Ontology Engineering

Lecture 5: Methods and Methodologies

Maria Keet

email: mkeet@cs.uct.ac.za

home: http://www.meteck.org

Department of Computer Science
University of Cape Town, South Africa

Semester 2, Block I, 2019

Outline

Ontology development

- You have some experience with an ontology language and representing some knowledge, which was given to you
- How to come up with the whole ontology in the first place?
- What can, or should, you do when you have to develop your own ontology?

Ontology development

- You have some experience with an ontology language and representing some knowledge, which was given to you
- How to come up with the whole ontology in the first place?
- What can, or should, you do when you have to develop your own ontology?

⇒ Just like in software engineering, there are methods and methodologies to guide you through it

Ontology development

- You have some experience with an ontology language and representing some knowledge, which was given to you
- How to come up with the whole ontology in the first place?
- What can, or should, you do when you have to develop your own ontology?

⇒ Just like in software engineering, there are methods and methodologies to guide you through it

- Recall (L1): an ontology is a logical theory "plus some more"
- In this Block II, we also look into that "some more"

Topics for this lecture

- **Methodologies**
  - Most are coarse-grained, i.e., a macro-level, processual information systems perspective; they do not (yet) contain all the permutations at each step
  - The actual modelling, or *ontology authoring*, using micro-level guidelines and tools

Topics for this lecture

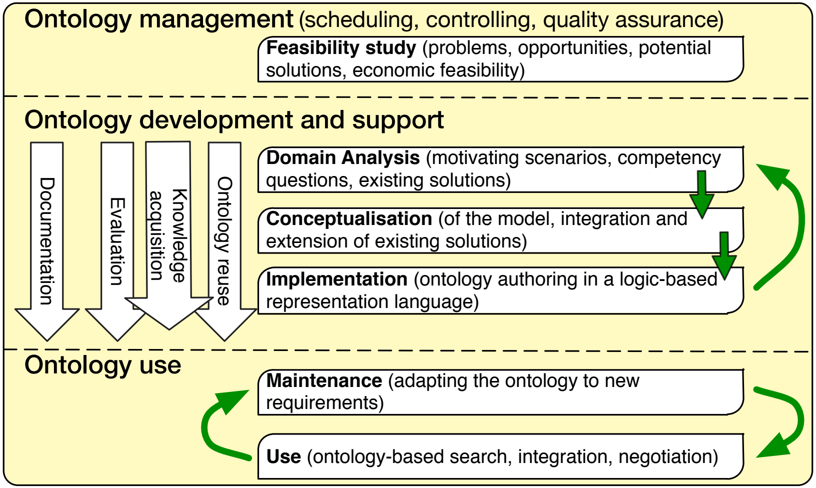
- **Methodologies**
  - Most are coarse-grained, i.e., a macro-level, processual information systems perspective; they do not (yet) contain all the permutations at each step
  - The actual modelling, or *ontology authoring*, using micro-level guidelines and tools
- **Methods**, e.g.,:
  - Reverse engineering and text mining to start
  - OntoClean and OntoParts to improve an ontology's quality
- **Parameters** that affect ontology development (e.g.,purpose, starting/legacy material, language)
- **Tools** to model, to reason, to debug, to integrate, to link to data

Outline

Outline

1. Methodologies and tools
   - Macro-level methodologies
   - Micro-level methodologies
2. Methods
   - Logic-based debugging
   - Logic & philosophy
3. Methods - philosophy
4. Parameters and dependencies

Typical stages of macro-level methodologies


Ontology management (scheduling, controlling, quality assurance)
Feasibility study (problems, opportunities, potential solutions, economic feasibility)
Ontology development and support
Documentation
Evaluation
Knowledge acquisition
Ontology reuse
Domain Analysis (motivating scenarios, competency questions, existing solutions)
Conceptualisation (of the model, integration and extension of existing solutions)
Implementation (ontology authoring in a logic-based representation language)
Ontology use
Maintenance (adapting the ontology to new requirements)
Use (ontology-based search, integration, negotiation)

Methodologies for ontology development

- Waterfall
- Lifecycle
- Agile

Methodologies for ontology development

- Waterfall
- Lifecycle
- Agile
- Variants among them, such as eXtreme Design and TDD approaches, and different waterfalls

Methodologies for ontology development

- Waterfall
- Lifecycle
- Agile
- Variants among them, such as eXtreme Design and TDD approaches, and different waterfalls
- They have not yet been compared in a controlled experiment
- Nor is it clear which is popular (read: actually used) among the developers of domain ontologies, if any

A waterfall methodology: Methontology

- Basic methodological steps:
  - Specification: why, what are its intended uses, who are the prospective users
  - Conceptualization: with intermediate representations
  - Formalization: transforms the domain-expert understandable ‘conceptual model’ into a formal or semi-computable model
  - Implementation: represent it in an ontology language
  - Maintenance: corrections, updates, etc.

A waterfall methodology: METHONTOLOGY

- Basic methodological steps:
  - Specification: why, what are its intended uses, who are the prospective users
  - Conceptualization: with intermediate representations
  - Formalization: transforms the domain-expert understandable ‘conceptual model’ into a formal or semi-computable model
  - Implementation: represent it in an ontology language
  - Maintenance: corrections, updates, etc.
- Additional tasks:
  - Management activities (schedule, control, and quality assurance)
  - Support activities (knowledge acquisition, integration, evaluation, documentation, and configuration management)

A waterfall methodology: METHONTOLOGY

- Basic methodological steps:
  - Specification: why, what are its intended uses, who are the prospective users
  - Conceptualization: with intermediate representations
  - Formalization: transforms the domain-expert understandable 'conceptual model' into a formal or semi-computable model
  - Implementation: represent it in an ontology language
  - Maintenance: corrections, updates, etc.
- Additional tasks:
  - Management activities (schedule, control, and quality assurance)
  - Support activities (knowledge acquisition, integration, evaluation, documentation, and configuration management)
- Applied to chemical, legal domain, and others

Extending the methodologies

- METHONTOLOGY and others (e.g., On-To-Knowledge, KACTUS approach) are methods for developing one *single* ontology
- Changing landscape in ontology development towards building "ontology networks"

