# **Ontology Creation**

* Relation Extraction is extracting relations between entities and arguments. After extraction all relations, these axioms can be represented into the ontology using different standardized languages XML, RDF or OWL.
* RDFs are less powerful than OWL but allows schema Relation like class hierarchies, domains and ranges for predicate to be expressed in RDF.
* RDF is holding a linked collection of (entity, attribute, value) which supports in term o0f storage and query applications.
* RDF specification permits an attribute (or property, or predicate=an edge joining 2 nodes in graph terms) to be a subject node as well.
* For instance, in order to specify domain and range of the attribute as rdf statements;

(Picasso, paints, “Guernica”)

Paints is a predicate or edge is complemented by two schema relations defining the predicate.

(Paints, domain, Painter)

(Paints, range, Painting)

Where Paint is a node.

* RDF is a directed, labeled graph data format for representing information in the Web. This specification defines the syntax and semantics of **the SPARQL query** language for RDF. SPARQL can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions. SPARQL also supports extensible value testing and constraining queries by source RDF graph. The results of SPARQL queries can be results sets or RDF graphs.

# **Ontology Evaluation Catalogue**

## **Accuracy:**

A higher accuracy comes from correct definitions and descriptions of classes, properties, and individuals. There are different methods to evaluate accuracy of the ontology.

|  |  |
| --- | --- |
| **Methods** | **Description** |
| Hash vs Slash URI | * The basic question is on the difference between using http://example.org/ontology#joe and <http://example.org/ontology/joe> in order to refer to a non-information resource. * The former type of URI is called a hash URI (since the local part is separated by the hash character # from the namespace), the latter type a slash URI (since the local part is separated by the slash character / from the namespace). * When resolving a hash URI, only the namespace is resolved. All hash URIs with   the same namespace thus resolve to the same resource. This has the advantage that the ontology can be downloaded in one pass, but it also has the disadvantage that the file can become very big.  Therefore, terminological ontologies and ontologies with a closed, rarely changing, and rather **small set of individuals** (e.g. a list of all countries) would use **hash URIs**, whereas open domains with often **changing individuals** often **use slash URIs** |
| **Querying for anti-patterns:** To detect so called anti-patterns is at least as important as detecting patterns in  ontologies. Anti-patterns are strong indicators for problems in an ontology. | (Searching for Anti-Patterns)  SPARQL queries over the ontology graph can be used to discover potentially problematic patterns. For example results to the following queries have been found to be almost always problems.  Detecting the anti-pattern of subsuming nothing:  select ?a where {  ?a rdfs:subClassOf owl:Nothing .  }  Detecting the anti-pattern of skewed partitions:  select distinct ?A ?B1 ?B2 ?C1 where {  ?B1 rdfs:subClassOf ?A .  ?B2 rdfs:subClassOf ?A .  ?C1 rdfs:subClassOf ?B1 .  ?C1 owl:disjointWith ?B2 .  } |
| Class and relation ratio | Measure (M29) in (Gangemi et al., 2005) is called the "Class / relation ratio",  suggesting that it returns the ratio between classes and relations (or properties). The  exact de\_nition of the measure is: "nG∈S/nR∈S  where nG∈S is the cardinality of the set of  classes represent[ed] by nodes in g, and nR∈S is the cardinality of the set of relations  represented by arcs in g" (Gangemi et al., 2005). |
| Checking competency questions against results | * Formalize your competency question as a SPARQL query. Write down the expected answer as a SPARQL query result * Compare the actual and the expected results. Note that the order of results is often undefined. |
| Checking competency questions with constraints | Formalize your competency question for ontology O as a SPARQL CONSTRUCT  query that formulates the result in RDF as ontology R. Merge R with O and  a possibly empty ontology containing further constraints C. Check the merged ontology for inconsistencies. |
| Affirming derived knowledge | * Unit tests for ontologies test if certain axioms can or cannot be derived from the Ontology * For each axiom A+I in the positive test ontology T+ test if the axiom is being inferred by the tested ontology O. For every axiom that is not being inferred, issue an error message. * For each axiom A- in the negative test ontology T- test if the axiom is being inferred by the tested ontology O. For every axiom that is being inferred, issue an error message. |
| Expressive consistency check | An ontology O can be accompanied by a highly axiomatized version of the ontology,  C. The merged ontology of OUC has to be consistent, otherwise the inconsistencies  point out to errors in O. |
| Inconsistency checking with rules | * Translate the ontology to be evaluated and possible constraint ontologies to a logic program. This translation does not have to be complete. Formalize further constraints as rules or integrity constraints. * Concatenate the translated ontologies and the further constraints or integrity constraints. Run the resulting program. If it raises any integrity constraints, then the evaluated ontology contains errors. |

