Ontology Engineering

Based on work of Daniel Schober, Grigoris Antoniou, Frank von Harmelen, and Yuri Tijerino
Quick summary of the previous lecture

• OWL2
  – Syntactic Sugar:
    • Disjoint Union/Classes, Negative Property Assertions
  – New properties
    • Self restriction, Qualified Cardinality Restriction, Property Chains, Keys, Disjoint Properties
  – Three tractable fragments
    • OWL EL
    • OWL QL
    • OWL RL
What is an Ontology?

• An ontology is a formal specification of a commonly agreed conceptualization
  – Tom Gruber (1992)

• Usually a taxonomy of classes enriched with relations
Ontology Engineering: Simple to komplex
Getting started

Start with a simple and traceable domain

• TASK:
  – Let us develop ”The Ontology of Cutlery”
Naive start: Cutlery Ontology
Overwhelming quantity of trivial stuff ...
So, ... create a 'Cutlery Ontology'

This is easy...
starting out with a simple taxonomy ...

• Cutlery
  – Knife
    • Bread Knife
    • Butter Knife
  – Fork
  – Spoon
    • Table Spoon
    • Tea Spoon

... easy, but are we done yet?
... surprise ... !

A Spoon & Fork chimaera ... a Spork!

- From Mono to Bi-Parenthood
- Spoon & Fork can’t be disjoint any longer
- Need to introduce intersections
Knork? A Knife & Fork

These weird examples show there are always exceptions, which might later turn out to be the rule

→ Lack of domain knowledge in even the simplest areas
→ Hard to decide when you’re ‘complete’
... and then ... a 'Splayd‘

Should be classified under Spoon, Fork and Knife

→ Multiple Parenthood (Tri-Parenthood)
→ Asserting all parents will blow up the hierarchy & impair navigation
→ Untangling (let reasoner maintain polyhierarchy)
From the simple to the complex

- At the end this is still naive ...
- Often there are more ‘exceptions’ than classic (cutlery) examples!
Complexity to be found everywhere

• ... even in domains envisioned as simple & self-sufficient
  – Combinatory explosion
  → Introduce Processes or Functions
Need for a formal "Ontology Engineering" process
Software engineering versus programming

• Software engineering is about more than just programming/coding
• It is about design principles and methodologies that yield programs that are
  – Robust
  – Manageable
  – Reusable
Software life-cycle

- Software development stages
  - Specification
  - Design
  - Implementation
  - Integration
  - Validation
  - Operation/Maintenance/Evolution
- Different types of system organise these generic activities in different ways
- *Waterfall approach* treats them as distinct stages to be signed off chronologically
- In practice usually an iteration of various steps
Despite of sharing a certain quantity of topics, Software Engineering (SE) and Ontological Engineering (OE) communities have been working separately.

- Is there room for effort integration?
- What are the learned lessons?
- How can each part profit from this union?
Scenario 1

Apply SE for Ontology development
(lifecycle, metrics, modeling languages, etc.)

Ontological Engineering
Scenario 2

Adoption of Ontologies to improve the SE processes (support during the lifecycle, execution infra-structure)

(Ontology Based) Software Engineering
Collaborations

Ontologies

May support the Development of new SE approaches

Software Engineering

Inspirations for the OE development and maturing
The importance of names and precise language
Syntax Variations in Class Names

KitchenKnife, kitchen knife, kitchenKnife, KitKn, KitchenKnifeClass, Kitchen knifes, Kitchen_Knife, KITCHEN-KNIFE, i-Knife™, κοπίς, 魚切

Variation in Case, Separator, Typography, Encoding, Explicitness … → Naming conventions
Multicultural settings even more tricky
Multilingual settings even more tricky

- “Everything is great!”
- “You are worthless!” (zero)
- “You are an indelicate sphincter!”
- “Your car needs new piston rings!”
- Any of the above, somewhere
Multilingual settings even more tricky

Cocktail lounge, Norway:
  LADIES ARE REQUESTED NOT TO HAVE CHILDREN IN THE BAR.

Airline ticket office, Copenhagen:
  WE TAKE YOUR BAGS AND SEND THEM IN ALL DIRECTIONS.