Extending the methodologies

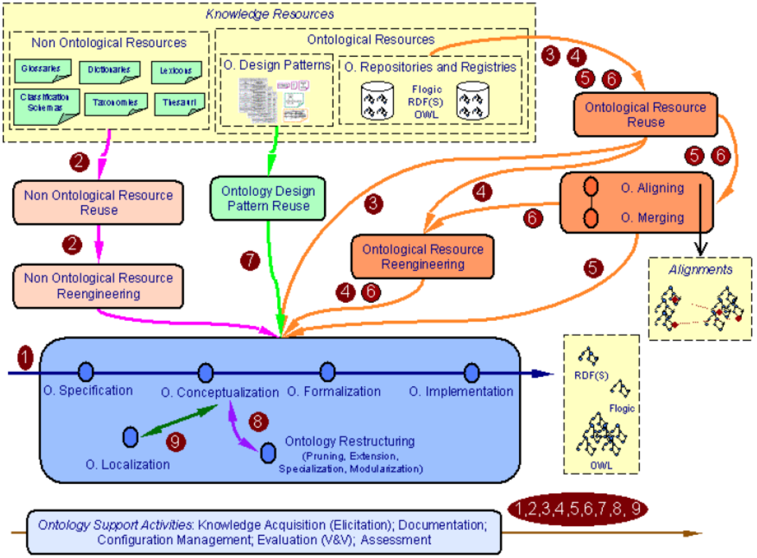
- METHONTOLOGY and others (e.g., On-To-Knowledge, KACTUS approach) are methods for developing one *single* ontology
- Changing landscape in ontology development towards building “ontology networks”
- Characteristics: dynamics, context, collaborative, distributed
- E.g., the NeOn methodology

Extending the methodologies: NeOn

- NeOn's "Glossary of Activities" identifies and defines 55 activities when ontology networks are collaboratively built
- Among others: ontology localization, -alignment, -formalization, -diagnosis, -enrichment etc.
- Divided into a matrix with "required" and "if applicable"
- Recognises there are several scenarios for ontology development, refining the typical monolithic 'waterfall' approach

(more info in neon_2008_d5.4.1.pdf)

Several scenarios for Building Ontology Networks


Knowledge Resources
Non Ontological Resources
Glossaries
Dictionaries
Lexicons
Classification Schemas
Taxonomies
Thesauri
Ontological Resources
O. Design Patterns
O. Repositories and Registries
Flogic
RDF(S)
OWL
Ontological Resource Reuse
Non Ontological Resource Reuse
Ontology Design Pattern Reuse
O. Aligning
O. Merging
Ontological Resource Reengineering
Non Ontological Resource Reengineering
Alignments
O. Specification
O. Conceptualization
O. Formalization
O. Implementation
O. Localization
Ontology Restructuring (Pruning, Extension, Specialization, Modularization)
RDF(S)
Flogic
OWL
1,2,3,4,5,6,7,8, 9
Ontology Support Activities: Knowledge Acquisition (Elicitation); Documentation; Configuration Management; Evaluation (V&V); Assessment

Tools for methodologies – not exactly

- Current tools support one or more aspects of a methodology
- e.g., WebProtégé, MOdelling wiKI **MoKi** for collaborative work, add a documentation plugin, a merging plugin etc.

Tools for methodologies – not exactly

- Current tools support one or more aspects of a methodology
- e.g., WebProtégé, MOdelling wiKI **MoKi** for collaborative work, add a documentation plugin, a merging plugin etc.
- e.g., **MoKi**'s collaboration:
  - based on a *SemanticWiki*, used for collaborative and cooperative ontology development
  - 'multi-modal access *at different levels of formality*: informal, semi-formal and formal (enables actors with different expertise contribute)

Outline

Micro-level methodologies

- Guidelines detailing how to go from informal to logic-based representations with instructions how to include the axioms and figure out which ones are better than others
- To represent the formal and ontological details
- (very) Expressive ontology language, so as to include guidance also for the axioms and ontological quality criteria

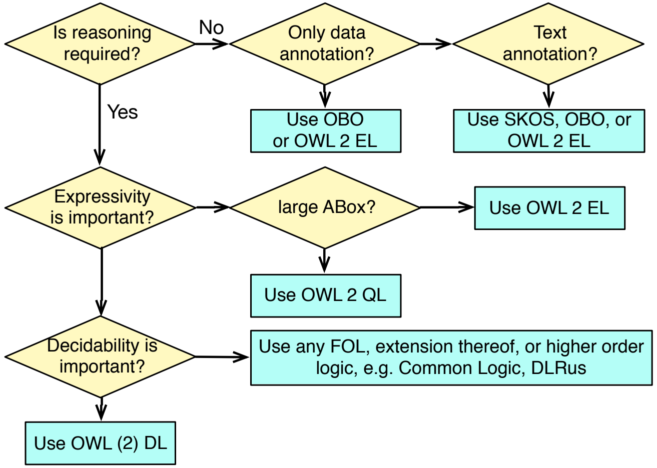
Micro-level methodologies

- Guidelines detailing how to go from informal to logic-based representations with instructions how to include the axioms and figure out which ones are better than others
- To represent the formal and ontological details
- (very) Expressive ontology language, so as to include guidance also for the axioms and ontological quality criteria
- Notably: OntoSpec, “Ontology development 101” (outdated!!!), DiDOn, TDD
- Methods & tools for sets of axioms; e.g.: FORZA, advocatus diaboli, *SubProS* & *ProChainS*, ...

More detailed steps (generalised from DiDOn) (1/2)

1. Requirements analysis, regarding expressiveness (temporal, fuzzy, n-aries etc.), types of queries, reasoning services needed;
2. Design an ontology architecture, such as modular, and if so, in which way, distributed or not, etc.
3. Choose principal representation language and consider encoding peculiarities;

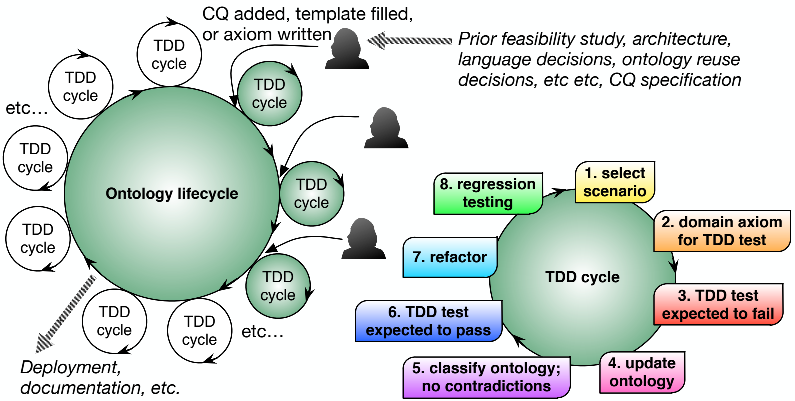
A few basic hints for choosing a language


Is reasoning required?
No
Only data annotation?
Text annotation?
Yes
Use OBO or OWL 2 EL
Use SKOS, OBO, or OWL 2 EL
Expressivity is important?
large ABox?
Use OWL 2 EL
Use OWL 2 QL
Decidability is important?
Use any FOL, extension thereof, or higher order logic, e.g. Common Logic, DLRus
Use OWL (2) DL

