Statistical Knowledge Patterns for Characterising Linked Data

Eva Blomqvist[1], Ziqi Zhang[2], Anna Lisa Gentile[2],
Isabelle Augenstein[2], and Fabio Ciravegna[2]

[1] Department of Computer and Information Science, Linköping University, Sweden
[2] Department of Computer Science, University of Sheffield, UK
eva.blomqvist@liu.se,
{z.zhang,a.l.gentile,i.augenstein,f.ciravegna}@dcs.shef.ac.uk

Abstract. Knowledge Patterns (KPs), and even more specifically Ontology Design Patterns (ODPs), are no longer only generated in a top-down fashion, rather patterns are being extracted in a bottom-up fashion from online ontologies and data sources, such as Linked Data. These KPs can assist in tasks such as making sense of datasets and formulating queries over data, including performing query expansion to manage the diversity of properties used in datasets. This paper presents an extraction method for generating what we call Statistical Knowledge Patterns (SKPs) from Linked Data. SKPs describe and characterise classes from any reference ontology, by presenting their most frequent properties and property characteristics, all based on analysis of the underlying data. SKPs are stored as small OWL ontologies but can be continuously updated in a completely automated fashion. In the paper we exemplify this method by applying it to the classes of the DBpedia ontology, and in particular we evaluate our method for extracting range axioms from data. Results show that by setting appropriate thresholds, SKPs can be generated that cover (i.e. allow us to query, using the properties of the SKP) over 94% of the triples about individuals of that class, while only needing to care about 27% of the total number of distinct properties that are used in the data.

1 Introduction

Originally, the notion of Ontology Design Patterns (ODPs) referred only to a top-down view on modelling best practices, and constituted manually designed patterns representing those best practices. More recently, however, Knowledge Patterns (KPs), as a generalisation of ODPs and other patterns, have also been created in a bottom-up fashion, i.e., representing the way information on the Web or Linked Data is actually represented, rather than how it "should" be represented according to some best practice. This paper follows the more recent tradition and presents what we call Statistical Knowledge Patterns (SKPs), which aim to characterise concepts that exist within Linked Data based on a statistical analysis of those data. Since the SKPs are wholly based on the characteristics of data itself, their construction is a completely automatic process, which means that they can be kept up-to-date with respect to data without any manual effort.

In a related paper [15] we have presented the details of the initial steps of the SKP generation method, with specific focus on discovering relations that are (to some extent)

synonymous, and evaluating that part of the extraction in the context of query expansion. In this paper we instead focus on the pattern extraction method as a whole, and the resulting resource, i.e., the pattern catalogue, and in particular discuss the parts of the method not covered by the previous paper. In Section 2 we first present some related work on ODP generation from different sources. We then briefly present our SKP extraction method in Section 3, and exemplify the resulting SKPs in Section 4. In Section 5 we show through some empirical findings that the SKPs fulfill their purpose, i.e., characterise and provide access to the underlying data, but in particular we study and evaluate the range extraction method. Finally, in Section 6 we discuss some general implications of this research, and in Section 7 we provide conclusions and outline future work.

2 Related Work

Ontology Design Patterns (ODPs) were originally conceived for the task of ontology engineering, and in particular were intended to encode general best practices and modelling principles in a top-down fashion [5,6]. Since then several kinds of patterns have been proposed, such as *Content Ontology Design Patterns* (CPs) [7]. CPs focus on domain-specific modelling problems and can be represented as small, reusable pieces of ontologies. CPs are similar to the SKPs presented in this paper, in the way that they also represent concepts with their most distinguishing characteristics. Unlike SKPs however, CPs are usually created manually, and since they are abstract patterns intended for being used as "templates" in ontology engineering they usually lack any direct connection to data and cannot directly (without manual specialisation) be used for querying Linked Data. Since CPs represent an abstract top-down view, they additionally do not consider aspects such as diversity and synonymy among properties, which is one of the benefits that our proposed SKPs display.