Hotel, Yugoslavia:
  THE FLATTENING OF UNDERWEAR WITH PLEASURE IS THE JOB OF THE CHAMBERMAID.
Multilingual settings even more tricky

• As of June 2010, 72.6% of the almost 2 billion Web users, the ultimate audience for the Semantic web, speak a language other than English.
Benefits of Consistent Naming

• Increase consistency, accuracy & clarity of labels
  • Normalize appearance
• Reduce diversity with which meta-tools have to cope with
  • Ease text mining
  • Ease ontology mapping & alignment
    • E.g. PROMPT or Lexical Owl Ontology Mapper (LOOM)
Good starting point: Wordnet 3.1

- A lexical database for English
- Can be seen as a large taxonomy over terms
- All terms are related to
  - Synonyms (equivalent classes)
  - Hyponyms (sub classes)
  - Hypernyms (super classes)

- [http://wordnet.princeton.edu/](http://wordnet.princeton.edu/)
- [http://www.w3.org/TR/wordnet-rdf/](http://www.w3.org/TR/wordnet-rdf/)
# OBO Foundry Naming Conventions

<table>
<thead>
<tr>
<th>Naming Convention</th>
<th>Description</th>
<th>Example</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1. Be clear and unambiguous</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use explicit and concise names</td>
<td>Keep names short and memorable, but precise enough to capture the intended meaning. Keep names linguistically correct and intuitively meaningful to human readers. Articles should be omitted.</td>
<td>‘wall of esophagus’, ‘physical part’ instead of ‘the wall of the esophagus’, ‘distinct identifiable physical part’</td>
<td>Faster term recognition</td>
</tr>
<tr>
<td>Use context independent names</td>
<td>Apply names that are self-explanatory and understandable even when viewed outside of the immediate context of the ontology. Avoid truncated names and colloquialisms. In names, capture inherent and intrinsic characteristics rather than asserted and extrinsic characteristics. Avoid using names for non-role entities that refer to roles the entity referred to may potentially play in a particular context at a particular time. Capture product names as they are, but render them intelligible adding contextual information: [company name] + [product name] + [product type] (usually the superclass name). Additional information like the legal status of a company (e.g. Corp. or Inc.) should be omitted.</td>
<td>‘NMR magnet’ ‘chemotherapy’ and ‘1ml pipette tip’ instead of ‘magnet’, ‘chemo’ and ‘blue pipette tip’</td>
<td>Increases precision in the interpreted meaning. Helps string matching. Faster term recognition</td>
</tr>
<tr>
<td>Avoid taboo words</td>
<td>Affixes reflecting epistemological claims e.g., words that indicate types of representational units should be avoided in name.</td>
<td>‘protocol’ instead of ‘protocol class’ or ‘protocol type’</td>
<td>Faster term recognition, redundancy reduction</td>
</tr>
<tr>
<td>Avoid encoding administrative metadata in names</td>
<td>Administrative metadata, e.g., a class’ status and version should be factored out of the name and into suitable separate representational units</td>
<td>‘protocol’ instead of ‘protocol (definition incomplete)’</td>
<td>Increases precision in the interpreted meaning</td>
</tr>
<tr>
<td>2. Be univocous</td>
<td>Names should have the same meaning on every occasion of use and refer to the same types of entities in reality. Homonyms, ambiguous terms that share the same spelling but have many different meanings, are to be avoided as part of editor-preferred names. Use terms with fewest possible amount of homonyms in building names.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Avoid conjunct ions</td>
<td>Words that are used to join other words, such as the logical connectives ‘and’ and ‘or’ should be avoided in names as they can introduce ambiguity and may hamper inference by causing excessive branching. The same applies to qualifiers such as ‘in some cases’. In ‘anatomic structure, system or substance’ it is not clear whether the adjective “anatomic” is restricted to “structure” or extends also to “system and substance”. In the first case the substances “drug” and “chemical” would be classified under this class, otherwise not.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prefer singular nominal form</td>
<td>Use singular names throughout. Where plurals need to be captured, e.g. when one instance of the plural class represents a plurality itself, consistently use explicit plural indicating postfixes as part of the class names, e.g. use ‘aggregate’, ‘collective’ or ‘population’ consistently, but only as applicable.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use positive names</td>
