Checking Class-Labels against Naming Conventions: Experiences with the OntoCheck Plugin

Daniel Schober, Vojtech Svatek, Martin Boeker
Diversity in class-labels

**OBI Hierarchy**

- `bfo:Entity` entity
- `snap:Continuant` continuant
  - `snap:DependentContinuant` dependent_continuant
    - `snap:GenericallyDependentContinuant` generically_dependent_continuant
  - `obi-231` information entity
    - `obi_0000046` non-realizable information entity
      - `obi_0000005` Conclusion
      - `obi_0000142` Information content entity
        - `obi_0000071` Quantitative Confidence
          - `obi_0000175` P-value
          - `obi_0000266` curation_status
        - `obi_0000261` Digital Entity
          - `obi_0000078` EMedicalRecord
  - `obi_000141` material entity
    - `cell:CL_0000000` cell
    - `chebi:CHEBI_23091` chemical entity
      - `obi_0000065` polyethylene glycol p-(1,1,3,3-tetramethylbutyl)-phenyl ether
      - `obi_0000789` ?vanadium 3³
    - `obi_0000098` _biomaterial(deprecated)
      - `obi_0100010` epstein barr virus transformed b cell
      - `obi_400162` Bronchial_Aveolar_Lavage [BAL]
      - `obi_400164` α amylase
    - `obi_0100000` biomaterial
    - `obi_0100051` entity of organismal origin
  - `obi_0302714` collection
    - `obi_0000164` cola lib
    - `obi_0000181` collection of individuals
  - `obi_0400002` device
    - `obi_0000036` LC_column
    - `obi_0400003` instruments
    - `obi_0400147` microarray
      - `obi_0400148` DesoxyRibonucleicAcidMicroarray
        - `obi_400165` Dual GMO®

**Found Heterogeneities**

- **Identifier**
  - no prefix+string
  - uppercase prefix+underscore+number

- **Negative name**

- **Word separator**
  - underscore
  - camelcase (no separator)
  - space

- **Case**
  - lower case
  - upper camel case

- **Administrative metadata**

- **Proper noun and person name**

- **Synonym**

- **Explicitness**

- **Special character**

- **Abbreviation**

- **Acronym**

- **Singularity**

- **Product name**
Why harmonize labels?

- Increase consistency & clarity
  - Normalize appearance & increase readability
- Render axioms human-readable
  - StateOwned_deprecated Director only (nomination some min.) vs. StateOwnedCompany hasDirector only (nominatedBy some Ministry)
- Ease tool processing
  - Ontology alignment & text mining
- Ease maintenance of modularity & orthogonality
- Ease communication & annotation

→ Naming conventions proposed by policy makers

Ontology Best Practices

There is a relative lack of guidance on how one actually builds and maintains ontologies. While there is much of a theoretic basis in the literature and on the Web, and much methodology and algorithms, there is surprisingly little on how one actually goes about creating an ontology.

The new classic Ontology Development 101 article as a good starting point. Another really excellent starting point is the Protégé 4 user manual. Though it is obviously geared towards the Protégé tool and its interface, it also is an instructive tutorial on general ontology (OWL) topics and constructs. I highly recommend printing it out and reading it in full.

Naming and Vocabulary Best Practices

- Name all concepts as single nouns. Use CamelCase notation for these classes (that is, class names should start with a capital letter and not contain any spaces, such as MyNewConcept).

- Name all properties as verb senses (so that triples may be actually read), e.g., hasProperty. Try to use mixedCase notation for naming these predicates (that is, begin case sensitive, but still capitalize after and don't use spaces).

- Try to use common and descriptive prefixes and suffixes for related properties or classes (while they are just labels and their names have no inherent semantic meaning, they can be useful way for humans to cluster and understand your vocabularies). For example, properties about languages or tools might contain suffixes such as 'Language' or 'Tool' related properties.

- Provide inverse properties where it makes sense, and adjust the verb senses in the predicates to accommodate. For example, <Father> <hasChild> <Janie> would express inversely as <Janie> <isChildOf> <Janie>.