The approach closest to our SKPs is the *Encyclopedic Knowledge Patterns* (EKPs) [11], which were intended mainly for use in an exploratory search application [9,10]. The EKP generation process exploits statistics about link-usage from Wikipedia[3] to determine which relations are the most representative for each concept. The assumption is that if instances of a target concept `A` frequently link to instances of concept `B`, then concept `B` is an important concept for describing instances of `A`. This information is then formalised and stored as small OWL ontologies (the EKPs), each having one main class as their focus and all its significantly frequent relations (based on the wiki-link counts) to other classes represented as object properties of the main class. The main purpose of these EKPs is presenting relevant information to a human user, e.g., the ability to filter out irrelevant data when presenting information about DBpedia entities, while the ability to query for data is not a primary concern. This is reflected by the fact that EKPs mainly contain abstractions of relevant properties, such as "linksToClassB", where `linksToClassB` expresses the fact that the pages in Wikipedia representing instances of concept `A` (the class in focus of the EKP) commonly links to pages in Wikipedia representing instances of concept `B` (links which could in many cases in turn be represented by various DBpedia properties, but not necessarily). This is however not sufficient for our case, since our main goal is to use our SKPs to characterise

[3] `http://en.wikipedia.org`

and give effective access to actual data. In such a use case one needs to be able to distinguish between, for instance, different properties that link instances of the same classes but have different meaning (e.g., birth place and death place, which both link a person to a location). Hence, we propose an extension of the existing EKPs, which also include a sufficient coverage of actual properties of the underlying datasets, together with additional features we attach to each of those properties, such as range axioms.

There exist other approaches aiming to statistically characterise datasets, such as the one by Basse et al. [3], which also exploits statistics from a specific dataset to produce topic frames of that dataset. In contrast to Nuzzolese et al. [11] they do not produce a pattern for each class but rather generate clusters of classes (up to 15 classes each) that reflect main topics of the dataset. For giving access to data (querying), however, the main focus needs to be on the properties of the classes, rather than the classes themselves. Also Atencia et al. [1] perform statistical analyses on datasets, but for the purpose of detecting key properties (i.e., to be expressed through the OWL2 notion of "key") rather than characterising the complete property landscape of a class. A related approach is also the LODStat framework [2], which has the broader scope of extracting an publishing many kinds of interesting statistics about datasets. While that framework also takes into account statistics on property usage, and declaratively represents the statistics, the approach is focused on per-dataset statistics, rather than per-class, and does not induce new information (e.g., synonymity or new range axioms) from the extracted statistics.

Looking at patterns from a more general perspective, however, Knowledge Patterns (KPs) have been defined as general templates or structures used to organise knowledge [8], which can encompass both the "traditional" view of ODPs and more recent effort such as EKPs and our SKPs. In the Semantic Web scenario they are used both for constructing ontologies [4,7,13] and for using and exploring them [3,9,10,11,12]. Presutti et al. [12] explore the challenges of capturing Knowledge Patterns in a scenario where explicit knowledge of datasets is neither sufficient nor straight-forward, which is the case for Linked Data. They propose a dataset analysis approach to capture KPs and support datasets querying. Our SKPs expand on this work as not only do we capture direct statistical information from the underlying datasets, but also further characterise relevant properties with additional features (e.g., synonymous properties and range axioms), which is highly beneficial for querying the datasets.

3 SKP Construction Method

A Statistical Knowledge Pattern (SKP) is an ontological view over a class that summarises the usage of the class (hereafter called the *main class* of the SKP) in data. The main class of an SKP can be seen as the "focus", or the context, of that SKP, hence, each SKP has exactly one main class. The term "statistical" refers to that the pattern is constructed based on statistical measures on data. Each SKP contains: (1) properties and axioms involving the main class that relates it to other classes, derived from a reference ontology or from a pre-existing EKP characterising that class; (2) properties and axioms involving the main class that are not formally expressed in the reference ontology, but which can be induced from statistical measures on statements published as

Linked Data. The information from (1) and (2) is consolidated in the form of an SKP, which is represented and stored as a small OWL ontology.