<td>Avoid use of negations in formulating names. Avoid complements and negative names like ‘non-separation device’ because logically this will include everything in the universe that is not a separation device. The absence of a characteristic is not a concise differentiating criterion. Do not represent the absence of a characteristic (e.g. wing) as the presence of the non-existence of a characteristic, e.g.: ‘wing has status “absent”’. ‘pair of lungs’, ‘population’ instead of ‘lungs’, ‘people collection’</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Avoid catch-all terms</td>
<td>Avoid ‘rag-bag’ words that do not designate natural kinds. The existence of classes is not dependent on our biological knowledge. Avoid ‘unlocalised’, ‘unknown’, ‘unclassified’</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>‘protocol collection’ instead of ‘protocol set’ for a plurality of protocols (store the latter as synonym), ‘parameter adjustment’ instead of ‘protocol set’ for the act of setting parameters</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Increases precision in the interpreted meaning. Faster term recognition</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Increases precision in the interpreted meaning</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Increases precision in the interpreted meaning, helps string matching</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Increases precision in the interpreted meaning</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Recycle strings</td>
<td>Word compositions should be constructed in a consistent manner, rather than using para-synonymous strings interchangeably. When creating compound names re-use strings as they occur in names of entities already defined elsewhere in this or in other ontologies.</td>
<td>‘x part of process’, ‘y part of process’ instead of ‘x component of process’, ‘y portion of process’</td>
<td>Helps matching</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Use genus-differentia style names</td>
<td>Class names should reflect the differentia that distinguish the class from its parent class (modifiers to the head word). These should be the same that are modelled explicitly, so that the name compounds can be mapped to representational units that are connected to that class.</td>
<td>‘DNA-microarray’ is a ‘microarray’ ‘protein-microarray’ is a ‘microarray’, where ‘DNA’ and ‘protein’ are defined elsewhere.</td>
<td>Eases cross products generation. Helps string matching.</td>
</tr>
<tr>
<td>Use space as word separators</td>
<td>Use the bar space (_) character as word separator, just as it would normally appear in the language of choice. Where use of the bar space is not allowed by the type of representational unit in use to store a name, the underscore (_) should be used instead. Camel case should not be used as a means of word separation.</td>
<td>‘DNA microarray’, ‘pH value’ instead of ‘DNA microarray’, ‘pHValue’</td>
<td>Faster term recognition. Helps string matching.</td>
</tr>
<tr>
<td>Expand abbreviations and acronyms</td>
<td>Spell out abbreviations and acronyms and capture truncated versions as synonyms. Acronyms that result in expressions that have other meanings should be avoided. Widely known acronyms (anacronyms) such as DNA and LASER can be used.</td>
<td>‘high resolution probe’ instead of ‘HRP’ or ‘high res. probe.’</td>
<td>Faster term recognition, increases precision in the interpreted meaning. Helps string matching.</td>
</tr>
<tr>
<td>4. Typography</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use lower case beginnings</td>
<td>Don’t enforce dogmatically, but prefer lower case beginnings for class and property names. Capture names just as they would appear in normal English written text, i.e. where acronyms and proper nouns cannot be avoided in names they should be capitalized.</td>
<td>Use ‘microarray’, ‘DNA microarray’, ‘pH value’, ‘Golgi apparatus’</td>
<td>Faster term recognition.</td>
</tr>
<tr>
<td>Avoid character formatting</td>
<td>Use plain ASCII format to keep names as computationally plat as possible. Subscripts, superscripts and accents should be avoided.</td>
<td>‘SIGMA-ALDRICH’ instead of ‘Σ-ALDRICH™’</td>
<td>Helps string matching.</td>
</tr>
</tbody>
</table>
Formal process of ontology development

1. Constructing new ontologies manually
2. Reusing existing ontologies
3. Ontology mapping
Constructing ontologies manually
Main Stages in Manual Ontology Development

1. Determine scope
2. Consider reuse
3. Enumerate terms
4. Define taxonomy
5. Define properties
6. Define constraints
7. Add instances
8. Check for anomalies