More detailed steps (generalised from DiDOn) (2/2)

4. Formalisation, including:
   - examine and add the classes, object properties, constraints, rules taking into account the imported ontologies;
   - use an automated reasoner for debugging/anomalous deductions;
   - use ontological reasoning services for quality checks (OntoClean, RBox Compatibility);
   - add annotations;
5. Generate versions in other ontology languages, 'lite' versions, etc, if applicable;

Test-driven Development


CQ added, template filled, or axiom written
Prior feasibility study, architecture, language decisions, ontology reuse decisions, etc etc, CQ specification
etc...
Ontology lifecycle
TDD cycle
etc...
Deployment, documentation, etc.
TDD cycle
1. select scenario
2. domain axiom for TDD test
3. TDD test expected to fail
4. update ontology
5. classify ontology; no contradictions
6. TDD test expected to pass
7. refactor
8. regression testing

The landscape

- Isn't ontology development just like conceptual data model development?

The landscape

- Isn't ontology development just like conceptual data model development?
  - **yes**: e.g., interaction with the domain expert, data analysis, "*knowledge acquisition bottleneck*"

The landscape

- Isn't ontology development just like conceptual data model development?
  - **yes**: e.g., interaction with the domain expert, data analysis, "*knowledge acquisition bottleneck*"
  - **no**: e.g., logic, automated reasoning, using (parts of) other ontologies, different scopes/purposes, specific isolated application scenario vs. general knowledge

The landscape

- Isn't ontology development just like conceptual data model development?
    - **yes**: e.g., interaction with the domain expert, data analysis, "*knowledge acquisition bottleneck*"
    - **no**: e.g., logic, automated reasoning, using (parts of) other ontologies, different scopes/purposes, specific isolated application scenario vs. general knowledge
- Difference between writing down what you did (to make it a 'guideline') vs. experimentally validating a methodology

The landscape

- Isn't ontology development just like conceptual data model development?
    - **yes**: e.g., interaction with the domain expert, data analysis, "*knowledge acquisition bottleneck*"
    - **no**: e.g., logic, automated reasoning, using (parts of) other ontologies, different scopes/purposes, specific isolated application scenario vs. general knowledge
- Difference between writing down what you did (to make it a 'guideline') vs. experimentally validating a methodology
- Difference between *method* and *methodology*

The landscape

- Isn't ontology development just like conceptual data model development?
  - **yes**: e.g., interaction with the domain expert, data analysis, "*knowledge acquisition bottleneck*"
  - **no**: e.g., logic, automated reasoning, using (parts of) other ontologies, different scopes/purposes, specific isolated application scenario vs. general knowledge
- Difference between writing down what you did (to make it a 'guideline') vs. experimentally validating a methodology
- Difference between *method* and *methodology*
- There are plenty methods and guidelines for ontology development—**use them**

Outline

Overview

- Domain experts are expert in their subject domain, which is not logic

Overview

- Domain experts are expert in their subject domain, which is not logic
- Modellers often do not understand the subject domain well

Overview

- Domain experts are expert in their subject domain, which is not logic
- Modellers often do not understand the subject domain well
- The more expressive the language, the easier it is to make errors or bump into unintended entailments

Overview

- Domain experts are expert in their subject domain, which is not logic
- Modellers often do not understand the subject domain well
- The more expressive the language, the easier it is to make errors or bump into unintended entailments
- Simple languages can represent more than it initially may seem (by some more elaborate encoding), which clutters the ontology and affects comprehension

Overview

- Domain experts are expert in their subject domain, which is not logic
- Modellers often do not understand the subject domain well
- The more expressive the language, the easier it is to make errors or bump into unintended entailments
- Simple languages can represent more than it initially may seem (by some more elaborate encoding), which clutters the ontology and affects comprehension
- In short: people make errors (w.r.t. their intentions) in the modelling task, and automated reasoners and other approaches can help fix that

Overview

The methods can be divided roughly into:

- logic-based only
- purely based on philosophy
- a combination of the former two
- practical rules or guidelines

Each of these categories has several methods with more or less tool support

Example of heuristics: OOPS!

- Pitfall: a (common) mistake, incorrect assumptions that may, or may not, concern the logic
- Among others: undesirable deductions, inconsistent naming scheme, declaring a property transitive but has incompatible domain and range
- Heuristic, because they are not always clear-cut cases and can have false positives; e.g.:
  - 'missing inverse': but some relations don't take one, not needed, or language has `inverse()` feature
  - '(in)consistency in naming': your ontology may be systematic, but an imported one may not be, or had a different naming convention
  - P7 merging different concepts: `StyleAndPeriod` should not be combined, but a `RumAndRaisin` flavour of ice cream does belong together

Outline

Overview

- Using automated reasoners for ‘debugging’ ontologies, requires one to know about reasoning services
- Using standard reasoning services
- Reasoning services tailored to pinpointing the errors and explaining the entailments

Common errors

- Unsatisfiable classes
  - In the tools: the unsatisfiable classes end up as direct subclass of `owl:Nothing`
  - Sometimes one little error generates a whole cascade of unsatisfiable classes

Common errors

- Unsatisfiable classes
  - In the tools: the unsatisfiable classes end up as direct subclass of `owl:Nothing`
  - Sometimes one little error generates a whole cascade of unsatisfiable classes
- Satisfiability checking can cause rearrangement of the class tree and any inferred relationships to be associated with a class definition: 'desirable' vs. 'undesireable' inferred subsumptions

Common errors

- Unsatisfiable classes
  - In the tools: the unsatisfiable classes end up as direct subclass of `owl:Nothing`
  - Sometimes one little error generates a whole cascade of unsatisfiable classes
- Satisfiability checking can cause rearrangement of the class tree and any inferred relationships to be associated with a class definition: ‘desirable’ vs. ‘undesireable’ inferred subsumptions
- Inconsistent ontologies: all classes *taken together* unsatisfiable

Common errors

- Basic set of clashes for concepts (w.r.t. tableaux algorithms) are:
  - Atomic: An individual belongs to a class and its complement
  - Cardinality: An individual has a max cardinality restriction but is related to more distinct individuals
  - Datatype: A literal value violates the (global or local) range restrictions on a datatype property

Common errors

- Basic set of clashes for concepts (w.r.t. tableaux algorithms) are:
  - Atomic: An individual belongs to a class and its complement
  - Cardinality: An individual has a max cardinality restriction but is related to more distinct individuals
  - Datatype: A literal value violates the (global or local) range restrictions on a datatype property
- Basic set of clashes for KBs (ontology + instances) are:
  - Inconsistency of assertions about individuals, e.g., an individual is asserted to belong to disjoint classes or has a cardinality restriction but related to more individuals
  - Individuals related to unsatisfiable classes
  - Defects in class axioms involving nominals (`owl:oneOf`, if present in the language)