More formally, let the main class of an SKP be c_{main}, which is a class of the selected reference ontology – in fact, this is the only thing we need from the reference ontology, hence, the ontology can simply consist of one or more class URIs if nothing else is available, as long as there is some data using that class. The main class is the starting point for extracting an SKP, hence, it is selected before the construction process begins, and normally one would build SKPs for as many of the classes in the reference ontology as possible (or for the classes that are of specific importance in some use case). The SKP of c_{main} contains the set of properties from the reference ontology $P_{ont} = \{p_{ont1} \dots p_{ont-n}\}$ and the set of properties from any pre-existing EKP of the main class $P_{ekp} = \{p_{ekp1} \dots p_{ekp-m}\}$, with the requirement that only properties that are actually used in data (or have relations to properties that are actually used in data, see further below, are included). A property from the reference ontology or an EKP, p_i may have a set of "synonymous properties" SP_i induced from data. The decision on synonymity of properties is based on a *synonymity* measure (described in detail in [14]), hence, almost none of the properties are actual synonyms (i.e., with a maximum score) but rather represent properties that are to some extent exchangeable in the particular context of the main class. While we will continue to use the term "synonymous properties" throughout this paper, the reader should bear in mind that these are rarely perfect synonyms, but rather "close matches" (as we shall see later, this is also represented in the resulting model through `skos:closeMatch` rather than equivalence). To decide which properties, or synonym clusters of properties, should be selected to be included in the SKP, their *relevance* is measured based on the frequency of usage of the properties in available Linked Data.

In practice, since SKPs are an extension of EKPs [11], if an EKP already exists it can be used as an abstract frame for the concrete properties and axioms that are added through our SKP generation method. In particular, we use the abstract properties introduced by EKPs (i.e., "links to class X") in order to group properties with range axioms overlapping the general EKP property, to give the SKP a more intuitive structure and improve human understandability of the pattern. The properties are thereby organised in two hierarchical layers, through the `rdfs:subPropertyOf` relation, where, in particular, domain and range restrictions of properties are used to induce sub property relations between the very general properties of a pre-existing EKP and the properties retrieved from data. Note that we are, at this point, not attempting to induce a sub-property structure among the properties found in data, hence, we only group them under the general EKP properties. A more elaborate structuring of the extracted properties is still part of future work.

The most important characteristics of SKPs and their generation are:

- SKPs encode class-specific characterisations of properties that are commonly used with individuals of that class, i.e., synonymous properties, ranges, etc. are all specific to the use of the properties with instances of that class, which provides an interesting and detailed account of property meanings and usage in Linked Data. For example, the same property may be present in several SKPs, but with distinct range axioms, and as part of separate property synonym clusters, depending on that the property is used differently with instances of the respective main class of each SKP.

- Synonymous (i.e., to some extent interchangeable) properties are identified, and information about them are stored to be reused; one possible usage is query expansion, when querying the data underlying the SKP. See [15] for details.
- Ranges are identified for properties that have no range in the reference ontology, hence, showing the actual use of the property in data, which can be used to restrict property selection when building a query or to filter out unwanted data at query-time.
- The method for SKP generation is fully automated, whereby SKPs can be regenerated as soon as data changes, without manual effort, but SKPs are in the meantime used as stored resources, for increased usage efficiency.

The SKP generation process consists of three key components: (1) discovering and grouping synonymous properties of the main class, (2) selecting properties (and groups of properties) to include in the SKP, and (3) collecting additional axioms describing the selected properties, such as `rdfs:subPropertyOf` relations and domain and range restrictions, and creating an ontological representation of the SKP.

Synonymity of Properties To create an SKP we identify the properties used for the SKP main class based on data and measure their synonymity. In [14] we have proposed a novel synonymity measure of properties. The overall process is:

1. Query the dataset for all the instances (IND) of the main class; query the dataset for all triples having any $i \in IND$ in subject position (we denote this triple set TS) and additionally collect the types (through querying for *rdf:type* statements or for a datatype) of the objects of all those triples.
2. For each property used in TS, collect the subset of IND having the property as predicate, IND_{prop}, and collect the corresponding objects of each subject in IND_{prop} – the *subject-object* pairs of this set represents the characteristics of that property, given the main class at hand.
3. Do a pairwise comparison of all subject-object pairs of IND_{prop} for all the properties and calculate a *synonymity* score for each pair of properties.
4. Use the *synonymity* scores (representing evidence of properties being interchangeable) to cluster properties that are likely to represent a sufficiently similar (i.e., sufficiently synonymous) semantic relation.