Not a linear process!
Determine Scope

• There is no correct ontology of a specific domain
  – Difference to software engineering!
  – An ontology is an abstraction of a particular domain, and there are always viable alternatives

• What is included in this abstraction should be determined by
  – the use to which the ontology will be put
  – by future extensions that are already anticipated
Determine Scope (2)

• Basic questions to be answered at this stage are:
  – What is the domain that the ontology will cover?
  – For what we are going to use the ontology?
  – For what types of questions should the ontology provide answers?
  – Who will use and maintain the ontology?
Consider Reuse

• With the spreading deployment of the Semantic Web, ontologies will become more widely available

• We rarely have to start from scratch when defining an ontology
  – There is almost always an ontology available from a third party that provides at least a useful starting point for our own ontology
  – Use Sindice.com for searching ontologies
Enumerate Terms

• Write down in an unstructured list all the relevant terms that are expected to appear in the ontology
  – Nouns form the basis for class names
  – Verbs (or verb phrases) form the basis for property names

• Traditional knowledge engineering tools (e.g. laddering and grid analysis) can be used to obtain
  – the set of terms
  – an initial structure for these terms

• Task: start to model an animals & plant ontology
Example: animals & plants ontology

- Dog
- Cat
- Cow
- Person
- Tree
- Grass
- Herbivore
- Male
- Female
- Carnivore
- Plant
- Animal
- Fur
- Child
- Parent
- Mother
- Father
- Living Thing
- Dangerous
- Pet
- Domestic Animal
- Farm animal
- Draft animal
- Food animal
- Fish
- Carp
- Goldfish
Define Taxonomy

• Relevant terms must be organized in a taxonomic hierarchy
  – Opinions differ on whether it is more efficient/reliable to do this in a top-down or a bottom-up fashion

• Ensure that hierarchy is indeed a taxonomy:
  – If A is a subclass of B, then every instance of A must also be an instance of B (compatible with semantics of `rdfs:subClassOf`)

• Task: how about our animals&plant ontology?
Example: Animals & Plants

- Dog
- Cat
- Cow
- Person
- Tree
- Grass
- Herbivore
- Male
- Female

- Carnivore
- Plant
- Animal
- Fur
- Child
- Parent
- Mother
- Father
- Living Thing

- Healthy
- Pet
- Domestic Animal
- Farm animal
- Draft animal
- Food animal
- Fish
- Carp
- Goldfish
Example: Animals & Plants

- **Living Thing**
  - Animal
    - Mammal
      - Cat
      - Dog
      - Cow
      - Person
    - Fish
      - Carp
      - Goldfish
  - Plant
    - Tree
    - Grass
    - Fruit

- **Modifiers**
  - domestic
    - pet
    - Farmed
      - Draft
      - Food
  - Wild
  - Health
    - healthy
    - sick
  - Sex
    - Male
    - Female
  - Age
    - Adult
    - Child

- **Relations**
  - eats
  - owns
  - parent-of
  - ...  

- **Definable**
  - Carinvore
  - Herbivore
  - Child
  - Parent
  - Mother
  - Father
  - Food Animal
  - Draft Animal
Define Properties

• Often interleaved with the previous step
• The semantics of subClassOf demands that whenever A is a subclass of B, every property statement that holds for instances of B must also apply to instances of A
  – It makes sense to attach properties to the highest class in the hierarchy to which they apply
Define Properties (2)

• While attaching properties to classes, it makes sense to immediately provide statements about the domain and range of these properties

• There is a methodological tension here between generality and specificity:
  – Flexibility (inheritance to subclasses)
  – Detection of inconsistencies and misconceptions

• Task: how about animals & plants?
Example: Animals & Plants

• Modifiers
  – Domestication
    • Domestic
    • Wild
  – Use
    • Draft
    • Food
    • pet
  – Risk
    • Dangerous
    • Safe
  – Sex
    • Male
    • Female
  – Age
    • Adult
    • Child

◆ Relations
  ◆ eats
  ◆ owns
  ◆ parent-of
  ◆ ...
Example: Animals & Plants

• **Animal** *eats* Living_thing
  – *eats* domain: Animal;
    range: Living_thing

