to Developing Your First Ontology. *Stanford University, 2001.*
- More advanced concerns
 - *DOLCE and OntoClean*
<http://www.springerlink.com/content/5p86jk323xotjktc/fulltext.pdf>

A simple ontology development methodology

Based on (N. Noy, D. McGuinness, 2001)

http://protege.stanford.edu/publications/ontology_development/ontology101.pdf

An Ontology Editor: Protégé

The screenshot displays the Protégé ontology editor interface. The main window title is "http://www.co-ode.org/ontologies/pizza/pizza.owl". The menu bar includes "File", "Edit", "Reasoner", "Tools", "Refactor", "Tabs", "View", "Window", and "Help". The address bar shows the URL "http://www.co-ode.org/ontologies/pizza/pizza.owl".

The interface is divided into several panes:

- Active Ontology:** Contains tabs for "Entities", "Classes", "Object Properties", "Data Properties", "Individuals", "OWLviz", and "DL Query".
- Asserted Class Hierarchy: PizzaTopping:** A tree view showing the class hierarchy. The root is "Thing", which branches into "DomainConcept" and "ValuePartition". "DomainConcept" includes "Country", "Food", "IceCream", "Pizza", "PizzaBase", and "PizzaTopping". "ValuePartition" includes "Spiciness", which further branches into "Hot", "Medium", and "Mild".
- Selected entity:** Shows the selected entity "PizzaTopping".
- Class Annotations: PizzaTopping:** Displays annotations for the selected class, including a "label" with the value "CoberturaDaPizza@pt".
- Class Description: PizzaTopping:** Shows the class description, including "Equivalent classes", "Superclasses" (listing "Food"), "Inherited anonymous classes", "Instances", and "Disjoint classes" (listing "PizzaBase", "Pizza", and "IceCream").
- Object Properties:** Lists object properties such as "hasCountryOfOrigin", "hasIngredient", "hasSpiciness", and "isIngredientOf".

The bottom of the interface shows a status bar with "Object property hierarchy", "Data property hierarchy", and "Individuals".

Some general remarks

- There is no correct way
 - *Best solution depends on the application*
- The process is iterative
- The concepts should be close to objects and relations in your domain

Seven-step process

Step 1: Determine the domain and scope of the ontology

- What is the domain that the ontology will cover?
- For what we are going to use the ontology?
- For what types of questions the information in the ontology should provide answers?
- Who will use and maintain the ontology?

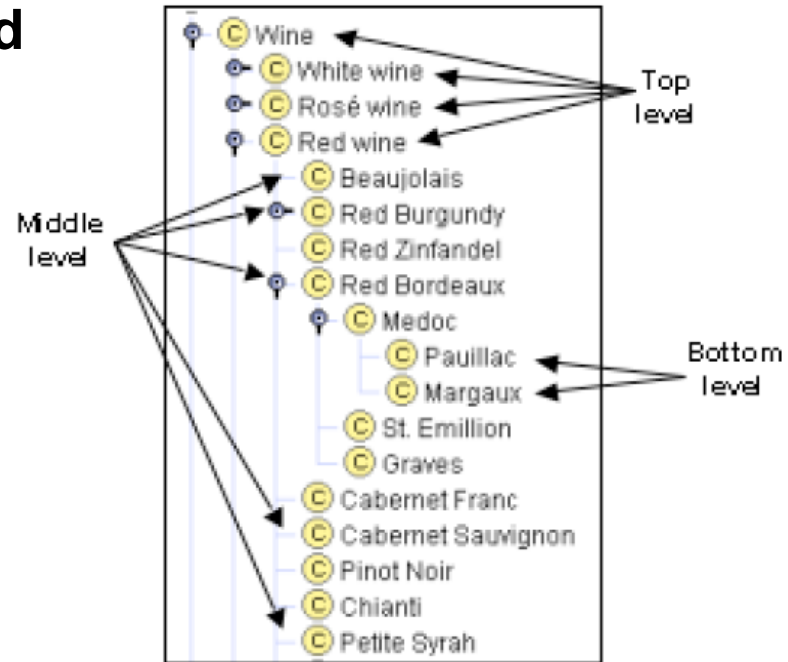
Step 2. Consider reusing existing ontologies