Selection of Properties The aim of the above process is to discover, for each specific main class, clusters of properties with the same meaning. In practice, a certain number of properties are found to be noise or non-representative of the main class. Thus, we further refine the set of properties for each SKP as follows:

5. Calculate the frequencies of properties used in data, i.e., counting distinct objects in IND_{prop}. For clusters, treat the cluster as if it was a single property hence add the frequency counts of the constituent properties.
6. Use a cutoff threshold T (explored further in [15]) to filter out infrequent properties (or clusters), as they may represent noise in the data. Add those above the threshold to the SKP, including information about their appropriate property type (e.g. `owl:DatatypeProperty` or `owl:ObjectProperty`), with their original namespace intact.
7. For each member of a property cluster that is added to the SKP, add a `skos:closeMatch` relation between the cluster members.

Characterisation of Properties Finally, we add as much information as we can about the selected properties, based on what we can induce from the data, and retrieve from the reference ontology or the pre-existing EKP.

8. For each property, add a range axiom that consists of any range that is given to the property in the reference ontology or the EKP (if present), but if not present instead add any range that is identified in data (i.e., by looking at the frequencies of the object types of the triples above a certain threshold).
9. Add `rdfs:subPropertyOf` axioms for those properties where the ranges match some abstract EKP property (i.e., the "links to class X" abstract properties).
10. Store the SKP as an OWL file.

More in detail the range extraction method starts by inspecting the types of all the triple objects in TS that were retrieved at the beginning of the overall process. This is done on a per-property-basis, i.e., for each property selected for inclusion in the SKP, which does not have a range axiom defined in the reference ontology, the corresponding subject-object pairs are again analysed, and this time inspected together with the types of the objects of those pairs. Assume that the set of distinct objects, for the triples of TS using a property p_i is OBJ_{p_i}. Now, count the frequency of the types of the instances in OBJ_{p_i}, i.e., associating each class (or datatype) $type_j$ that is a type of one of the instances in OBJ_{p_i} with a count value $count_{type_j}$. Then calculate the relative frequency of this type, for the specific property, by dividing $count_{type_j}$ with the total number of distinct objects of that property, i.e., $|OBJ_{p_i}|$. Intuitively, this is a measure of how large fraction of the triple objects in the set of triples characterising this property that "support" this type being in the range of p_i.

For avoiding to include too much noise in the axiomatisation of the SKPs, a threshold is set on this "range support" value, i.e., a class should not be included unless it has sufficient support in the data. Where, "sufficient" may differ depending on if one prioritises precision or recall. We investigate a reasonable trade-off for the relative threshold in Section 5, however, we also set an absolute threshold (for really small triple sets) not to include any type that has less than 10 occurrences in the triple set. Since this process may result in a set of classes being selected as the appropriate range of a property, the range axiom included in the SKP is then expressed as the union of those classes.

4 Results

The resulting patterns have been published[4] in the form of small OWL ontologies. Where pre-existing EKPs exist, they can be extended with new properties, while if no pre-existing EKP existed, the SKP is generated completely from scratch. Overall, an SKP contains the main class that is the focus of the pattern, and the properties that are selected for that SKP, including their domain and range axioms. The name of the SKP is the same as the name of the main class. As an example, we illustrate a small part of the resulting SKP called Language[5] in Figure 1, with the main class `dbpedia:Language`. This is one

[4] SKPs are being made available at `http://www.ontologydesignpatterns.org/skp/`
[5] http://www.ontologydesignpatterns.org/skp/Language.owl

of the smallest SKPs generated in our evaluation set (see Section 5), only including 36 distinct properties, distributed over 35 object properties and 1 datatype property. Each property has kept its original URI, so as to be directly usable for querying data, and is given the main class of the SKP as domain. In this particular SKP we, for instance, find properties such as `dbpedia:spokenIn`, `dbprop:region` and `foaf:name`, i.e., coming from three different namespaces. At a first glance, `foaf:name` may seem to be an error, however, this nicely exemplifies the SKPs ability to reflect actual usage in data. The property was certainly not intended for expressing the name of languages, however, for this particular class the property is actually used in this way and could be useful to include when querying for data about languages. Without seeing the SKP, or experimenting with queries manually, this may be hard to discover.

