Éléments de réflexion sur les composants d'ontologies et leur manipulation par RàPC

Béatrice Fuchs[1] et Amedeo Napoli[2]
[1] LIRIS, Université de Lyon 1, Nautibus, 8 bd Niels Bohr, F-69100 Villeurbanne
Email: bfuchs@liris.univ-lyon1.fr
[2] LORIA – UMR 7503 – BP 239, 54506 Vandœuvre-lès-Nancy
Email: Amedeo.Napoli@loria.fr

Séminaire RàPC
Paris, 30 juin 2009

A formal definition of an ontology

An ontology $\mathcal{O}$ is a system that (roughly) consists of:

- a set $\mathtt{S_C}$ of concepts organized within a hierarchy $\mathtt{H}$,
- in $\mathtt{H}$, concepts are hierarchically related by a subsumption relation $\sqsubseteq$ (specialization), generally reflexive, acyclic and transitive, where $c_1 \sqsubseteq c_2$ means that c_1 is a subconcept of c_2,
- a set $\mathtt{S_R}$ of binary relations specifying pairs $(\mathtt{D}, \mathtt{R})$ of domains and ranges (in $\mathtt{S_C}$) (that can also be organized within a relation hierarchy).

A well-designed ontology...

- Obeys to capital questions:
  - What are we talking about?
  - Why do we want to talk about it?
  - Where to find reusable knowledge?
- Whats, whys and wheres constitute the Problem Space of an ontology project.
- Ontology designers need to find solutions from a Solution Space.
- Matching problems to solutions is not trivial!

What is ontology design?

- Ontologies are artifacts:
- having a structure (linguistic, taxonomic, logical),
- having a function, i.e. to encode a description of the world (actual, possible, counter-factual, impossible, desired, etc.) for some purpose.
- Ontologies must match both domain and task:
- by allowing the description of the entities (domain) whose attributes and relations are concerned by some purpose,
- by serving a purpose (task).

Methods for ontology engineering

- Designing (parts of) an ontology manually and from scratch.
- Reusing existing ontologies and ontological resources.
- Using semi-automatic design methods, i.e. KDD techniques such as text mining based on Formal Concept Analysis (FCA) and Relational Concept Analysis (RCA).

Ontological resources (from informal to formal)

- Text corpora.
- Folksonomies: tag sets, directories, topic trees, subject indexes, etc.
- Lexica: dictionaries, wordnets, terminologies, nomenclatures.
- Knowledge organization systems: thesauri, classification schemas. frames, semantic networks.
- DB schemas.
- Computational ontologies.
- Suppose we need to design an ontology of desire... where to start from?