Test-last vs test-first

- The previous list is useful for starting where to look when an error (unsat class, ontology) occurs
- Based on the approach ‘try and see’ what the reasoner says
- Idea of ‘unit tests’ also introduced in ontology engineering ⇒ like *test-last* in programming

Test-last vs test-first

- The previous list is useful for starting where to look when an error (unsat class, ontology) occurs
- Based on the approach 'try and see' what the reasoner says
- Idea of 'unit tests' also introduced in ontology engineering $\Rightarrow$ like *test-last* in programming
- TDD is a *test-first* approach; can we use that for ontology development?
  - yes. Theory, draft methodology, and tool for it
  - TDDonto2 description and link to Protégé plugin:

    `https://keet.wordpress.com/2016/12/15/`

    `improved-tddonto-v2-more-types-of-axioms-supported-and-better-feedback/`

Outline

RBoxes: Questions and Problems

- What are the features of a ‘good’ RBox w.r.t. object property expressions?
- Modelling flaws in the RBox show up as unexpected or undesirable deductions regarding classes in the TBox, but current explanation algorithms mostly do not point to the actual flaw in the RBox
- How to guide the modeller how to revise the ontology once a flaw is found?

Preliminaries (1/2)—OWL 2/$\mathcal{SROIQ}$

- "basic form" for sub-properties, i.e., $S \sqsubseteq R$,
- "complex form" with property chains
- $R \sqsubseteq C_1 \times C_2$ as shortcut for domain and range axioms $\exists R \sqsubseteq C_1$ and $\exists R^- \sqsubseteq C_2$ where C_1 and C_2 are generic classes; `ObjectPropertyDomain(OPE CE)` and `ObjectPropertyRange(OPE CE)` in OWL.
- $R \sqsubseteq \top \times \top$ when no domain and range axiom has been declared

Definition (User-defined Domain and Range Classes)

Let R be an OWL object property and $R \sqsubseteq C_1 \times C_2$ its associated domain and range axiom. Then, with the symbol D_R we indicate the *User-defined Domain* of R—i.e., $D_R = C_1$—and with the symbol R_R we indicate the *User-defined Range* of R—i.e., $R_R = C_2$.

Preliminaries (2/2)—OWL 2/$\mathcal{SROIQ}$

Definition ((Regular) Role Inclusion Axioms (Horrocks et al, 2006))

Let $\prec$ be a regular order on roles. A **role inclusion axiom** (RIA for short) is an expression of the form $w \sqsubseteq R$, where w is a finite string of roles not including the universal role U, and $R \neq U$ is a role name. A **role hierarchy** $\mathcal{R}_h$ is a finite set of RIAs. An interpretation $\mathcal{I}$ **satisfies** a role inclusion axiom $w \sqsubseteq R$, written $\mathcal{I} \models w \sqsubseteq R$, if $w^{\mathcal{I}} \subseteq R^{\mathcal{I}}$. An interpretation is a **model** of a role hierarchy $\mathcal{R}_h$ if it satisfies all RIAs in $\mathcal{R}_h$, written $\mathcal{I} \models \mathcal{R}_h$. A RIA $w \sqsubseteq R$ is $\prec$**-regular** if R is a role name, and

- $w = R \circ R$, or
- $w = R^-$, or
- $w = S_1 \circ \ldots \circ S_n$ and $S_i \prec R$, for all $1 \geq i \geq n$, or
- $w = R \circ S_1 \circ \ldots \circ S_n$ and $S_i \prec R$, for all $1 \geq i \geq n$, or
- $w = S_1 \circ \ldots \circ S_n \circ R$ and $S_i \prec R$, for all $1 \geq i \geq n$.

Object sub-properties

- Given $S \sqsubseteq R$, then all individuals in the property assertions involving property S must also be related to each other through property R.
- Subsumption for OWL object properties (DL roles) holds if the subsumed property is more constrained such that in every model, the set of individual property assertions is a subset of those of its parent property

Object sub-properties

- Given $S \sqsubseteq R$, then all individuals in the property assertions involving property S must also be related to each other through property R.
- Subsumption for OWL object properties (DL roles) holds if the subsumed property is more constrained such that in every model, the set of individual property assertions is a subset of those of its parent property
- Two ways to constrain a property, and either one suffices:
    - By specifying its domain or range
    - By declaring the property's characteristics

Constraining a property

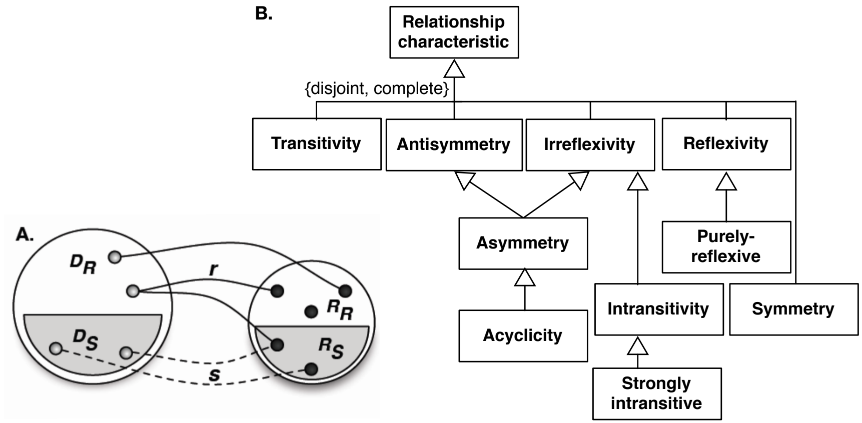


Figure: A: Example, alike the so-called 'subsetting' idea in UML; B: hierarchy of property characteristics (Based on Halpin 2001)

*

Constraining a property

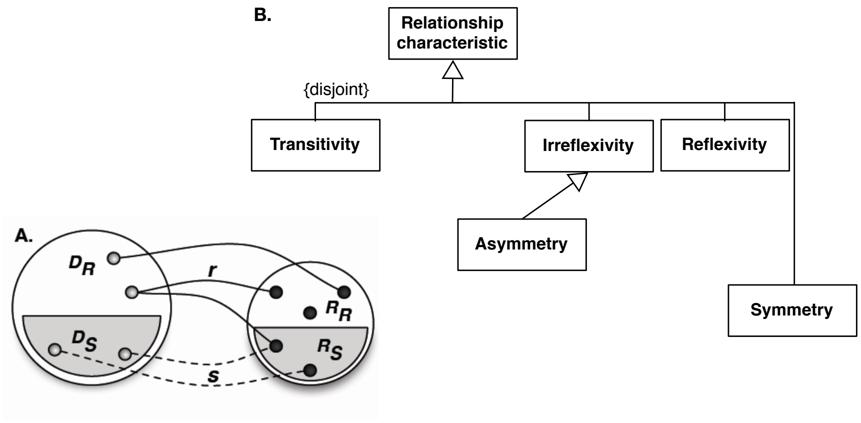


Figure: A: Example, alike the so-called ‘subsetting’ idea in UML; B: hierarchy of property characteristics relevant for OWL 2.