- Give all concepts and properties a definition. The matching and alignment of things is done on the basis of concepts (not simply labels) which means each concept must be defined. Providing clear definitions (along with the coherence of its structure) gives an ontology its semantics. Remember not to confuse the label for a concept with its meaning. (This approach also aids multi-linguality). In its own ontologies, Structured Dynamics uses the property of skos:definition, though others such as rdf:comment or dc:description are also commonly used.

- Provide a preferred label annotation property that is used for human readable purposes and in user interfaces. For this purpose, Structured Dynamics uses the property of skos:prefLabel.

- Include explicit consideration for the idea of a 'sense', which means a series of alternate labels and terms to describe the concept. These alternatives include true synonyms, also be more expansive and include jargon, slang, acronyms or alternative terms that usage suggests refers to the same concept. The sense construct is similar to the WordNet synset, but with a broader use understanding. Included in the sense construct is the single (per language) preferred (human-readable) label for the concept, the list of alternative and terms for the concept (including acronyms, synonyms, and matching jargon), the altLabels, and a listing of prominent or common spellings for the concept or its alternatives, the hiddenLabels. The sense construct is an integral part of Structured Dynamics approach to using ontologies for information extraction and tagging of unstructured text.

- Try to assign logical and short names to namespaces used for your vocabularies, such as foaf:XXX, uniml:XXX or skos:XXX, with a maximum of five letters prefered.

- Enable multi-lingual capabilities in all definitions and labels. This is a rather complicated best practice in its own right. For the time being, it means being attentive to the xmlns:lang="en" (for English, in this case) property for all annotation properties.

- (If you disagree with these naming conventions, use your own, but in any event, be consistent!!)
The Ten Commandments of Ontological Engineering

Ludger Jansen\textsuperscript{1,2,*}, Stefan Schulz\textsuperscript{3,4}

1 Institute of Philosophy, University of Rostock, Germany
2 Philosophical Institute, RWTH Aachen University, Germany
3 Institute of Medical Biometry and Medical Informatics, Freiburg University Medical Center Freiburg, Germany
4 Institute for Medical Informatics, Statistics and Documentation, Medical University of Graz, Austria

ABSTRACT
The realist approach to ontology design has recently been criticized for being idle philosophical red herring, without any advice for ontology developers. We present guidelines for ontology design in the realist spirit and demonstrate not only how these guidelines are motivated from an realist understanding of ontologies, but also indicate how they are thought to improve the performance of scientific ontologies, especially in biomedicine and the life sciences.

3 TERMS, DEFINITIONS, AND RELATIONS

IV. You shall not make wrongful use of names.

Even if words are not the normal object of representation in an ontology, words are important for the ontologist as instruments to refer to the types and classes to be represented. Names of representational units in an ontology (be they names for classes or for relations) are ideally unambiguous and self-explaining. To this end, naming conventions have been proposed (Schober et al. 2009). Words and terms used in practice tend to be ambiguous. Their makeup is not always fully compositional; figurative use is common, and literal interpretations are often misleading: A complicated pregnancy is a pregnancy, but a prevented pregnancy isn’t, as little as a planned biopsy is a biopsy, or a suspected asthma a kind of asthma. Such idioms should be avoided, as they bear the risk of incorrect subclass assertions such as ‘SuspectedAsthma subClassOf Asthma’

The ambiguity of many terms often remains unrecog...
## Summary of Naming Conventions

The editor-preferred name in an OBO Foundry ontology should comply with the following list of naming conventions:

<table>
<thead>
<tr>
<th>Naming Convention</th>
<th>Description</th>
<th>Example</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Be clear and unambiguous</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1 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>1.2 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', Use 'Bruker US 2 NMR magnet' instead of 'US 2'</td>
<td>Increases precision in the interpreted meaning, Helps string matching, Faster term recognition</td>
</tr>
<tr>
<td>1.3 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>1.4 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></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1 Use univocous names and avoid homonyms</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,</td>
<td>'protocol_collection' instead of 'protocol_set' for a plurality of protocols (store the latter as synonym), 'parameter_adjustment' instead of</td>
<td>Increases precision in the interpreted meaning, Faster term recognition</td>
</tr>
</tbody>
</table>
The OntoCheck Protege 4.1 plugin

Problem: Lack of tool support hindered widespread compliance
Solution: Enrich Protégé 4 with label-check functionalities

OntoCheck plugin

• Checks on naming conventions & metadata cardinality
• Quantifies found violations
• Stores checks & results for later reuse & sharing
Supported Checks

Typographical checks
• Word Case    CamelCase, all lower case, ALL UPPER CASE, ...
• Word Separator none, space, hyphen, underscore, ...