• **Person** *owns* Living_thing *except* person
  – *owns* domain: Person
    range: Living_thing & not Person

• **Living_thing** *parent_of* Living_thing
  – *parent_of*: domain: Living_thing
    range: Living_thing
Define Facets: From RDFS to OWL

- Cardinality restrictions
- Required values
  - owl:hasValue
  - owl:allValuesFrom
  - owl:someValuesFrom
- Relational characteristics
  - symmetry, transitivity, inverse properties, functional values
- Task: what is left for our animals & plants ontology?
Example: Animals & Plants

• “A ‘Parent’ is an animal that is the parent of some other animal” (Ignore plants for now)
  – Parent = Animal and parent_of some Animal

• “A ‘Herbivore’ is an animal that eats only plants” (NB All animals eat some living thing)
  – Herbivore = Animal and eats only Plant

• “An ‘omnivore’ is an animal that eats both plants and animals”
  – Omnivore = Animal and eats some Animal and eats some Plant
Define Instances

• Filling the ontologies with such instances is a separate step
• Number of instances >> number of classes
• Thus populating an ontology with instances is not done manually
  – Retrieved from legacy data sources (DBs)
  – Extracted automatically from a text corpus
Check for Anomalies

• An important advantage of the use of OWL over RDF Schema is the possibility to detect inconsistencies
  – In ontology or ontology+instances

• Examples of common inconsistencies
  – incompatible domain and range definitions for transitive, symmetric, or inverse properties
  – cardinality properties
  – requirements on property values can conflict with domain and range restrictions

• Task: why is inconsistency such a big deal?
Reusing existing ontologies
Existing Domain-Specific Ontologies

• **Medical domain**: Cancer ontology from the National Cancer Institute in the United States

• **Cultural domain**:
  – Art and Architecture Thesaurus (AAT) with 125,000 terms in the cultural domain
  – Union List of Artist Names (ULAN), with 220,000 entries on artists
  – Iconclass vocabulary of 28,000 terms for describing cultural images

• **Geographical domain**: Getty Thesaurus of Geographic Names (TGN), containing over 1 million entries
Integrated Vocabularies

• Merge independently developed vocabularies into a single large resource

• E.g. Unified Medical Language System integrating 100 biomedical vocabularies
  – The UMLS metathesaurus contains 750,000 concepts, with over 10 million links between them

• The semantics of a resource that integrates many independently developed vocabularies is rather low
  – But very useful in many applications as starting point

• Task: what do you think are major problems with these “super” vocabularies?
Upper-Level Ontologies

• Some attempts have been made to define very generally applicable ontologies
  – Most domain-specific
• Cyc, with 60,000 assertions on 6,000 concepts
• Standard Upperlevel Ontology (SUO)
Topic Hierarchies

• Some “ontologies” do not deserve this name:
  – simply sets of terms, loosely organized in a hierarchy

• This hierarchy is typically not a strict taxonomy but rather mixes different specialization relations (e.g. is-a, part-of, contained-in)

• Such resources often very useful as starting point

• Example: Open Directory hierarchy, containing more then 400,000 hierarchically organized categories and available in RDF format
Some resources were originally built not as abstractions of a particular domain, but rather as linguistic resources. These have been shown to be useful as starting places for ontology development. For example, WordNet.
Ontology Libraries

• Attempts are currently underway to construct online libraries of online ontologies
  – Rarely existing ontologies can be reused without changes
  – Existing concepts and properties must be refined using `rdfs:subClassOf` and `rdfs:subPropertyOf`
  – Alternative names must be introduced which are better suited to the particular domain using `owl:equivalentClass` and `owl:equivalentProperty`
  – We can exploit the fact that RDF and OWL allow private refinements of classes defined in other ontologies

• Sindice.com is one example
Ontology Mapping
Ontology Mapping

• A single ontology will rarely fulfill the needs of a particular application; multiple ontologies will have to be combined

• This raises the problem of ontology integration (also called ontology alignment or ontology mapping)

• Current approaches deploy a whole host of different methods;
  — linguistic, statistical, structural and logical methods
Ontology-Mapping Techniques Conclusion

• Although there is much potential, and indeed need, for these techniques to be deployed for Semantic Web engineering, this is far from a well-understood area

• No off-the-shelf techniques are currently available, and it is not clear that this is likely to change in the near future
Ontology design patterns
Ontology Design Patterns

www.ontologydesignpatterns.org

OPTypes

Ontology Design Pattern types
There are several types of ODPs.