- E.g., ontology repositories: <http://onki.fi>, <http://finto.fi>

Step 3. Enumerate important terms in the ontology

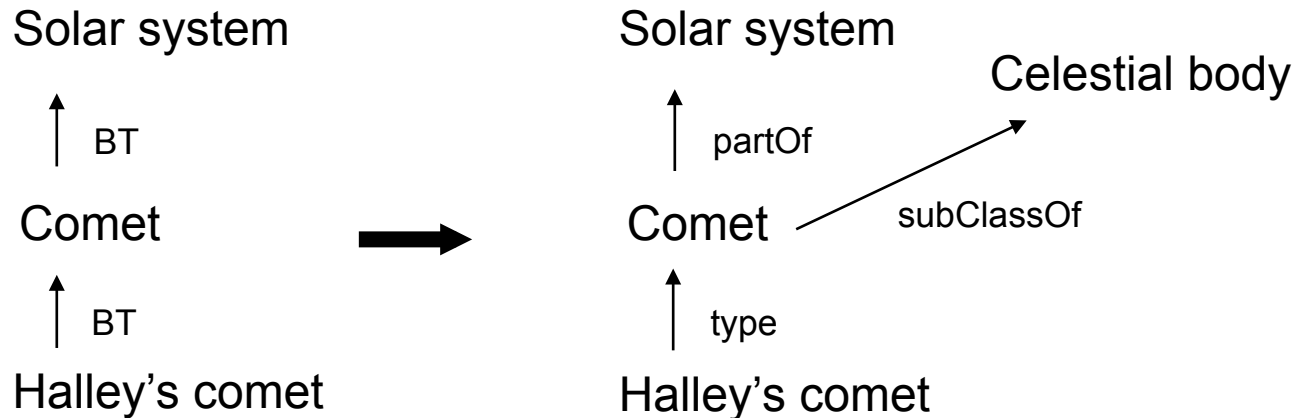
Step 4. Define the classes and the class hierarchy

- Top-down
- Bottom-up
- Mixed approach



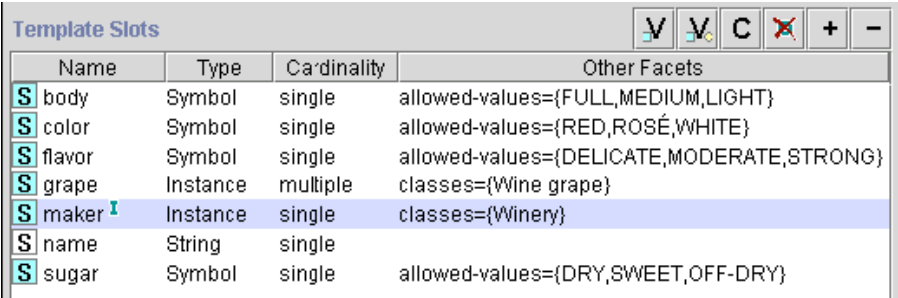
The most important principle in constructing the taxonomy

- If a class A is a superclass of class B, then every instance of B is also an instance of A



Step 5. Define the properties of classes—slots

- Intrinsic: e.g., flavor or color of wine
- Extrinsic: e.g., name or area of wine
- Structural properties (parts)
- Relations to other objects: e.g., grape of wine

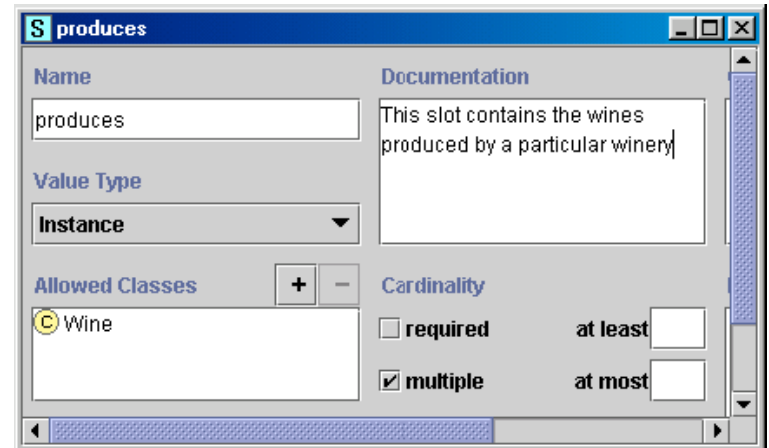


The screenshot shows a 'Template Slots' dialog box with a table of properties. The table has columns for Name, Type, Cardinality, and Other Facets. The 'maker' property is highlighted in blue.