Lexical checks
• Regular Expressions
  – Affix patterns
    • All Role subclasses have ‘_role’ – postfix
    • Check on metalevel postfixes such as ‘_class’, ‘_type’, ‘_concept’, ‘_relation’
  – Boolean operators
    • or, and, non (anti, dis, un)
  – Punctuation
    • Dots hint for abbreviations
  – Digits
    • Numbers hint for cardinality indicators

Character & word counts
• Checks for potentially unclear names
  • Alert on labels shorter than 4 characters
  • Alert on unreadable labels longer than 40 characters

  e.g. ParticipatingInterestsSharesAssociatedEnterprisesUncalledAmountsMovements_lab
The Check Tab (OntoCheck GUI)
OntoCheck Use Case

• Apply OntoCheck in diverse Projects
  – DebugIT project, BioTop, GoodOD, Aneurist, PatOMat

• Check labels in six ontologies
  – Biotop, DCO, NTDO, Good Relations, Vehicle Sales Ontology, @neurist Ontology

• Quantify violations on artefacts own conventions
• Verify if OntoCheck can contribute to test Obo Foundry naming conventions
## Example: Checks on DCO and quantified violations

<table>
<thead>
<tr>
<th>OntoCheck</th>
<th>Tested Entity</th>
<th>Entry Node</th>
<th>Violations abs (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>RegExp,’ValueRegion’</code> postfix</td>
<td><code>&lt;rdf:ID&gt;</code></td>
<td>ValueRegion</td>
<td>167 (54)</td>
</tr>
<tr>
<td>MinCardinality &gt; 2</td>
<td><code>&lt;ru-meta:synonym&gt;</code></td>
<td>Thing</td>
<td>238 (40)</td>
</tr>
<tr>
<td>NameEqualsLabel</td>
<td><code>&lt;rdf:ID&gt;,&lt;rdfs:label&gt;</code></td>
<td>Thing</td>
<td>304 (21)</td>
</tr>
<tr>
<td>MinCardinality = 1</td>
<td><code>&lt;rdfs:label&gt;</code></td>
<td>Thing</td>
<td>184 (12)</td>
</tr>
<tr>
<td>CamelCase</td>
<td><code>&lt;rdf:ID&gt;</code></td>
<td>Thing</td>
<td>34 (8)</td>
</tr>
<tr>
<td>SpaceDelimiter</td>
<td><code>&lt;rdfs:label&gt;</code></td>
<td>Thing</td>
<td>7 (4)</td>
</tr>
<tr>
<td>MaxCharCount &lt; 20</td>
<td><code>&lt;ru-meta:shortLabel&gt;</code></td>
<td>Thing</td>
<td>3 (.5)</td>
</tr>
</tbody>
</table>
1.1 Use explicit and concise names
  RegExp for stopword detection (‘Other’), name length checks
  i.e. labels shorter than three characters are an important source of mismatches in
  alignment algorithms (Burgun & Bodenreider, 2005)

1.2 Use context independent names
  RegExp on explicit pre-, in-, or postfixes

1.3 Avoid taboo words
  RegExp warn on ‘metalevel’ postfixes like ‘class’, ‘type’, ‘concept’, and ‘entity’

2.2 Avoid conjunctions
  RegExp warn on logical connectives

2.4 Use positive names
  RegExp warn on negation indicators e.g. ‘non’, ‘anti ‘or ‘dis’

3.3 Use space as word separator
  Word delimiter checks

3.4 Expand abbreviations and acronyms
  RegExp for punctuation, e.g ‘\.’. CaseConventionTest on all upper case.

4.1 Prefer lower case beginnings: Word case check
OBO Foundry Naming Conventions not supported

- **Need to exploit ontological structure (subsumption hierarchy or relations)**
  - 2.1 Be univocous & avoid homonyms
  - 2.3 Prefer singular
  - 3.1 Recycle strings
  - 3.2 Use genus-differentia style names