We have identified several types of ODPs, and have grouped them into six families as shown in the figure below.

- Structural ODPs
  - Logical ODPs
  - Architectural ODPs
- Correspondence ODPs
  - Re-engineering ODPs
  - Alignment ODPs
- Content ODPs (CPs)
- Reasoning ODPs
- Presentation ODPs
  - Naming ODPs
  - Annotation ODPs
- Lexico-Syntactic ODPs
Hard and „not so hard“ ontologies
Domains modeled with Ontologies

- A kitchen and its dishes
- Finance markets
- Simple machines
- Human behavior, social interactions
- Metabolic pathways
- The weather
- CPU architectures
- Quantum-molecular properties for NMR analysis

• TASK: Sort these ontology domains based on how hard you think they are to model!
Not all domains are equally suited to be modeled ontologically!

• Easier
  – Simple machines
  – CPU architectures
  – A kitchen and its dishes

• Harder
  – Finance markets
  – Human behavior, social interactions
  – Metabolic pathways
  – Quantum-molecular properties for NMR analysis
  – The weather
*Difficult* Domains

What characterizes hard and untraceable domains?

- **Complexity**
  - Large sum of components
  - Massively connected
  - n:n relationships

- **Non-deterministic nature**
  - Non-linearity
  - Feedback loops
  - Chaotic systems, displaying butterfly effect
  - Synergy, Emergence, Self-organization

→ Typically found in biological organisms, but not exclusively
‘Difficult‘ Domains

– Dynamically changes over time
  • Highly time-resolved
– Unclear borders
  • i.e. when does ‘life’ start? When is ‘death’ completed?
– Analogue, rather than discrete
– Probabilistic and fuzzy, rather than Boolean
– Non-meso domains
  • i.e. macro or micro domain
    – Uncertainty (Heisenberg)
    – Lack of knowledge
    – Distant from daily experiences our brains are trained in

➔ For numeric & quantitative representations use …
– Mathematical models via differential equations
– Probabilistic models
– …
‘Easy‘ Domains

What characterizes simple modeling domains?

• Low complexity
  – Small number of components and their parts
    • Discrete / enumerated list
  – Minor interaction between components
    • 1:1 relationships
  – Few feedback loops
    • Linearity

• Clearly delineated borders between components

• Static over time

• Meso-domain (Not macro-, nor micro-domain)
  – Traceable
    • Directly perceived by our human senses
    • Easy to be verified via our mental representation
Conclusion I

- Not all domains are equally well suited for ontological representations
- Be aware that the world is always more complicated, than you expect.
- Multiple top level ontologies to choose from
  ... and all still under discussion/construction
- Ontologies are never static ... no matter how trivial a domain may seem
  – One single instance can ’ruin’ a whole ontology
  – OE is an iterative optimization process
- Ontologies change over time due to
  – The advance of science
  – Our modeling expertise
  – Changing requirements regarding the application
Conclusion II

• Trend to increase formal expressivity
  – E.g. from simple taxonomy to full DL-EL formal semantics

• With increased granularity & explicitness, definitions get more and more complex and unreadable:
  – A Sporc is_a Spoon AND Fork
  – A Sporc is_a Spoon AND (Cutlery usedFor FoodPickup)
  – A Sporc is_a (Cutlery usedFor LiquidPickup) AND (Cutlery usedFor NonLiquidPickup)

..., add DL quantifiers, cardinalities, closure axioms, metadata, ...
... running into performance problems!
Conclusion III

Formal OE is still largely an art and hard work!

→ Find balance between
  • Formality vs. Readability
  • Traceability vs. Intuition
  • Expressivity vs. Performance
  • Philosophical Realism vs. Pragmatics

„Simplicity is the ultimate sophistication“

Leonardo Da Vinci