Semantic Data

Chapter 5 part 2 : Ontology engineering

Jean-Louis Binot

Sources

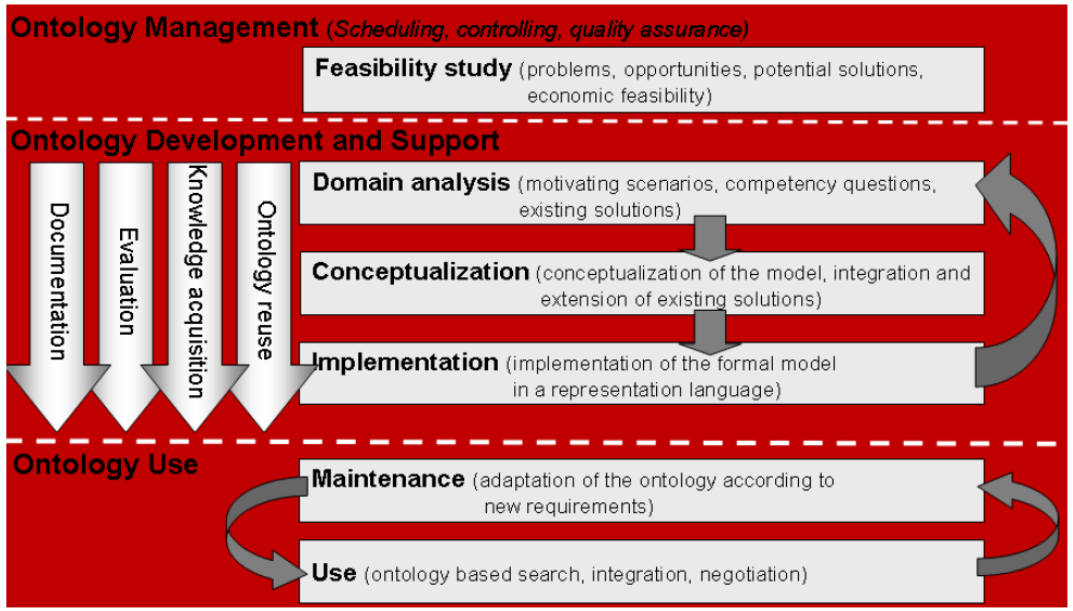
- There are no additional required references for this chapter.
- Sources and useful additional readings :
  - The ontology engineering methodology is based on *"Ontology Development 101: A Guide to Creating Your First Ontology" (Noy and McGuinness 2001)*.
  - The design criteria are based on *(Gruber 1993)*.

Ontology Engineering

- Ontology engineering : the process of building and maintaining an ontology.
- Main objective : build a structured and formalized representation of a domain, including:
  - Concepts (classes)
  - Properties (roles, slots)
  - Hierarchies
  - Restrictions
  - Rules…
- Ontology engineering has additional concerns :
  - Methodology : developing a real-life ontology is part of an industrial project.
  - Solution architecture : the ontology is often only one component of a larger solution, which may include a database, interfaces, processing subsystems …
  - Process : the ontology needs to be successfully deployed in an organization to achieve its benefits.

Ontology engineering process

- Knowledge acquisition and formalization is only one part of the engineering process.
- Real-life ontologies, like all IT applications, require an industrial-strength process, from feasibility to maintenance.



Main tasks in ontology engineering (Simperl et al. 2010)

Ontology engineering : initial considerations

- The ontology should reflect as far as possible the reality of the domain, but…
  - There are limits to our capability to model reality.
  - There is no unique correct way to model a domain : there are always viable alternatives.
  - There are nevertheless mistakes to avoid.

- An ontology in Computer Sciences is an engineering artifact.
  - Heated arguments have occurred about ontologies between philosophers and computer scientists.
  - Philosophy has a stronger requirement of trying to model accurately the real world (“what is”).
  - From an engineering point of view the test of an artifact is whether it is fit to purpose.