- **Need to exploit lexical background knowledge via lexica**
  - 1.4 Avoid encoding administrative metadata in names
  - 2.1 Be univocous & avoid homonyms
  - 2.3 Prefer singular
  - 2.5 Avoid catch-all terms
  - 3.1 Recycle strings
  - 3.2 Use genus-differentia style names
  - 3.5 Expand special symbols to words
  - 4.2 Avoid character formatting

- **Need for inclusion of predefined term-lists to be checked for**
  - 1.4 Avoid encoding administrative metadata in names
  - 2.1 Be univocous & avoid homonyms
  - 2.5 Avoid catch-all terms

- **Need to apply character transliteration tables, mapping special characters to exp. UTF codes**
  - 3.5 Expand special symbols to words
  - 4.2 Avoid character formatting
Summary

• 61 checks carried out on 6 ontologies
  – Mostly on rdf:ID and rdfs:label
    Most frequent: Regular expressions (28%)
    Medium frequent: Case and separator checks (8%)
    Rarest: Comparison between classes (2%)
  – For only two checks, a specific entry node was selected
    • i.e. test Function subclasses for ‘_function’ postfix

• Only half of the Foundry conventions were supported tool in present state
• Some false negatives detected

• ‘Guesstimation’ on violation prevalence
  – Most prevalent violations were of type Affix-inconsistency
  – Medium prevalence were metadata incompleteness and Name-to-Label divergence
  – Less prevalent were delimiter and Case inconsistencies
Next Steps

• **Engage collaboration with OntologyDesignPatterns.org**
  – Formalize ‘Naming ODP’
  – Correlate OntoCheck storage format with ‘Naming ODP’
  – Enable loading checks via ‘Naming ODP’
  – Pre-formalize ‘Sets’ of consistent Naming Conventions
    • Along semantic spectrum
    • OBO Foundry, Manchester style, semweb, ISO ...

• **Expand ‘lexical awareness’ via Parsers / POS tagging**
  – LiLA-API, Linguistic Label Analysis

• **Implement word lists to be checked in labels**
  – Taboo words like Logical connectives, metalevel terms, ‘other’ ...
Conclusions

• Heterogeneity in class labels still prevalent
• A part can be detected & rectified by tool support
• OntoCheck contributed to quality assurance in test setting
  — Also metadata incompleteness detected

• Only half of the OBO Foundry naming conventions could be assisted by OntoCheck
  — Remaining fraction relying on parsing & lexica
• Due to informality and blurredness of conventions, result exploitation was difficult
  — Easier if formalized as Naming-ODPs
Resources & Acknowledgements

Resources

• OntoCheck plugin download
  • www.imbi.uni-freiburg.de/ontology/OntoCheck

• OBO Foundry Naming Conventions
  • http://obofoundry.org/wiki/index.php/Naming

Acknowledgements

• Ilinca Tudose for implementing OntoCheck. Martin Boeker, Vojtech Svatek & Ondrej Svab-Zamazal for reviewing the manuscript and doing tests
• Timothy Redmond for solving Protégé API problems
• This work was supported by the Deutsche Forschungsgemeinschaft (DFG) grant JA 1904/2-1, SCHU 2515/1-1 GoodOD (Good Ontology Design)
### 2. Be univocous

**Use univocous names and avoid homonyms**
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.

**Avoid conjunctions**
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’.

<table>
<thead>
<tr>
<th>Avoid conjunctions</th>
<th><strong>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.</strong></th>
<th>Increases precision in the interpreted meaning</th>
</tr>
</thead>
</table>

**Prefer singular nominal form**
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 postixes as part of the class names, e.g. use ‘aggregate’, ‘collective’ or ‘population’ consistently, but only as applicable.

<table>
<thead>
<tr>
<th>Prefer singular nominal form</th>
<th>‘pair of lungs’, ‘population’ instead of ‘lungs’, ‘people collection’</th>
<th>Increases precision in the interpreted meaning, helps string matching</th>
</tr>
</thead>
</table>

**Use positive names**
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”.

<table>
<thead>
<tr>
<th>Use positive names</th>
<th>‘data recording device’ instead of ‘non-separation device’</th>
<th>Increases precision in the interpreted meaning</th>
</tr>
</thead>
</table>

**Avoid catch-all terms**
Avoid ‘rag-bag’ words that do not designate natural kinds. The existence of classes is not dependent on our biological knowledge.