Outline Sub-Property compatibility Service

- First part extends the basic notions from the *RBox compatibility* (defined for $\mathcal{ALCQI}$)
- Informally, it first checks the 'compatibility' of domain and range axioms w.r.t the object property hierarchy and the class hierarchy.

Outline Sub-Property compatibility Service

- First part extends the basic notions from the *RBox compatibility* (defined for $\mathcal{ALCQI}$)
- Informally, it first checks the 'compatibility' of domain and range axioms w.r.t the object property hierarchy and the class hierarchy.
- After that, *SubProS* checks whether the object property characteristic(s) conform to specification, provided there is such an expression in the ontology.
- It exhaustively checks each permutation of domain and range and then of the characteristic of the parent and child property in the object property hierarchy

Definition (Sub-Property compatibility Service (*SubProS*))

For each pair of object properties, $R, S \in \mathcal{O}$ such that $\mathcal{O} \models S \sqsubseteq R$, and $\mathcal{O}$ an OWL ontology adhering to the syntax and semantics as specified in OWL 2 Standard, check whether:

Test 1. $\mathcal{O} \models D_S \sqsubseteq D_R$ and $\mathcal{O} \models R_S \sqsubseteq R_R$;

Test 2. $\mathcal{O} \not\models D_R \sqsubseteq D_S$;

Test 3. $\mathcal{O} \not\models R_R \sqsubseteq R_S$;

Test 4. If $\mathcal{O} \models \text{Asym}(R)$ then $\mathcal{O} \models \text{Asym}(S)$;

Test 5. If $\mathcal{O} \models \text{Sym}(R)$ then $\mathcal{O} \models \text{Sym}(S)$ or $\mathcal{O} \models \text{Asym}(S)$;

Test 6. If $\mathcal{O} \models \text{Trans}(R)$ then $\mathcal{O} \models \text{Trans}(S)$;

Test 7. If $\mathcal{O} \models \text{Ref}(R)$ then $\mathcal{O} \models \text{Ref}(S)$ or $\mathcal{O} \models \text{Irr}(S)$;

Test 8. If $\mathcal{O} \models \text{Irr}(R)$ then $\mathcal{O} \models \text{Irr}(S)$ or $\mathcal{O} \models \text{Asym}(S)$;

Test 9. If $\mathcal{O} \models \text{Asym}(R)$ then $\mathcal{O} \not\models \text{Sym}(S)$;

Test 10. If $\mathcal{O} \models \text{Irr}(R)$ then $\mathcal{O} \not\models \text{Ref}(S)$;

continues....

Definition (Sub-Property compatibility Service (*SubProS*))

... *continued from previous page*

Test 11. If $\mathcal{O} \models \mathsf{Trans}(R)$ then $\mathcal{O} \not\models \mathsf{Irr}(R)$, $\mathcal{O} \not\models \mathsf{Asym}(R)$, $\mathcal{O} \not\models \mathsf{Irr}(S)$, and $\mathcal{O} \not\models \mathsf{Asym}(S)$;

An OWL object property hierarchy is said to be compatible iff

- *Test 1* and (*2* or *3*) hold for all pairs of property-subproperty in $\mathcal{O}$, and
- *Tests 4–11* hold for all pairs of property-subproperty in $\mathcal{O}$.

What to do if not compatible

- Guidelines for fixing a flaw, with one or more options for revision
    - "raising a warning" denotes that it is not a logical error but an ontological one
    - "forcing" a revision indicates there is a logical error that must be fixed in order to have a consistent ontology with satisfiable classes
    - "propose" indicates suggestions how the flaw can be best revised

Revisions (selection)

A. If `Test 1` fails, raise a warning "domain and range restrictions of either R or S are in conflict with the property hierarchy", and propose to

- Change the object property hierarchy, i.e., either remove $S \sqsubseteq R$ and add $R \sqsubseteq S$ or add $S \equiv R$ to $\mathcal{O}$, or
- Change domain and range restrictions of R and/or S, or
- If the test on the domains fails, then propose a new axiom $R \sqsubseteq D'_R \times R_R$, where $D'_R \equiv D_R \sqcap D_S$ (and similarly when `Test 1` fails on the range).

B. ...

C. Run *SubProS* again if any changes have been made in steps A or B, and record changes in the hierarchy (to be used in step I).

Revisions (selection)

...

F. If Irr(R) and `Test 8` fails to detect either Irr(S) or Asym(S), raise a warning "R is irreflexive, hence S should be either Irr(S) or Asym(S), and propose:

- Add Asym(S) or Irr(S) to obtain expected inferences;
- Remove Irr(R);
- Change the positions of R and/or S in the object property hierarchy;

...

I. Run *SubProS* again if any changes have been made in steps D-H, and check any changes in the property hierarchy made against those recorded in step C. If a change from steps E or F reverts a recorded change, then report "unresolvable conflict on subproperty axiom. You *must* change at least one axiom to exit an otherwise infinite loop of swapping two expressions".

BioTop's inconsistent 'has process role'

'has process role' in BioTop (v. June 17, 2010) is inconsistent.
Relevant axioms are:

$$\text{'has process role'} \sqsubseteq \text{'temporally related to'} \tag{E.1}$$

$$\text{'has process role'} \sqsubseteq \text{'processual entity'} \times \text{role} \tag{E.2}$$

$$\text{'temporally related to'} \sqsubseteq \text{'processual entity'} \sqcup \text{quality} \times \text{'processual entity'} \sqcup \text{quality} \tag{E.3}$$

$$\text{role} \sqsubseteq \neg\text{quality} \tag{E.4}$$

$$\text{role} \sqsubseteq \neg\text{'processual entity'} \tag{E.5}$$

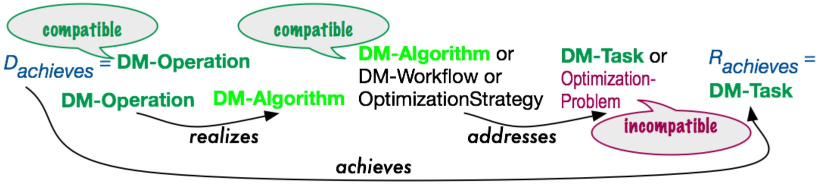
$$\text{Sym}(\text{'temporally related to'}) \tag{E.6}$$

BioTop's inconsistent 'has process role'

Use *SubProS* to isolate the flaw:

- `Test 1`: fail, because $R_{\text{hasprocessrole}} \sqsubseteq R_{\text{temporallyrelatedto}}$ is false, as the ranges (see E.2 cf. E.3) are disjoint (see E.4, E.5) and therewith 'has process role' is inconsistent;
- `Test 2` and 3: pass.
- `Test 4`: not applicable.
- `Test 5`: fail, because $\mathcal{O}$ does not contain Sym('has process role').
- `Test 6–11`: not applicable.