Design criteria

- **Clarity**
  - An ontology should effectively communicate the intended meaning of the defined terms.
- **Coherence**
  - An ontology should sanction inferences that are consistent with the definitions.
- **Extendibility**
  - An ontology should be designed to anticipate the uses of the shared vocabulary.
- **Minimal encoding bias**
  - The conceptualization should be specified at the knowledge level without depending on a particular symbol-level encoding.
- **Minimal ontological commitment**
  - An ontology should require the minimal ontological commitment sufficient to support the intended knowledge sharing activities.

(from Gruber 1993)

Why is ontology engineering so hard?

It looks deceptively easy, but it is not ! Why ?

- Ontologies are tricky : modeling mistakes and inconsistencies occur frequently.
- Modeling reality (even a part of reality) is hard :
  - Complexity, ambiguities, inconsistencies; scientists and philosophers have been at it for thousands of years.
- Ontology languages are tricky :
  - They rely on logic but with restrictions ensuring computability and tractability.
  - Most people are not trained in logic.
  - Multiple layers coexist in the W3C stack of standards.
- Complex design decisions must be made :
  - Fit to purpose, extensible, scalable, maintainable…
  - Many languages, syntaxes, tools.
- Alignment for reuse remains a challenge.

How is it different from object-oriented modeling ?

- The purpose is different:

  An ontology reflects the structure of the world, it is about the meaning and organisation of knowledge.

  An UML model reflects an allowed set of system states, it is about the constraints the system must meet.

  An OO class structure reflects the structure of the data and code, is about execution behavior (methods).

- Main differences between description logics/OWL-based ontologies and UML :

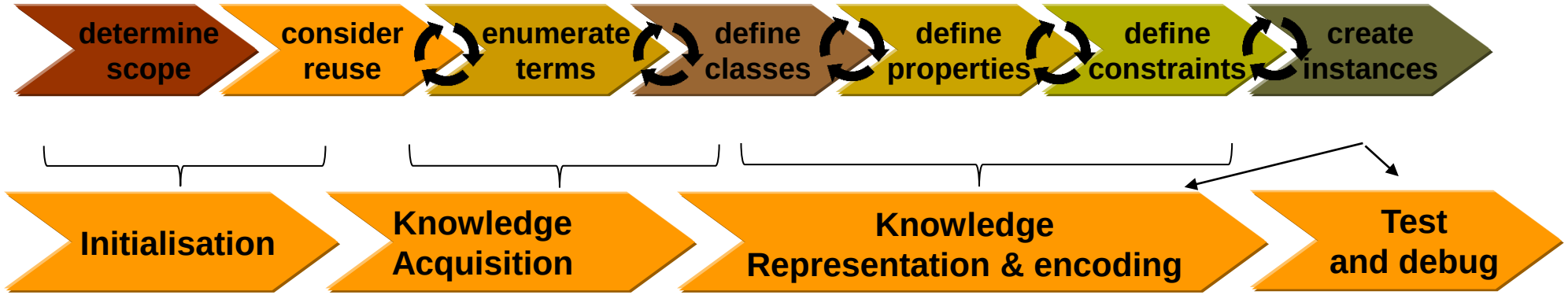
1. DL/OWL support inferences based on model-theoretic semantics. UML has no established formal semantics.
2. DL/OWL use a concept description language to model intensional knowledge (UML can use constraints to emulate some intensional information).
3. They use different world model hypotheses : DL/OWL use OWA and no UNA; UML uses CWA and UNA.
4. DL/OWL use global scope (of properties) …

- As a result, DL/OWL is not fully translatable into UML.

(cf. a.o. Atkinson & Kiko 2005)

A simple knowledge engineering methodology

- Multiple methodologies for ontology engineering have been proposed.
- We use a simple 7 steps process proposed by *(Noy and McGuinness 2001)* to illustrate tasks and issues.
  - These 7 steps can be related to a typical knowledge engineering process.
  - This is not a linear process : iteration can and will take place at every stage !



What are we missing ?

Simple process step 1: determine the scope

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

As in any IT project, answers to these questions may evolve during the lifecycle !