Name	Type	Cardinality	Other Facets
S body	Symbol	single	allowed-values={FULL,MEDIUM,LIGHT}
S color	Symbol	single	allowed-values={RED,ROSÉ,WHITE}
S flavor	Symbol	single	allowed-values={DELICATE,MODERATE,STRONG}
S grape	Instance	multiple	classes={Wine grape}
S maker ¹	Instance	single	classes={Winery}
S name	String	single	
S sugar	Symbol	single	allowed-values={DRY,SWEET,OFF-DRY}

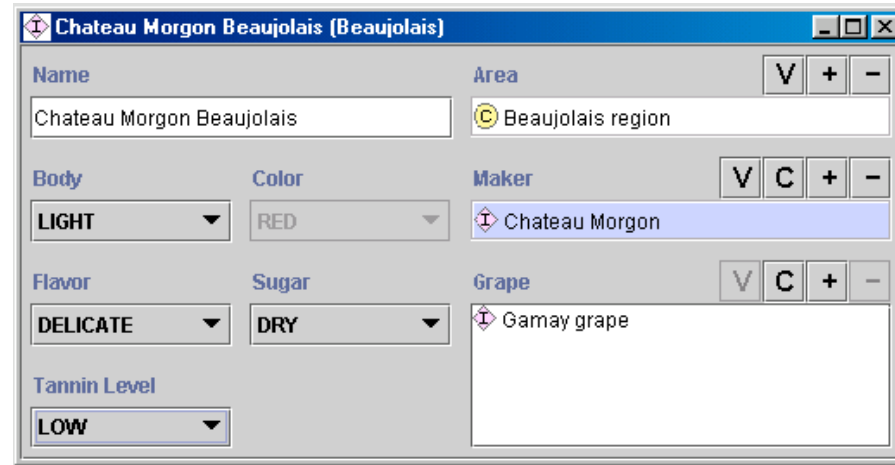
Step 6. Define the facets of the slots

- Cardinality
- Value type
 - *String, number, Boolean, Enumerated, Instance*
- Domain and range
 - *Constraints for properties*
 - E.g., the producer of a wine must be a winery



Step 7. Create instances

- Choose a class
- Create an instance
- Fill in slot values



The screenshot shows a software window titled "Chateau Morgon Beaujolais (Beaujolais)". The window contains several slots for defining the instance:

- Name:** Chateau Morgon Beaujolais
- Area:** Beaujolais region
- Body:** LIGHT
- Color:** RED
- Maker:** Chateau Morgon
- Flavor:** DELICATE
- Sugar:** DRY
- Grape:** Gamay grape
- Tannin Level:** LOW

Design decisions in constructing the class hierarchy

Ensuring that class hierarchy is correct

- Transitivity, cycles, concepts vs. names

Siblings

- At the same level of generality
- Usually 2-12 in one level

Multiple inheritance

- Sometimes needed
- Usually try to avoid

When to introduce a new class

- Useful terminological concepts
- Subclasses have additional properties or restrictions
- Participate in different relations

When to use a new class vs. a property value?

- Restrictions needed in other concepts need new concepts
- Different kind of instances need different classes
- For example:
 - *white-wine goes-with fish* → *White-wine is a class*
 - *chilled wine* → *not a class, use property (instance-class membership should not change often)*

When to use instances vs. classes

- Instances are most specific concepts
 - *E.g., wine types vs. bottles of wine*
- In concepts for a natural hierarchy → use classes
 - *French-region > Bordeaux-region > St-Emilion-Region*

Limiting scope

- The ontology should not be too detailed: think about the application

Disjointness and exhaustiveness of subclasses

Naming conventions

- Be consistent and systematic
- Label naming for humans
 - *Unique labels in different languages*
 - *Understandable without ontology context?*
 - Labels will be seen in applications
- Concept naming (ID) for machines (and programmers)
 - *Content neutral URIs vs. names*
 - *P12345 vs. Finland*
 - *Understandable without context?*
 - *Finland vs. Finland-photos*
 - *Syntactic choices*
 - *Meal course vs. MealCourse vs. Meal-Course vs. meal_course*
 - *Singular vs. plural?*
 - *Cat vs. Cats*
 - *Prefix and suffix conventions*
 - *has-father vs. father-of*