## **Adaptability**

|  |  |
| --- | --- |
| **Methods** | **Description** |
| URI declarations and punning | * Web ontologies do not require names to be declared. This leads to the problem that it is impossible for a reasoner to discern if e.g. ex: Adress is a new entity or merely a typo of ex: Address. This can be circumvented by requiring to declare names, so that tools can check if all used names are properly declared. * The declarations are axioms, stating not only that a name exists but also its type,i.e. if it is declared as a class, an individual, a data type, object or annotation property. This feature was introduced in OWL 2. * For example, the entity lion, depending on the context, may represent the individual lion that is of the type species, or it may be the type of the lion Simba. There is no necessity to introduce different names for the two, or to render the merger of two ontologies inconsistent where lion is used as a class in the one ontology, and as an individual in the other. |
| Blank Nodes | Blank nodes should be avoided unless structurally necessary. |
| Querying for anti-patterns | (Searching for Anti-Patterns)  SPARQL queries over the ontology graph can be used to discover potentially problematic patterns. For example results to the following queries have been found to be almost always problems.  Detecting the anti-pattern of subsuming nothing:  select ?a where {  ?a rdfs:subClassOf owl:Nothing .  }  Detecting the anti-pattern of skewed partitions:  select distinct ?A ?B1 ?B2 ?C1 where {  ?B1 rdfs:subClassOf ?A .  ?B2 rdfs:subClassOf ?A .  ?C1 rdfs:subClassOf ?B1 .  ?C1 owl:disjointWith ?B2 .  } |
| Stability | * Calculate a normalized class depth measure, i.e. calculate the length of the longest subsumption path on the normalized version of the ontology md (N (O)). Now calculate the stable minimal depth of the ontology mdmin(O). If md (N (O)) != mdmin(O) then the ontology hierarchy is not stable and may collapse. * Ontology engineers will have these numbers available at engineering and maintenance time, they will learn easier how to achieve their actual goals. * For example, ontology engineers that want to create a class hierarchy that will not collapse to less levels can always check if the minimum depth as described above corresponds to the asserted depth. This would be useful when regarding a class hierarchy with a certain number of levels, which are known not to collapse (e.g. a biological taxonomy). * The ontology engineer now could check if the well-known number of levels indeed corresponds to the calculated minimum depth. * Tools could guide the ontology engineer towards achieving such goals. Ontology engineers get more aware of such problems, and at the same time get tools to measure, and thus potentially control them. |
| Maximum depth of the taxonomy |  |
| Affirming derived knowledge | * Unit tests for ontologies test if certain axioms can or cannot be derived from the Ontology * For each axiom A+I in the positive test ontology T+ test if the axiom is being inferred by the tested ontology O. For every axiom that is not being inferred, issue an error message. * For each axiom A- in the negative test ontology T- test if the axiom is being inferred by the tested ontology O. For every axiom that is being inferred, issue an error message. |
| Expressive consistency checks | An ontology O can be accompanied by a highly axiomatized version of the ontology,  C. The merged ontology of OUC has to be consistent, otherwise the inconsistencies  point out to errors in O. |
| Inconsistency checking with rules | * Translate the ontology to be evaluated and possible constraint ontologies to a logic program. This translation does not have to be complete. Formalize further constraints as rules or integrity constraints. * Concatenate the translated ontologies and the further constraints or integrity constraints. Run the resulting program. If it raises any integrity constraints, then the evaluated ontology contains errors. |