Competency questions

- Competency questions are one way to determine the scope of the ontology :
  - Write down a list of questions that a knowledge base built upon the ontology should be able to answer;
  - These questions do not need to be exhaustive, just a "sketch";
  - They will be reused for testing the ontology later.

Examples of competency questions for the wine ontology used in chapter 6 :

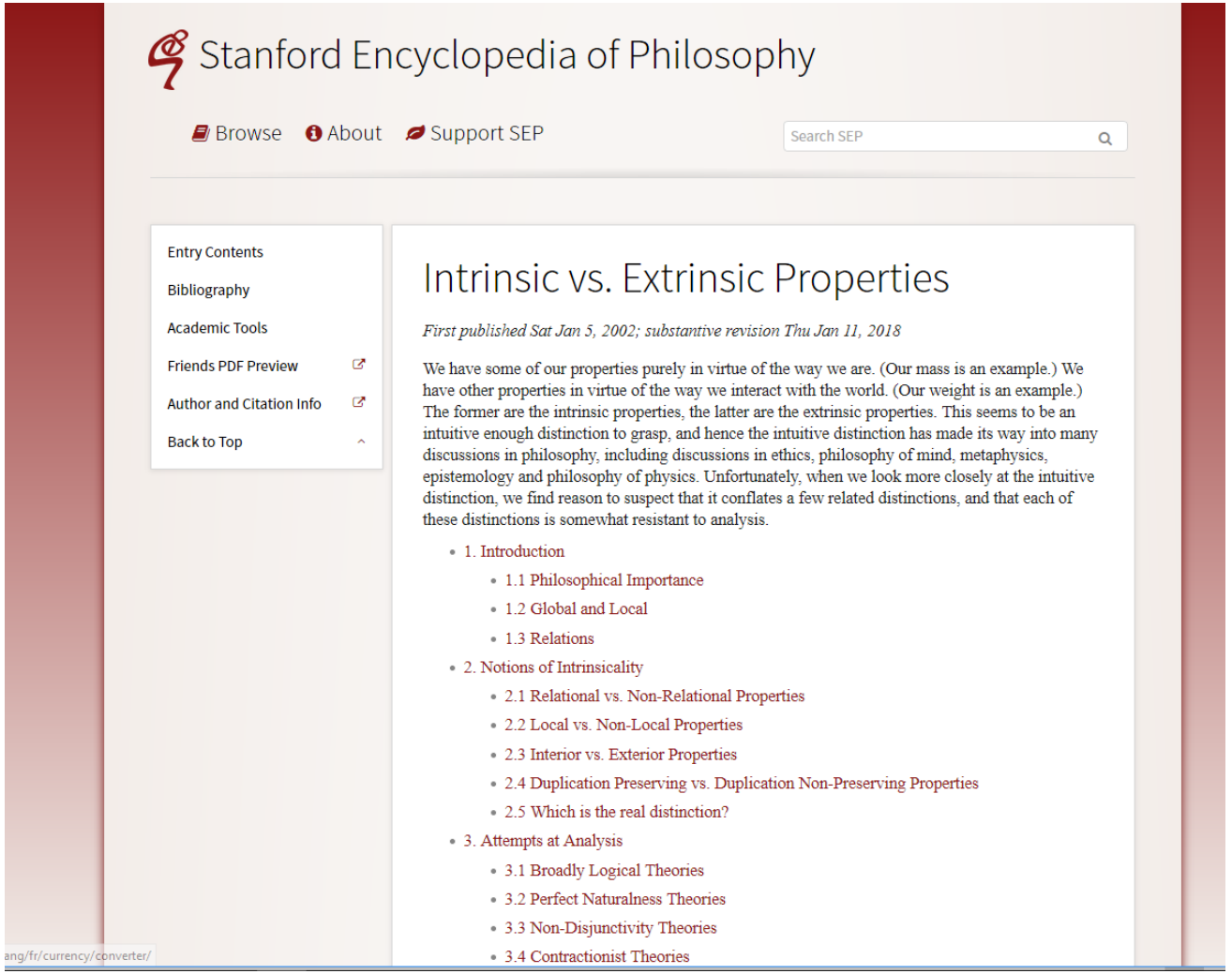
- *Which wine characteristics should I consider when choosing a wine ?*
- *Is Bordeaux a red or white wine ?*
- *Does Cabernet Sauvignon go well with seafood ?*
- *What is the best choice of wine for grilled meat ?*
- *Which characteristics of a wine affect its appropriateness for a dish ?*
- *Does a flavor or body of a specific wine change with vintage year ?*
- *What were good vintages for Napa Zinfandel ?*