<table>
<thead>
<tr>
<th>Avoid catch-all terms</th>
<th>Avoid ‘unlocalised’, ‘unknown’, ‘unclassified’</th>
<th>Increases precision in the interpreted meaning</th>
</tr>
</thead>
</table>
### 3. Reduce string variance

<table>
<thead>
<tr>
<th>Recycle strings</th>
<th>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</th>
<th>‘x part of process’, ‘y part of process’ instead of ‘x component of process’, ‘y portion of process’</th>
<th>Helps string matching. Eases cross products generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use genus-differentia style names</td>
<td>Class names should reflect the differentia that distinguish the class from its parent class (modifies 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 (‘<em>’) 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 (‘</em>’) 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>Expand special symbols to words</td>
<td>Special symbols and foreign language letter characters should be spelled out.</td>
<td>‘degree Celsius’, ‘alpha helicase’, ‘carbon-14’ instead of ‘°C’, ‘α helicase’, ‘C14’</td>
<td>Helps string matching</td>
</tr>
</tbody>
</table>

### 4. Typography

<table>
<thead>
<tr>
<th>Use lower case beginnings</th>
<th>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.</th>
<th>Use ‘microarray’, ‘DNA microarray’, ‘pH value’, ‘Golgi apparatus’</th>
<th>Faster term recognition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avoid character formatting</td>
<td>Use plain ASCII format to keep names as computationally pliant 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>
# Requirement analysis

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Aspects met and Implementation</th>
<th>OntoCheck Panel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy installation, usage and intuitive navigation.</td>
<td>Protégé plugin, structured into 3 self-explaining tabs. Tooltips providing on-the-spot guidance.</td>
<td>All</td>
</tr>
</tbody>
</table>
| Generation and display of numeric counts for selectable ontology metrics. | Making use of the Protégé and Java API, diverse metrices are available, amending the already present ‘Ontology Metrics’.
 | All |
| Selection of an ‘entry class node’ from where on - leaf-wards - a check should be done. | Allows test on a certain postfix e.g. ‘_Disposition’ only within a selected ‘Disposition’ entry node sub-tree. Allows checking for metadata availability in selectable subtrees. | All |
| Display of classes failing a specified test and export as list. | Found classes can be sorted according to different criteria and exported for later curation. | All |
| Display of quantitative results on detected issues in terms of absolute and percentage counts in a given subtree. | A statistical data pane verbalizes the numerical results in a copyable natural language sentence. | All |
| Storage and reload capabilities for created checks allowing for later re-use and propagation. | An xml file is generated storing all checks in a reproducible way. | All |
| Detection for ‘presence’ and ‘required cardinality’ of labels and metadata. | Checks are available on OWL elements capturing lexical information, i.e. rdf:ID, rdfs:label, own annotation properties and standard annotation properties e.g. from Dublin Core or SKOS. | Check |
| Check for syntactical and typographical patterns and label length i.e. to discover too short or too long names within string values of selectable entities. | Allows checking naming conventions via simple string matches and full regular expressions. Checks the length of labels. A significant fraction of the OBO Foundry naming conventions can be checked, i.e. case, separator but also morphemic conventions. | Check |
| Detection and counts of redundant class labels. | Label repetition can be checked for via the ComparePanel. | Compare |
| Comparison of values between pairs of entities to detect similarities and avoid redundancies. | Operators like equals, contains or starts with can be used to compare selectable entities. | Compare |
| Quantification of ontology measures useful for ontology evaluation, progress monitoring and complexity analysis. | Displays the percentage or absolute number of entities having ‘exactly’, ‘at least’ or ‘at most’ a certain number of annotation properties, direct sub-/superclasses, or ‘usages’, i.e. indicating ‘hub nodes’. | Count |
The Compare Tab

Compare property 'a', 'b' of the same class

Select a:
  OWLClass
  OWLClass

Select b:
  rdfs:label

Compare in different classes
  OWLClass
  OWLClass

Perform counts case sensitive
  Ignore separators
  Ignore annotation language

Search results

78 out of 1729 (4.5113%) classes passed the test. Count with root ParticularEntity being used at most (<=) 4 times.