Similar for chains (*ProChainS*); ex.: DMOP chain in v5.2


compatible
compatible
$D_{achieves}$ = DM-Operation
DM-Algorithm or DM-Workflow or OptimizationStrategy
DM-Task or Optimization-Problem
$R_{achieves}$ = DM-Task
DM-Operation
DM-Algorithm
realizes
addresses
incompatible
achieves

Of type *Case S*. `Test S-c` (for corrections) failed because $\mathcal{O} \not\models R_{\text{DM-Task}\sqcup\text{OptimizationProblem}} \sqsubseteq R_{\text{DM-Task}}$. Considering the suggestions for revision, step B's first option to revise the ontology was chosen, i.e., removing OptimizationProblem from the range axiom of addresses.

Outline

1. Methodologies and tools
   - Macro-level methodologies
   - Micro-level methodologies
2. Methods
   - Logic-based debugging
   - Logic & philosophy
3. Methods - philosophy
4. Parameters and dependencies

OntoClean overview (extra, for those interested)

- Problem: messy taxonomies on what subsumes what
- How to put them in the right order?

OntoClean overview (extra, for those interested)

- Problem: messy taxonomies on what subsumes what
- How to put them in the right order?
- OntoClean provides guidelines for this (see to Guarino & Welty, 2004 for an extended example)
- Based on philosophical principles, such as identity and rigidity

  (see Guarino & Welty's EKAW'00 and ECAI'00 papers for more information on the basics)

Basics

- A property of an entity is *essential* to that entity if it must be true of it in every possible world, i.e. if it necessarily holds for that entity.
- Special form of essentiality is *rigidity*

Definition (+R)

A *rigid* property ϕ is a property that is essential to *all* its instances, i.e., $\forall x \phi(x) \rightarrow \Box \phi(x)$.

Definition (-R)

A *non-rigid* property ϕ is a property that is not essential to *some* of its instances, i.e., $\exists x \phi(x) \land \neg \Box \phi(x)$.

Basics

Definition (∼R)

An *anti-rigid* property ϕ is a property that is not essential to *all* its instances, i.e., $\forall x \phi(x) \rightarrow \neg \Box \phi(x)$.

Definition (¬R)

A *semi-rigid* property ϕ is a property that is non-rigid but not anti-rigid.

- Anti-rigid properties cannot subsume rigid properties

Basics

- *Identity*: being able to recognize individual entities in the world as being the same (or different)

Basics

- *Identity*: being able to recognize individual entities in the world as being the same (or different)
- *Unity*: being able to recognize all the parts that form an individual entity; e.g., ocean carries unity (+U), legal agent carries no unity (-U), and amount of water carries anti-unity ("not necessarily wholes", $\sim$U)

Basics

- *Identity*: being able to recognize individual entities in the world as being the same (or different)
- *Unity*: being able to recognize all the parts that form an individual entity; e.g., ocean carries unity (+U), legal agent carries no unity (-U), and amount of water carries anti-unity ("not necessarily wholes", $\sim$U)
- *Identity criteria* are the criteria we use to answer questions like, "is that my dog?"

Basics

- *Identity*: being able to recognize individual entities in the world as being the same (or different)
- *Unity*: being able to recognize all the parts that form an individual entity; e.g., ocean carries unity (+U), legal agent carries no unity (-U), and amount of water carries anti-unity ("not necessarily wholes", $\sim$U)
- *Identity criteria* are the criteria we use to answer questions like, "is that my dog?"
- Identity criteria are conditions used to determine equality (sufficient conditions) and that are entailed by equality (necessary conditions)

Basics

Definition

A non-rigid property carries an IC Γ iff it is subsumed by a rigid property carrying Γ.

Definition

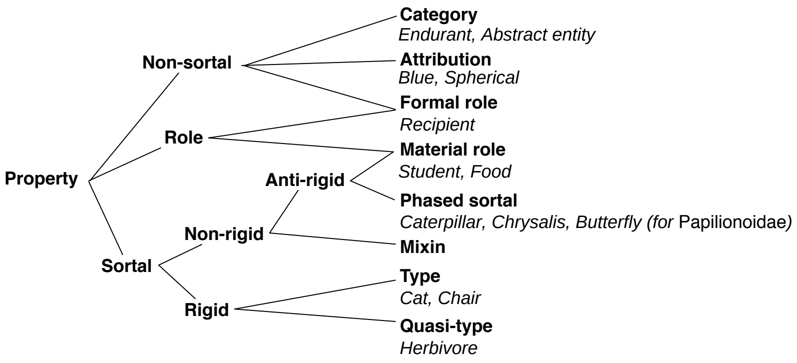
A property ϕ supplies an IC Γ iff i) it is rigid; ii) it carries Γ; and iii) Γ is not carried by all the properties subsuming ϕ. This means that, if ϕ inherits different (but compatible) ICs from multiple properties, it still counts as supplying an IC.

- Any property carrying an IC: +I (-I otherwise).
- Any property supplying an IC: +O (-O otherwise); "O" is a mnemonic for "own identity"
- +O implies +I and +R

Formal ontological property classifications

+O	+I	+R	+D	Type	Sortal
			-D		
-O	+I	+R	+D	Quasi-Type	
			-D		
-O	+I	~R	+D	Material role	
-O	+I	~R	-D	Phased sortal	
-O	+I	¬R	+D	Mixin	
			-D		
-O	-I	+R	+D	Category	Non-Sortal
			-D		
-O	-I	~R	+D	Formal role	
-O	-I	~R	-D	Attribution	
		¬R	+D		
			-D		