Step 2 : consider reuse

- Why reuse other ontologies?
  - To save efforts;
  - To interact with the tools that already make use of other ontologies;
  - To use ontologies that have been validated through other applications.
- What to reuse:
  - General ontologies / upper level ontologies /domain specific ontologies / ontology fragments …

Reuse : consider philosophy

- Philosophy has also an input to provide about modelling the world…

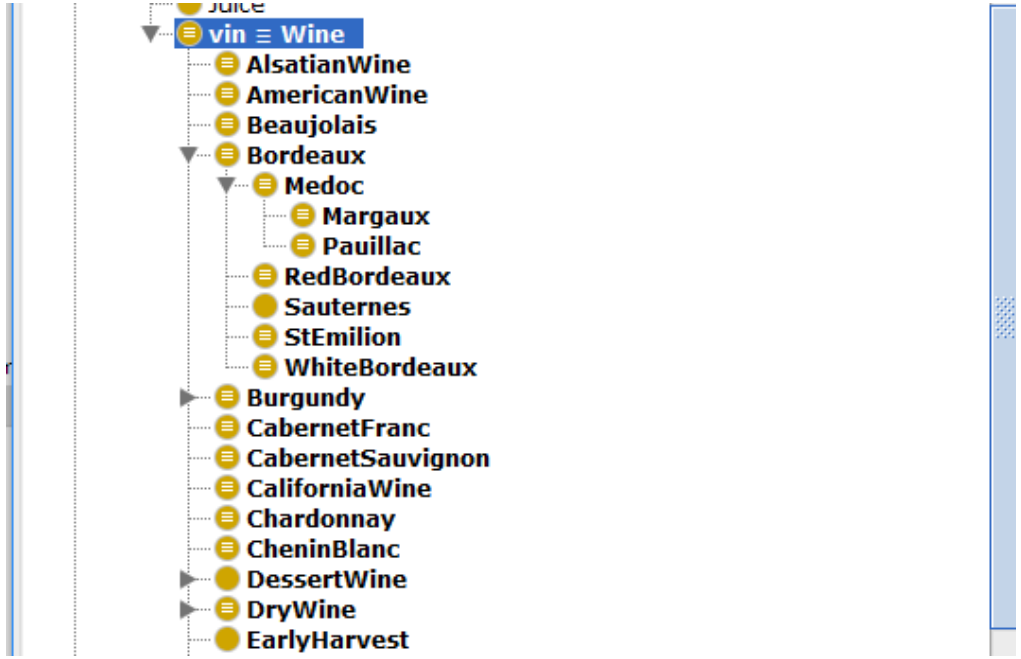


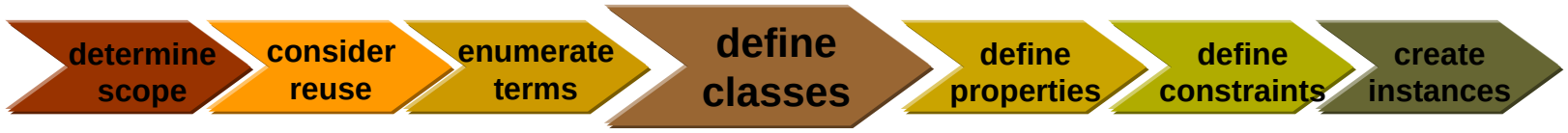
Step 3 : enumerate important terms

- What are the terms we need to talk about ?
- What are the properties of these terms ?
- What do we want to say about them ?
- Initially, it is important to get a comprehensive list of terms without worrying about overlap between concepts, relations among them, or any properties they may have.
  - Example : *wine, grape, winery, location, wine color, wine body, wine flavor, sugar content, white wine, red wine, Bordeaux wine, food, seafood, fish, meat, vegetables, cheese …*
- The structuring of these terms into a model comes in later steps.