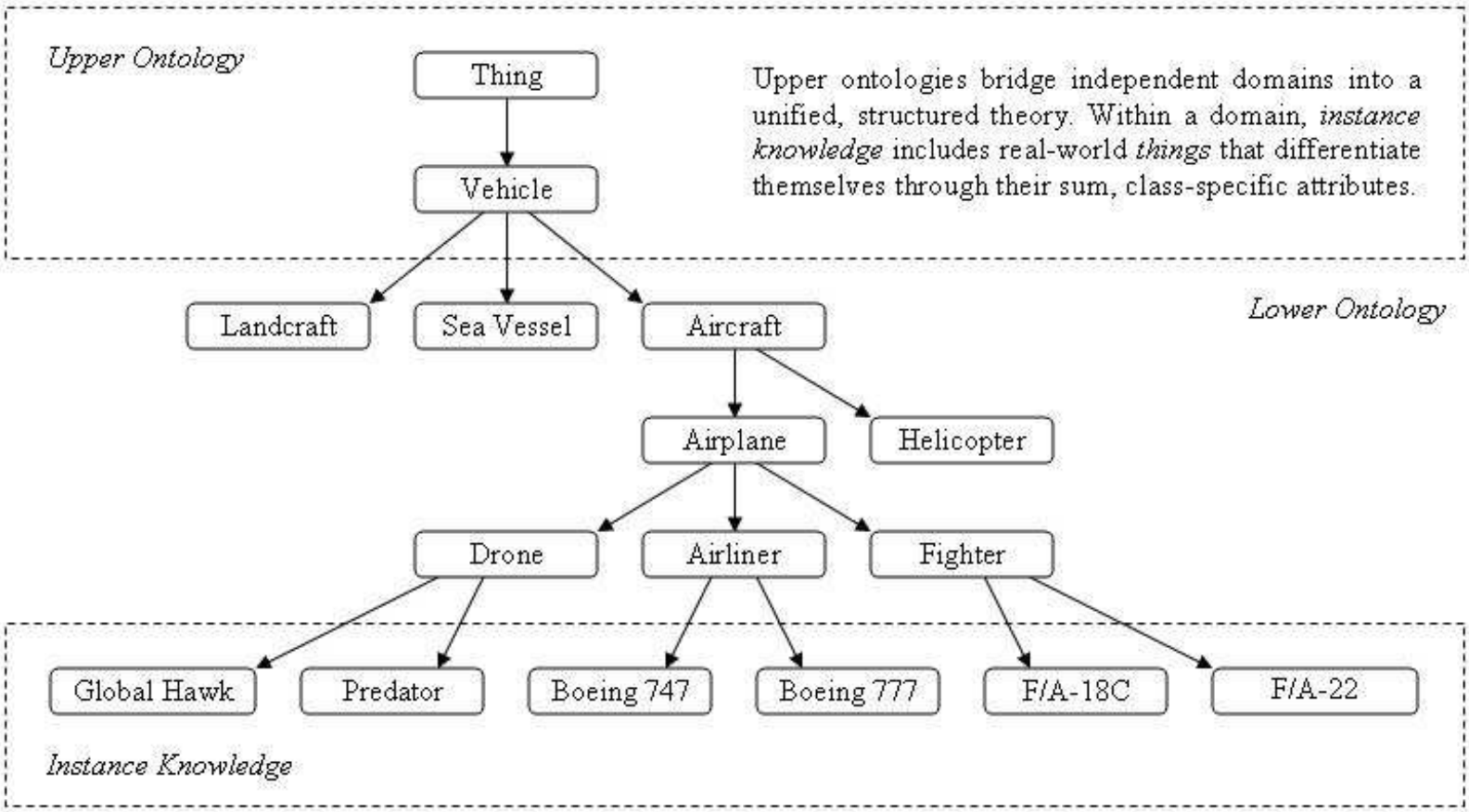
Constructing an ontology from scratch

- Determine scope: an ontology is a model.
- Consider reuse: an ontology has to be available for different needs.
- Enumerate terms: a list of relevant domain terms.
- Define taxonomy: a hierarchical organization of concepts, whenever `A` is a subconcept of `B`, then every instance of `A` must be an instance of `B`.

Constructing an ontology from scratch, continued...

- Define properties: with a domain and a range, and whenever A is a subconcept of B, then every property statement applying to instances of B must apply to instances of A.
- Define facets: cardinality, special values, relational characteristics (transitivity, symmetry, functionality, inverse properties...)
- Define instances: instance or concept, concept with only one instance...
- Check for anomalies.

A practical example



Common misconceptions

- Primitive classes are (most of the time) disjoint.
- And and/or Or?: conjunction and disjunction are interrelated.
- Relations have a domain and a range (useful for inferences).
- Existential and universal quantification do not have the same role and characteristics but are complementary.
- Closed and open worlds do not have (again) the same semantics and may be sources of confusion.

Pattern-based design

- Ontology design can be presented as the activity of searching, selecting, and composing different ontology design patterns (ODPs).
- ODPs provide a framework for understanding modeling choices –in the "Solution Space"– w.r.t. task and domain oriented requirements –in the "Problem Space".
- ODPs are comparable to software engineering design patterns.
- The concept of ODP is associable with the wider good/best practice of software engineering, but there may be some differences.

A quasi definition of ODPs

- In software engineering, a design pattern is defined as a simple and elegant solution to specific problems in object-oriented software design.
- In ontological engineering, ODPs have emerged as a way of helping ontology practitioners to model OWL ontologies.
- Principles about software DPs can be reused, adapted, and extended for ODPs.
- ODPs are modeling solutions to (standard) design problems in ontology engineering, based on best practices and making ontology design easier.