1454 out of 1729 (84.5732%) classes passed the test. Count with root ParticularEntity having at least (>=)2 Superclasses.

83 out of 1730 (4.7977%) classes failed the test. Check with root ParticularEntity, Naming convention.

206 out of 1748 (11.7345%) classes failed the test: Compare with root Thing OWLClass, Equals, rdfs:label

Found classes

- MicrogramPerKilogramPerHour
- Microbiological/InfectionValueRegion
- MolecularEntityByEntityPartition
- MixtureOfPharmacologicSubstance
- MolecularEntityByOrganicInorganicPartition
- MSTowardsTSMPSX
- NASAroute
- NeuritisProcess
- Nitroimidazole
- OralRoute
- OrganicMolecularEntityPartition
- OrderHemoptysisValueRegion
- OrganismByDevelopmentalStagePartition
- OrganismByTaxonPartition
- OrganismSample
- OrganPartSample
- PartPressureValueRegion
- PathologicalProcessCIGastrointestinalTract
- Palpation
- ParenteralRoute

Statistical data

Save Load

Sort Alphabetically
The Count Tab

Class hierarchy: Thing

- Thing
- Disposition
- ImmaterialNonphysicalEntity
- ImmaterialPhysicalEntity
- MaterialEntity
- ProcessualEntity
- Quality
- QualityRegion
- Role
- SpatialRegion
- State
- TemporalEntity

OntoCheck: Thing

Check
Compare
Count

Save/Load ontology checks
- Save
- Load

Statistical data

Search results

23 out of 344 (6,6360%) classes passed the test: Count, Thing.

Count classes having exactly 0 times.
Subclasses

Count classes that are used at least (>=) 10 times.

Get results

Result classes

Potentially problematic classes
- Role
- ImmaterialNonphysicalEntity
- TaxonRegion
- ProcessualEntity
- Affecting
- SpatialRegion
- ImmaterialPhysicalEntity
- MaterialEntity
- PluralityO1Organisms
- Population
- StructuredBiologicalEntity
- Cell
- OrganismPart
- CellularComponent
- Organism
- MonomolecularEntity
- PeptideStructure
- SubmolecularGroup
- Function
- Quality
- Canonicity
Can naming conventions facilitate Ontology Alignment?

- Lexical string mapping is core to most alignment methods
- Bad naming is a source for mismatches (Euzenat et al., 2004)
- Only half the automatic alignment tools generated acceptable mappings
  - Ontology Alignment Evaluation Initiative - OAEI-2011
- Hypothesis:
  - Applying naming conventions and enforcing them via OntoCheck should increase their precision and recall
- We reviewed literature and results of the 3 best OAEI alignment algorithms (AgreementMaker, LogMap, CODI)
- In String-based alignments, identify labeling problems leading to
  - wrong matches (false positives)
  - undetected matches (false negatives)
- Check if mismatches can be avoided by enforcing a naming convention via OntoCheck
- All a qualitative rather than quantitative study
### Matching issues & naming conventions to alleviate them

<table>
<thead>
<tr>
<th>Reference</th>
<th>Label Ontology A</th>
<th>Label Ontology B</th>
<th>Differences A B</th>
<th>Convention</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bodenreider, 2005</td>
<td>MA: uterine cervix</td>
<td>NCI: cervix uteri</td>
<td>Qualifyier Head-Noun order</td>
<td>in preparation</td>
</tr>
<tr>
<td>Bodenreider 2005</td>
<td>MA: cervical vertebra 1</td>
<td>NCI: C1 vertebra</td>
<td>Unresolved acronym</td>
<td></td>
</tr>
<tr>
<td>Svab, 2008</td>
<td>AssociatedChair</td>
<td>Chair</td>
<td>First a person role, Second and object</td>
<td></td>
</tr>
<tr>
<td>Cruz, 2009</td>
<td>MA: prostate gland smooth muscle</td>
<td>NCI: gallbladder smooth muscle tissue</td>
<td>Different muscles</td>
<td></td>
</tr>
<tr>
<td>Cruz, 2009</td>
<td>FMA: trapezoid</td>
<td>NCI: trapezoid</td>
<td>First a bone, second tissue</td>
<td></td>
</tr>
<tr>
<td>Jimenez-Ruiz, 2011LogMap</td>
<td>MA 0001741</td>
<td>NCI C49483</td>
<td>False positives</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MA 0001741</td>
<td>NCI C49306</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jimenez-Ruiz, 2011LogMap</td>
<td>MA: outer ear</td>
<td>NCI: auricle</td>
<td>False negative</td>
<td></td>
</tr>
</tbody>
</table>
### Checks per Ontology and association to Foundry naming convention