Step 4 : define Classes and the Class Hierarchy

- Sort your terms :
  - Which ones denote objects ?
  - Which ones denote properties ?
- Formulate classes corresponding to the types of objects identified.
- Start building the hierarchy…
- Be cautious about multiple inheritance !
  - OWL does not prevent multiple inheritance and does not support default inheritance.
- Use an ontology editor !





Criteria for checking the quality of the class hierarchy

What criteria can be used ?

- Reuse the broad categories (e.g., objects, animates, events…) of an upper ontology.
- Respect the meaning of the subsumption relation (e.g., do not mix it with part_of).
- Check for synonyms : they normally have the same extension and not should not be distinct.
- Avoid cycles !
- Keep siblings at the same level of generality :
  - *WhiteWine* and *Chardonnay* should not be both direct subclasses of *Wine*. *WhiteWine* is more general.
- Have a reasonable span for the hierarchy structure at each level :
  - Too few subclasses (1 or 2) may indicate an incomplete ontology or a modeling problem;
  - Too many subclasses: some meaningful intermediate levels may be missing.

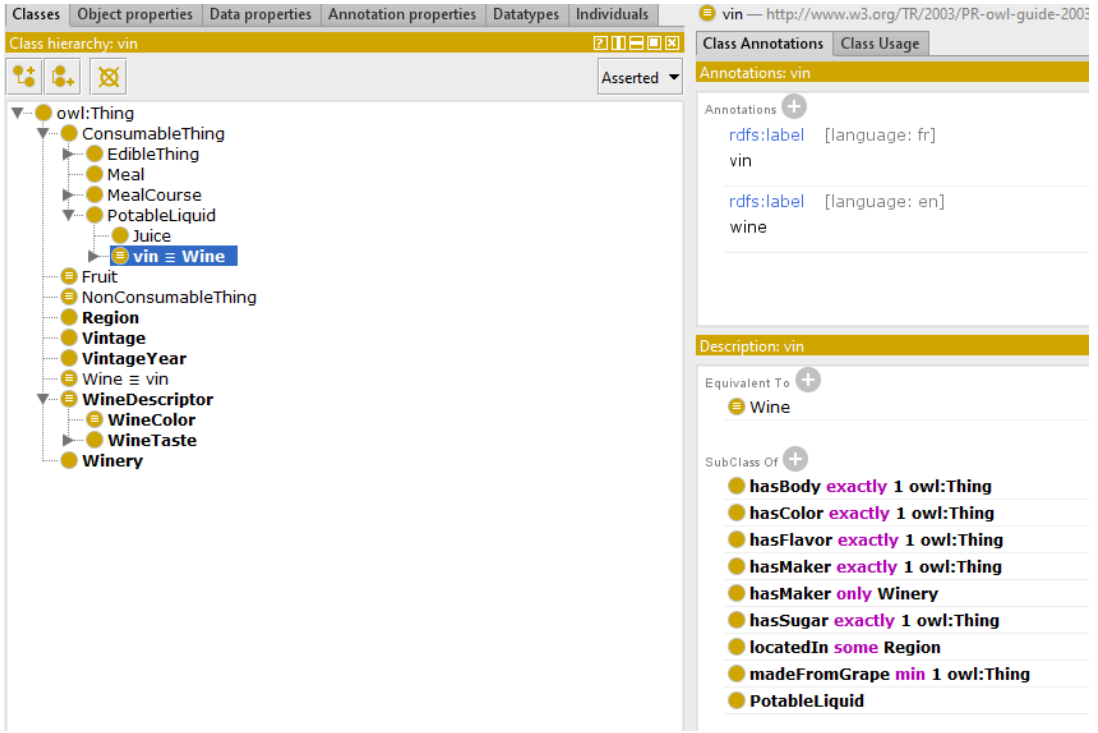
More specific criteria will be discussed during the practice sessions.