An example

The partition pattern:

- Given concepts $\mathtt{A}$ and $\mathtt{B}$:
- $\mathtt{A} \sqsubseteq \mathtt{A} \sqcup \mathtt{B}$
- $\mathtt{B} \sqsubseteq \mathtt{A} \sqcup \mathtt{B}$
- $\mathtt{A} \sqcup \mathtt{B} \equiv \top$
- $\mathtt{A} \sqcap \mathtt{B} \sqsubseteq \mathtt{A}$
- $\mathtt{A} \sqcap \mathtt{B} \sqsubseteq \mathtt{B}$
- $\mathtt{A} \sqcap \mathtt{B} \equiv \bot$

An example of real-world knowledge (FOAF)

- A Man is a Human.
- A Woman is a Human.
- No Man is a Woman (and *vice versa*).
- A Human is either a Man or a Woman.
- A Group of humans is (defined as) a Set with at least 2 members which are all Humans.

Another example

The graph pattern:

- A molecule can be represented by a graph $\mathtt{G} = (\mathtt{V}, \mathtt{E})$ where $\mathtt{V}$ is a set of vertices and $\mathtt{E}$ is the set of edges.
- A molecule is composed of a set of atoms and bonds.
- An atom has a name and a type (and possibly other additional characteristics).
- A bond has a name and a type and an order (and possibly other additional characteristics).

Different types of ODPs

The different types of ODPs can be distinguished by grouping them into six families:

- content,
- correspondence,
- lexical,
- presentation,
- reasoning,
- structure.
- Each family addresses different kinds of problems, and can be represented with different levels of formality.

Presentation ODPs (Definition)

- Presentation ODPs deal with usability and readability of ontologies from a user perspective.
- Naming ODPs are an example of presentation ODPs.
- Naming ODPs are conventions on how to create names for namespaces, files, and ontology elements in general (classes, properties, etc.).
- They are considered as good practices for enhancing ontology readability and understanding by humans, by supporting homogeneity in naming procedures.

Reasoning ODPs: classification and subsumption

- Reasoning ODPs are applications of Logical ODPs oriented to obtain certain reasoning results, based on the behavior implemented in a reasoning engine.
- Instance classification:

  $\mathtt{ChemFunction(x)} \equiv$
  $\mathtt{ChemStructure(x)} \wedge \exists \mathtt{y(hasFunction(x,y))}$
  $\{\mathtt{ChemStructure(C123)}, \mathtt{hasFunction(C123, Alcohol)}\}$
  $\models \mathtt{ChemFunction(C123)}$
- Class classification:

  $\mathtt{BioStructure(x)} \Longrightarrow \mathtt{ChemStructure(x)}$
  $\mathtt{hasAcidFunction(x,y)} \Longrightarrow \mathtt{hasFunction(x,y)}$
  $\models \mathtt{BioStructure(x)} \wedge \exists \mathtt{y(hasAcidFunction(x,y))} \Longrightarrow$
  $\mathtt{ChemFunction(x)}$

Inheritance and Instantiation ODPs

- Inheritance:

  $\texttt{ChemStructure(x)} \Longrightarrow \texttt{Structure(x)}$

  $\texttt{BioStructure(x)} \Longrightarrow \texttt{ChemStructure(x)}$

  $\models \texttt{BioStructure(x)} \Longrightarrow \texttt{Structure(x)}$

- Instantiation:

  $\texttt{hasFunction(x, y)} \equiv \texttt{isFunctionOf(y, x)}$

  $\texttt{hasFunction(C123, Alcohol)}$

  $\models \texttt{isFunctionOf(Alcohol, C123)}$

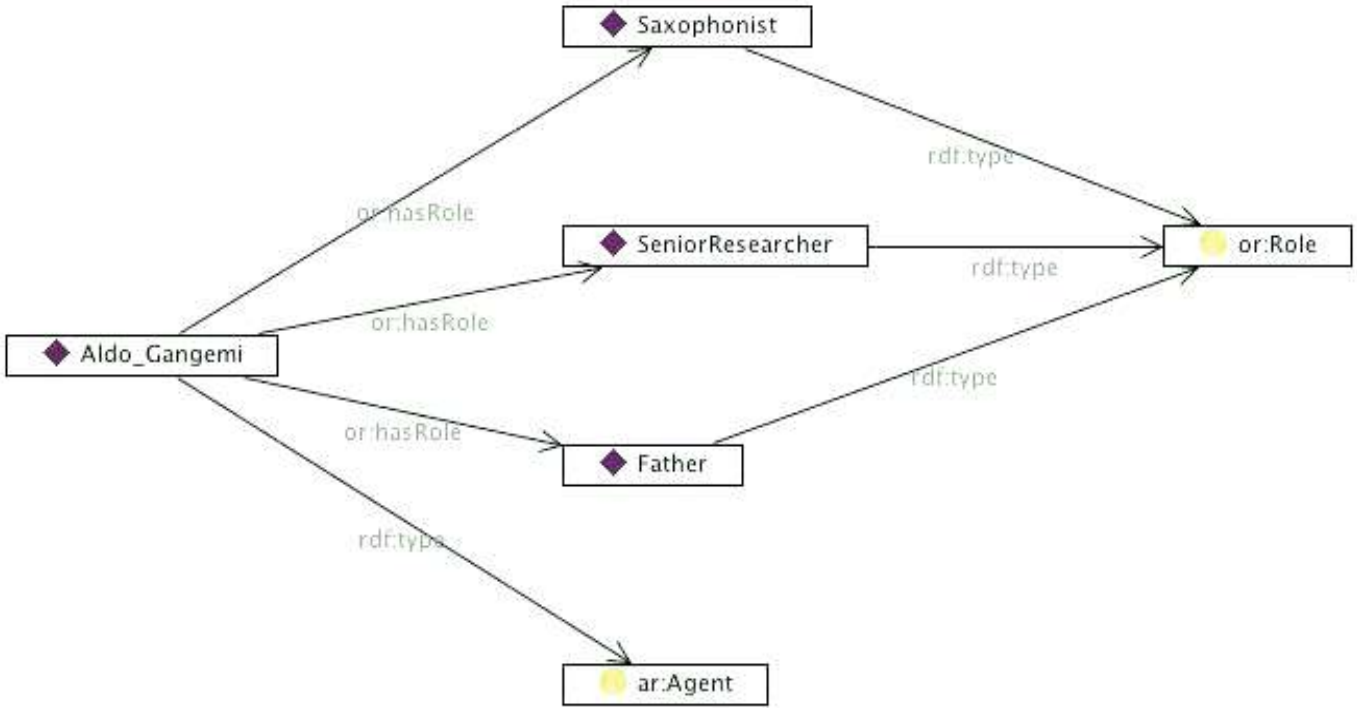
Logical macros

- Logical ODPs are compositions of logical constructs that solve a problem of expressivity.
- Logical macros provide a shortcut to model a recurrent intuitive logical expression.
- Example: the macro: $\nabla \mathtt{R}.\mathtt{C}$ means every y in relation with x through R must be of type C, formally: $\exists \mathtt{R}.\mathtt{C} \sqcap \forall \mathtt{R}.\mathtt{C}$

Content ODPs (CODPs)

- Content ontology design patterns (CODPs) encode conceptual patterns and address content problems.
- Modeling problems solved by CODPs have two components: domain and requirements.
- A typical way of capturing requirements is by means of competency questions (CQs) they satisfy.

An example of CODP: Agent Role Instantiation



CBR: retrieving ODPs

- Selection among a set of ODPs is "quite" similar to the task of ontology ranking performed by ontology search engines (OntoCase, Swoogle, Watson, Sindice).
- In ranking schemas (e.g. as proposed in AktiveRank), "similarity measures" are used:
  class matches based on string matching for concept names,
  centrality based on placement for evaluating representativeness,
  density measuring the detail degree in number of relations,
  and semantic similarity measuring the proximity in terms of path lengths traversing the ontology relations.
- and also coverage, structure, and connectedness, etc.

Adapting patterns

- The adaptation phase constitutes the process of specializing, extending and composing the patterns into a first version of the ontology.
- It can be hard to reuse only the "useful pieces" of an ontology, and consequently the cost of reuse can be higher than developing a new ontology from scratch.
- It is envisioned small or cleverly modularized ontologies with explicit documentation of design rationales, and best reengineering practices.

Conclusion

- Work in progress.
- Relations between CBR and ODP management: retrieval and adaptation.
- Semi-automatic design of ODPs and of ontologies: adaptation of text mining methods for doing the job.
- Thanks for your attention!