## **Clarity**

|  |  |
| --- | --- |
| **Methods** | **Description** |
| Linked Data: check use protocols | All URIs in the ontology are checked to be well-formed URIs. The evaluator has to choose a set of allowed protocols for the evaluation task. The usage of any protocol other than HTTP should be explained. All URIs in the ontologies have to use one of the allowed protocols. |
| Linked Data: check response code | For all HTTP URIs, make a HEAD call (or GET call) on them. The response code should be 200 OK or 303 See Other. Names with the same slash namespace should return the same response codes, otherwise this indicates an error. |
| Hash vs slash | same |
| Opaqueness of URIs | A proper naming can be checked by comparing the local part of the URI with the label given to the entity or by using lexical resources like WorldNet. Formalize naming conventions (like multi-word names and capitalization) and test if the convention is applied throughout all names of a namespace. Check if the URI fulfills the general guidelines for good URIs, i.e. check length, inclusion of query parameters, \_le extensions, depth of directory hierarchy, etc.)  Note that only local names from the same namespace, not all local names in the ontology, need to consistently use the same naming convention, i.e. names reused from other ontologies may use different naming conventions. |
| URI declarations  and punning | same |
| Typed literals and datatypes | * A set of allowed data types should be created. All data types beyond those recommended by the OWL specifications should be avoided. Creating a custom data type should have a very strong reason. xsd: integer and xsd:string should be the preferred data types (since they have to be implemented by all OWL conformant tools). * Check if the ontology uses only data types from the set of allowed data types. * All typed literals must be syntactically valid with regards to their data type. The evaluation tool needs to be able to check the syntactical correctness of all allowed data types. |
| Language tags | * Language tags can be used on plain literals to state the natural language used by the literal. This enables tools to choose to display the most appropriate literals based on their user's language preferences. An example for a literal with a language tag is "university"@en or "Universitat"@de. Language tags look rather simple, but are based on a surprisingly large set of speci\_cations and standards. * Check that all language tags are valid with regards to their specification. * Check if the shortest possible language tag is used (i.e. remove redundant information like restating default scripts or default regions). * Check if the stated language and script is actually the one used in the literal. |
| Class or Relation ratio | same |

## **Completeness**

|  |  |
| --- | --- |
| **Methods** | **Description** |
| Hash vs Slash | same |
| URI declarations and punning | same |
| Typed literals and datatypes | same |
| Labels and comments | same |
| Blank nodes | same |
| XML validation | * An ontology can be validated using a standard XML validator under specific circumstances. * In order to apply this, the ontology needs to be serialized using a pre-defined XML schema. The semantic difference between the serialized ontology and the original ontology will help in discovering incompleteness of the data (by finding individuals that were in the original ontology but not in the serialized one).   The peculiar advantage of this approach is that it can be used with well-known tools and expertise. |
| Structural metrics in practice | - |
| Stability | same |
| Maximum depth of the taxonomy | same |
| Formalized competency questions | same |

## **Computational efficiency**

|  |  |
| --- | --- |
| **Methods** | **Description** |
| URI declarations and punning | same |
| Typed literals and datatypes | same |
| Structural metrics in  practice |  |
| Language completeness |  |

## **Conciseness**

|  |  |
| --- | --- |
| **Methods** | **Description** |
| URI reuse | In order to ease sharing, exchange, andvaggregation of information on the Semantic  Web, the reuse of commonly used URIs proves to be helpful (instead of introducing new names). |
| Blank nodes | same |
| Stability | same |
| Maximum depth of the taxonomy | same |
| Class / relation ratio | same |
| Formalized competency questions | same |

## **Consistency**

|  |  |
| --- | --- |
| **Methods** | **Description** |
| Hash vs slash | same |
| Opaqueness of URIs | same |
| URI reuse | same |
| Labels and comments | same |
| Structural  metrics in practice | same |
| Querying for anti-patterns | same |
| Language completeness |  |
| Affrming derived knowledge | same |
| Expressive consistency checks | same |
| Consistency checking with rules | same |

## **Organizational fitness**

|  |  |
| --- | --- |
| **Methods** | **Description** |
| Linked Data: check use protocols | same |
| Linked Data: check response code | same |
| Hash vs Slash | same |
| Opaqueness of URIs | same |
| URI reuse | same |
| Language tags | same |
| Labels and comments | same |
| XML validation | same |
| Formalized competency questions | same |