<table>
<thead>
<tr>
<th>Ontology</th>
<th>Checks</th>
<th>% NC Checks</th>
<th>NC (times)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCO</td>
<td>11</td>
<td>81</td>
<td>1.2 (4x), 2.1, 2.2, 3.1, 4.1, 4.2</td>
</tr>
<tr>
<td>NTDO</td>
<td>2</td>
<td>50</td>
<td>1.3</td>
</tr>
<tr>
<td>Biotop</td>
<td>13</td>
<td>31</td>
<td>1.3 (2x), 3.1, 4.1</td>
</tr>
<tr>
<td>@neurist</td>
<td>13</td>
<td>38</td>
<td>1.3 (2x), 3.1, 4.1</td>
</tr>
<tr>
<td>GoodRelations</td>
<td>11</td>
<td>36</td>
<td>2.2 (2x), 2.4, 4.1</td>
</tr>
<tr>
<td>Vehicles</td>
<td>10</td>
<td>60</td>
<td>2.2 (3x), 4.1</td>
</tr>
</tbody>
</table>
## Applied OntoCheck functions, mapped onto Foundry conventions

<table>
<thead>
<tr>
<th>OntoCheck</th>
<th>Amount</th>
<th>%</th>
<th>NC (times)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegExp, infix</td>
<td>13</td>
<td>21.3</td>
<td>2.2(7x), 1.3(5x), 2.4</td>
</tr>
<tr>
<td>CaseConventionTest</td>
<td>5</td>
<td>8.1</td>
<td>4.1(5x)</td>
</tr>
<tr>
<td>CompareValuesForSingleCls</td>
<td>5</td>
<td>8.1</td>
<td>3.1(3x)</td>
</tr>
<tr>
<td>WordDelimiter</td>
<td>5</td>
<td>8.1</td>
<td>3.3(5x)</td>
</tr>
<tr>
<td>RegExp, length</td>
<td>3</td>
<td>4.9</td>
<td>1.2(3x)</td>
</tr>
<tr>
<td>RegExp, postfix</td>
<td>2</td>
<td>3.2</td>
<td>1.2(2x)</td>
</tr>
<tr>
<td>CompareValuesBetweenCls</td>
<td>1</td>
<td>1.6</td>
<td>2.1</td>
</tr>
</tbody>
</table>
Next Steps

• Expand OntoChecks ‘lexical awareness’ via Parsers / POS tagging

  • LiLA (Linguistic Label Analysis) API
    – Singularize labels
    – Correlate labels with log defs
      » (HospitalRegisteredPatient \(\rightarrow\) Patient SubclassOf registered_with Some Hospital)

  – Compositional analysis to
    • infer structural patterns
      – Stevens et al., 2003 show how labels can be exploited to infer missing subsumptions,
        » i.e. ‘heparin biosynthesis’ is-a ‘glycosaminoglycan biosynthesis’, as ‘heparin’ is-a ‘glycosaminoglycan’

• make suggestions for re-occurring naming patterns
  – Exploit re-occurring strings among subtrees
  – ‘Morpheme-frequency analyzer’ can normalize affixes.
    » X-itis, X-inflammation , inflammation-of-X \(\rightarrow\) inflammation-of-X
CheckTab - Future Extensions

• Check for naming clashes & redundancies
  • Classes with different IDs but equal labels
  • Check for plural word forms

• Check on non-ASCII characters
  • $\alpha \rightarrow$ alpha

• Check on redundant restrictions
  • Between own and inherited axiomatic class definitions