Step 5 : define properties

- Best handled at the same time as classes, while sorting through the vocabulary.
  - Properties may be intrinsic (color) or extrinsic (price), relating to other objects, or to data values.
- Associate properties to the highest class in the hierarchy to which they apply.
- Define their domain and range :
  - General enough for convenience of use and specific enough to detect inconsistencies.
  - Domain and range have global scope (cf. ch. 3); use role restrictors if you need to be more specific.
- Define the property hierarchy.
- Start capturing property characteristics :
  - Inverse, functional, transitive, symmetric… (cf. chapter 5 part 1).
  - Careful attention to the mathematical and logical behavior of these characteristics is required !

Step 6: define constraints

- Write-down and encode the formal definition of each concept, as far as possible.
  - Add concept descriptions : set-based constraints, role restrictions, cardinality constraints…
- Add or complete datatypes for property values.
- Iterate where needed.
- Check consistency of the ontology with a tool (e.g. Protégé) !



Step 7 : create instances

- Create instances.
  - Foresee at least enough instances to support testing.
- Indicate which ones are distinct.
- Fill in property values for the instances.
- Notes :
  - Some ontologies may not have instances.
  - Classes and instances may be kept in separate OWL files.

Document your ontology !

- Classes and properties should have associated documentation:
  - Describing the class in natural language, capturing intended use…
  - Listing domain assumptions relevant to the class definition;
  - Listing synonyms…
- Documenting an ontology is as important as documenting computer code !
- It is done through annotations.

Use a consistent naming scheme !

The naming scheme is part of the ontology documentation. As for programs, being systematic with names avoid stupid mistakes. Choices to make :

- Systematic naming convention for classes :
  - Same part of speech (noun), always plural or always singular.
  - If subclasses include the name of the superclass (*RedWine* and *Wine*), be systematic about it.
- A capitalization scheme :
  - e.g. class names start with a capital letter, property names do not …
- Word delimiter scheme :
  - Camel notation (*isMadeBy*) ; dash notation (*is-made-by*), underscore notation…
- Naming convention for properties :
  - Prefix convention (*hasMaker, has-maker*…) or suffix convention (*makerOf, maker-of*…).
- Avoid abbreviations !

Summary

- A Computer Sciences ontology is an engineering artifact, leading to design trade-offs. Its main requirement is not to model exhaustively the real world but to be fit to its purpose.
- A simple ontology engineering methodology in seven steps has been presented. This methodology fits within a typical knowledge engineering process.
- The full spectrum of an industrial-strength ontology engineering process is however wider, and, as for any IT project, should include considerations such as business benefits, feasibility, testing, deployment, maintenance…

References

- *[Atkinson & Kiko 2005] : Atkinson, C., & Kiko, K., A Detailed Comparison of UML and OWL (urn:nbn:de:bsz:180-madoc-18987).* https://madoc.bib.uni-mannheim.de/1898/.
- *[Gruber 1993]: Gruber, T. R.,* Towards Principles for the Design of Ontologies Used for Knowledge Sharing, *in "Formal Ontology in Conceptual Analysis and Knowledge Representation", Kluwer Academic Publishers, 1993.*
- *[Noy and McGuinness 2001] : Noy N. and McGuinness D.,* Ontology Development 101: A Guide to Creating Your First Ontology", *http://protege.stanford.edu/publications/ontology_development/ontology101.html.*
- *[Simperl et al. 2010]: Simperl, E., Mochol, M., and Burger, T. (2010).* Achieving maturity: the state of practice in ontology engineering, *in International Journal of Computer Science and Applications, 7(1):45-65, 2010.*

THANK YOU