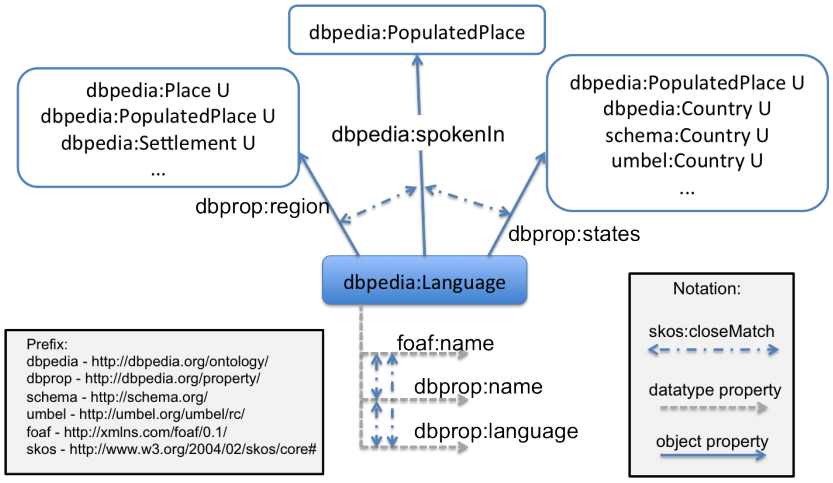


Fig. 1. Illustration of a small part of the Language SKP. Classes are illustrated as boxes, including the union classes representing complex ranges, and properties as arrows. An arrow starting from a class means that is the domain of the property, and the class at the end of the arrow is the range. The `skos:closeMatch`-arrows represent assertions on properties.

The property `foaf:name` is additionally part of a property cluster, which includes additional properties such as `dbprop:name` and `dbprop:language`, which represent properties that may be considered as synonymous to `foaf:name` in the context of the class `dbpedia:Language` and are linked to each other in the SKP though the property `skos:closeMatch`. The property `dbprop:language` is another good example of a highly ambiguous property name, which is not easy to interpret, without actually looking at its detailed use with individuals of this particular class (i.e., individuals of `dbpedia:Language`). Another example of a property cluster is the one containing the object properties `dbpedia:spokenIn`, `dbprop:region`, and `dbprop:states`, which are all used to express the area, or usually the country, where a language is spoken.

The properties dbprop:region and dbprop:states did not have any prior range axioms defined, since they are not part of the DBpedia ontology, but rather of the part of DBpedia that is generated completely automatically without aligning it to the ontology. As an obvious remedy, one may consider using the range of dbpedia:spokenIn also for the other members of the cluster. However, not all properties are involved in clusters that include properties with range axioms in the reference ontology, this is actually true only for a small fraction of the total number of properties. Hence, although not absolutely necessary in this case, we may generate range axioms directly from data for the two properties. The property dbprop:region then, for instance, receives the union of the following classes as its range: dbpedia:Place, dbpedia:PopulatedPlace, dbpedia:Settlement, schema:Place and opengis:_Feature.

5 Experiments

In the related paper [15] the extraction of synonymous properties was evaluated, together with the property selection threshold. In this paper we focus on analysing the range extraction method, but additionally show some general statistics in order to motivate the usefulness of the SKPs we are proposing. For performing the experiments we selected a set of 34 DBpedia classes to focus on, and generated SKPs for these. The classes were not selected randomly, but rather we focused on the DBpedia classes that are involved in answering the benchmark queries in the QALD-1 query set[6], as our evaluation set.

5.1 Pattern Characteristics

SKPs aim at reducing the complexity of understanding and querying data, by reducing the diversity of properties to only include the core properties of the main SKP class. However, to be useful in practice, such a reduced representation should still allow for accessing as large part of the underlying data as possible. This is a trade-off that the SKPs must be able to sufficiently support if they are to be used in practice. To illustrate that the SKPs do fulfill both these requirements sufficiently well, Table 1 presents some statistics of the set of 34 SKPs in our evaluation set.

	Min	Average	Max
Number of included properties	31	107	436
Percentage of included properties	12%	27%	38%
Percentage of data triples covered	88%	94%	97%

Table 1. Characteristics of the generated SKPs

The patterns range in size (in terms of the number of properties of the main class) between 31 and 436 properties. While 436 properties may be perceived as a large number,