Formal ontological property classifications



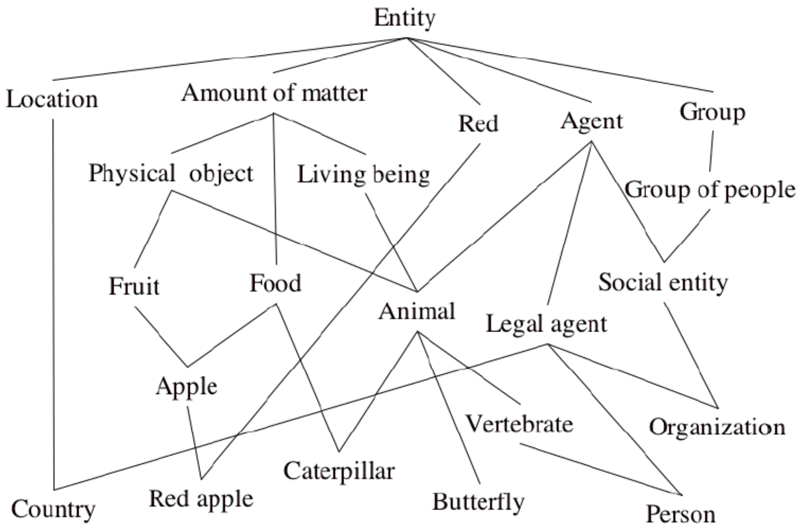
Basic rules

- Given two properties, p and q, when q subsumes p the following constraints hold:
  1. If q is anti-rigid, then p must be anti-rigid
  2. If q carries an IC, then p must carry the same IC
  3. If q carries a UC, then p must carry the same UC
  4. If q has anti-unity, then p must also have anti-unity

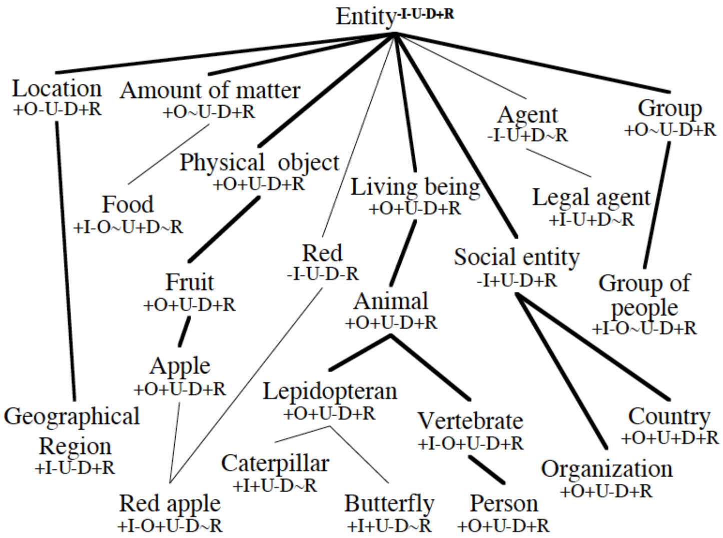
5. Incompatible IC's are disjoint, and Incompatible UC's are disjoint

- And, in shorthand:
  6. $+R \not\subset \sim R$
  7. $-I \not\subset +I$
  8. $-U \not\subset +U$
  9. $+U \not\subset \sim U$
  10. $-D \not\subset +D$

Example: before



Example: after



Outline

The changing landscape

- Solving the early-adopter issues has moved the goal-posts, and uncovered new issues; e.g.:
  - Which ontologies are reusable for one's own ontology, in whole or in part?
  - What are the consequences of choosing one ontology over the other?
  - OWL 2 has 5 languages: which one should be used for what and when?
  - Internationalisation/localisation of ontologies?
- Note: Next slides are a partial recap of topics that have been mentioned before, but now presented in a different way

Parameters

- Which parameters affect ontology development?
  Where?
  How?

Parameters

- Which parameters affect ontology development?
  Where?
  How?
- We consider:
  - Purposes
  - Reusing ontologies
  - Bottom-up development
  - Languages
  - Reasoning services

Purposes—examples

- Querying data by means of an ontology (OBDA) through linking databases to an ontology
- Database integration
- Structured controlled vocabulary to link data(base) records and navigate across databases on the Internet ('linked data')
- Using it as part of scientific discourse and advancing research at a faster pace, (including experimental ontologies)
- Coordination among and integration of Web Services

TBC on next page...

Purposes (cont'd)

- Ontology in an ontology-driven information system destined for run-time usage, e.g., in scientific workflows, MASs, ontology-mediated data clustering, and user interaction in e-learning
- Ontologies for NLP, e.g. annotating and querying Digital Libraries and scientific literature, QA systems, and materials for e-learning
- As full-fledged discipline "Ontology (Science)", where an ontology is a formal, logic-based, representation of a scientific theory
- Tutorial ontologies, e.g., the wine and pizza ontologies

Reusing ontologies

- Foundational ontologies
- Reference ontologies
- Domain ontologies that have an overlap with the new ontology;

Reusing ontologies

- Foundational ontologies
- Reference ontologies
- Domain ontologies that have an overlap with the new ontology;
- For each of them, resource usage considerations, such as
    - Availability of the resource (open, copyright)
    - If the source is being maintained or abandoned one-off effort;
    - Community effort, research group, and if it has already some adoption or usage;
    - Subject to standardisation policies or stable releases;
    - If the ontology is available in the desired or required ontology language.

Example

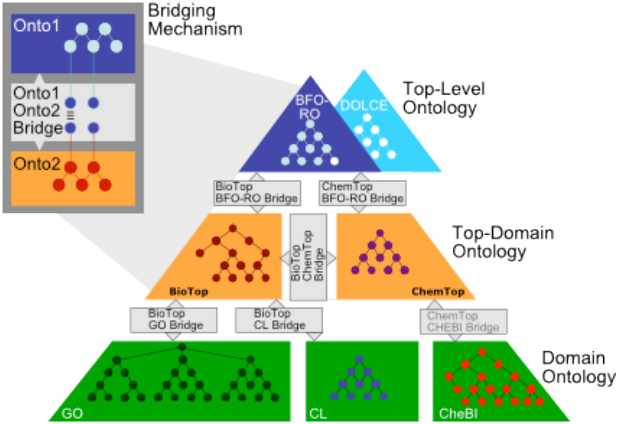


image from http://www.imbi.uni-freiburg.de/ontology/biotop/

Bottom-up development

- Reuse of other knowledge-based representations:
  - conceptual data models (UML diagrams, ER, and ORM)
- Database (and OO) reverse engineering, and least common subsumer and clustering to infer new concepts;
- Abstractions from or formalisations of models in textbooks and diagram-based software;
- Thesauri and other structured vocabularies;
- Other (semi-)structured data, such as spreadsheets and company product catalogs;
- Text mining of documents to find candidate terms for concepts and relations;
- Terminologies, lexicons, and glossaries;
- Wisdom of the crowds tagging, tagging games, and folksonomies;

Languages – preliminary considerations

- Depending on the purpose(s) (and available resources), one ends up with either
  (a) a large but simple ontology, i.e., mostly just a taxonomy without, or very few, properties (relations) linked to the concepts, where 'large' is, roughly, $>$ 10000 concepts, so that a simple representation language suffices;
  (b) a large and elaborate ontology, which includes rich usage of properties, defined concepts, and, roughly, requiring OWL-DL; or
  (c) a small and very complex ontology, where 'small' is, roughly, $<$ 250 concepts, and requiring at least OWL 2 DL

Languages – preliminary considerations

- Depending on the purpose(s) (and available resources), one ends up with either
  - (a) a large but simple ontology, i.e., mostly just a taxonomy without, or very few, properties (relations) linked to the concepts, where 'large' is, roughly, $>$ 10000 concepts, so that a simple representation language suffices;
  - (b) a large and elaborate ontology, which includes rich usage of properties, defined concepts, and, roughly, requiring OWL-DL; or
  - (c) a small and very complex ontology, where 'small' is, roughly, $<$ 250 concepts, and requiring at least OWL 2 DL
- Certain choices for reusing ontologies or legacy material, or goal, may lock one a language
- $\Rightarrow$ Separate dimension that interferes with the previous parameters: the choice for a representation language

Languages

- Older KR languages (frames, obo, conceptual graphs, etc.)

Languages

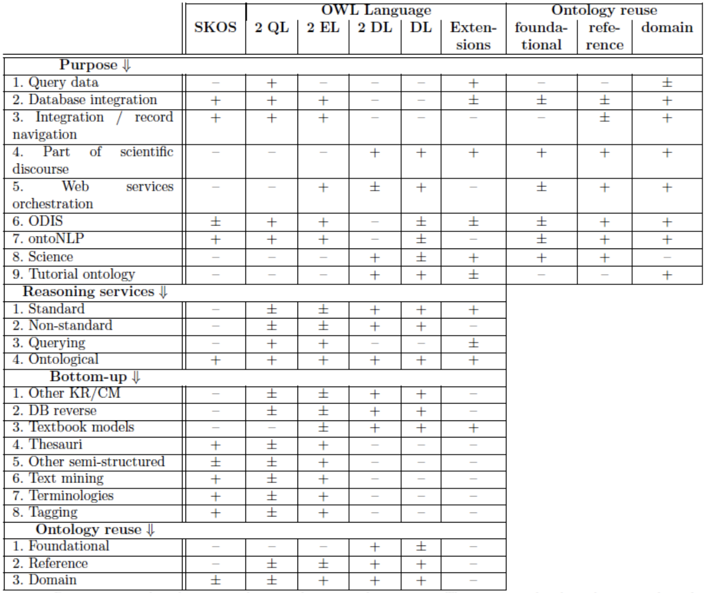
- Older KR languages (frames, obo, conceptual graphs, etc.)
- Web Ontology Languages:
  - OWL: OWL-Lite, OWL-DL, OWL full
  - OWL 2 with 4 languages to tailor the choice of ontology language to fit best with the usage scope in the context of a *scalable* and *multi-purpose* SW:
    - OWL 2 DL, based on the DL language $\mathcal{SROIQ}$
    - OWL 2 EL
    - OWL 2 QL
    - OWL 2 RL

Languages

- Older KR languages (frames, obo, conceptual graphs, etc.)
- Web Ontology Languages:
  - OWL: OWL-Lite, OWL-DL, OWL full
  - OWL 2 with 4 languages to tailor the choice of ontology language to fit best with the usage scope in the context of a *scalable* and *multi-purpose* SW:
    - OWL 2 DL, based on the DL language $\mathcal{SROIQ}$
    - OWL 2 EL
    - OWL 2 QL
    - OWL 2 RL
- Extensions (probabilistic, fuzzy, temporal, etc.)
- Alternatives (e.g., Common Logic)
- Differences between expressiveness of the ontology languages and their trade-offs

Reasoning services

- Description logics-based reasoning services
  - The standard reasoning services for ontology usage: satisfiability and consistency checking, taxonomic classification, instance classification;
  - 'Non-standard' reasoning services to facilitate ontology development: explanation/justification, glass-box reasoning, pin-pointing errors, least-common subsumer;
  - Querying functionalities, such as epistemic and (unions of) conjunctive queries;
- Ontological reasoning services (OntoClean, RBox reasoning service)
- Other technologies (e.g., Bayesian networks)



		OWL Language					Ontology reuse		
	SKOS	2 QL	2 EL	2 DL	DL	Extensions	foundational	reference	domain
Purpose ⇓									
1. Query data	−	+	−	−	−	+	−	−	±
2. Database integration	+	+	+	−	−	±	±	±	+
3. Integration / record navigation	+	+	+	−	−	−	−	±	+
4. Part of scientific discourse	−	−	−	+	+	+	+	+	+
5. Web services orchestration	−	−	+	±	+	−	±	+	+
6. ODIS	±	+	+	−	±	±	±	+	+
7. ontoNLP	+	+	+	−	±	−	±	+	+
8. Science	−	−	−	+	±	+	+	+	−
9. Tutorial ontology	−	−	−	+	+	±	−	−	+
Reasoning services ⇓									
1. Standard	−	±	±	+	+	+			
2. Non-standard	−	±	±	+	+	−			
3. Querying	−	+	+	−	−	±			
4. Ontological	+	+	+	+	+	+			
Bottom-up ⇓									
1. Other KR/CM	−	±	±	+	+	−			
2. DB reverse	−	±	±	+	+	−			
3. Textbook models	−	−	±	+	+	+			
4. Thesauri	+	±	+	−	−	−			
5. Other semi-structured	±	±	+	−	−	−			
6. Text mining	+	±	+	−	−	−			
7. Terminologies	+	±	+	−	−	−			
8. Tagging	+	±	+	−	−	−			
Ontology reuse ⇓									
1. Foundational	−	−	−	+	±	−			
2. Reference	−	±	±	+	+	−			
3. Domain	±	±	+	+	+	−			

Table 1 Basic cross-matching between realistic combinations of parameters. The more complex dependencies, such as the interaction between purpose, language, and reasoning service, can be obtained from traversing the table (*purpose* ↔

Tools

- Thus far, no tool gives you everything

Tools

- Thus far, no tool gives you everything
- Software-supported methodologies
- Ontology Development Environment (ODE)
- Software-supported methods and other features
- Portals
- Exporting ontologies for a different rendering (visualisation, documentation, ...)

Tools: Develop your own tool

- How to do that?

Tools: Develop your own tool

- How to do that?
- There are tools and APIs etc that help you manipulate OWL files and manage reasoners
    - Java: OWL API, OWLink, or Apache Jena
    - Python: Owlready

Summary