[6] QALD-1 contains a "gold standard" of natural language questions associated with appropriate SPARQL queries and query results, see: http://greententacle.techfak.uni-bielefeld.de/~cunger/qald1/evaluation/dbpedia-test.xml

this should be considered in light of the second row of the table, i.e. the fraction of the total number of properties used for that main class in the data that the included properties represent. For instance, the largest pattern, with 436 properties included, is the `AdministrativeRegion` pattern characterising the `AdministrativeRegion` class in the DBpedia ontology, which in total uses 1235 distinct properties with its 28229 instances in the DBpedia dataset. Hence, those 436 properties constitute only 35% of the total number of distinct properties, but still allows us to access 89% of the data triples, about `AdministrativeRegion` instances. In the last row of the table we summarise similar results for the complete SKP set, i.e., on average the SKPs allow us to still access 94% of the data about their instances, while reducing the number of properties to on average 27% of the original number. One should also keep in mind that these are SKPs generated with a particular property inclusion threshold (see [15] for a detailed evaluation and discussion of the threshold), whereby tailored sets of SKPs could also be generated with a specific use case in mind, prioritising either triple coverage or reduced size of the SKP as needed.

We have not yet evaluated how the accuracy of the data, and responses to queries, are affected by filtering out some portion of the properties used in data. This is mainly due to the difficulty of evaluating the quality of data in DBpedia in general, i.e., what is a correct triple and what is not? Ideally, we would like to be able to measure also how correct the data is, and evaluate if the data that is no longer accessible (if using only the SKP property set) is correct and useful data, or perhaps mostly consist of noise. However, we believe that crowdsourcing efforts such as the DBpedia Data Quality Evaluation launched, may be able to provide evaluation datasets that makes this feasible.

5.2 Range Extraction

For evaluating the range extraction method, which had to be done manually, a set of SKPs were selected (among the 34 we initially generated, corresponding to the QALD query classes). Unfortunately due to lack of evaluators, we were not able to evaluate the complete set of 34 SKPs, but had to focus on 8 SKPs that were randomly selected but where we made sure to cover both "small" and "large" SKPs (in terms of number of properties and range axioms). Using different cutoff thresholds for the inclusion of range classes, all the resulting proposals for range axioms were manually assessed by three evaluators (each range axiom was evaluated by at least 2 evaluators). The evaluators were asked to assess if the range class could be considered correct or not, in the context of the particular SKP main class, and for the property at hand. Initially, the evaluators simply assessed if the range class was correct or not (an "unsure" alternative was also available), but in addition, if deemed correct the evaluators were also asked to assess the level of abstraction of the range class. The latter, to evaluate if the method used was able to arrive at range classes that are neither too specific nor too general.

For instance, consider the SKP `Actor`, where the main class is `dbpedia:Actor`. This SKP includes a property `dbprop:spouse`, which relates an actor to his or her spouse. One class that is extracted as being part of the property range is the `dbpedia:Actor` class. However, despite this being a common type of the objects, it is not an appropriate range class – it is more of a coincidence that most actors are actually married to other

actors, rather than a general axiom. A more appropriate class to include would be a superclass of `dbpedia:Actor`, i.e., `dbpedia:Person`. On the other hand, more general is not always better. Consider the superclass of `dbpedia:Person`, which is `dbpedia:Agent` (a class that also includes subclasses such as `dbpedia:Organisation`). This would not be an appropriate class either, since there are agents, e.g., companies, that cannot be the spouse of an actor. Through this example, we note that there is often a level of abstraction that is the most appropriate for expressing the range axioms, although more specific or more general classes cannot be considered as "wrong".

To combine the results of the three evaluators we have classified something as correct if at least one evaluator considered it correct, and the others either agreed that it was correct or were not sure. We have classified something as incorrect if, on the contrary, one evaluator considered it incorrect, and the others either agreed or were not sure. If the evaluators disagreed, e.g., one considering it correct and one incorrect, or they agree on the "unsure" alternative, the combined result is classified into the "unsure" category.

In Figure 2 we can see the results of the correctness evaluation of range axioms. On average, for each SKP, the method is able to find an appropriate range (one or more classes) for about 8 properties that were to be included in the SKP but that previously had no range axioms. In the figure we can see that for a cutoff threshold of 0.1 (meaning that a range class is included if it is the assigned type of more than 10% of the objects in triples using this specific property, and that are covered by this SKP) already around 80% of the proposed range classes are deemed as correct by the evaluators. This fraction increases as the cutoff threshold is raised, and at a threshold of 0.5 it is about 87%. As can be seen, the fractions of incorrect (and unsure) range classes stays well below 10% for a threshold of 0.3 and higher, and even before that the maximum fraction of incorrect suggested ranges is only about 12%.

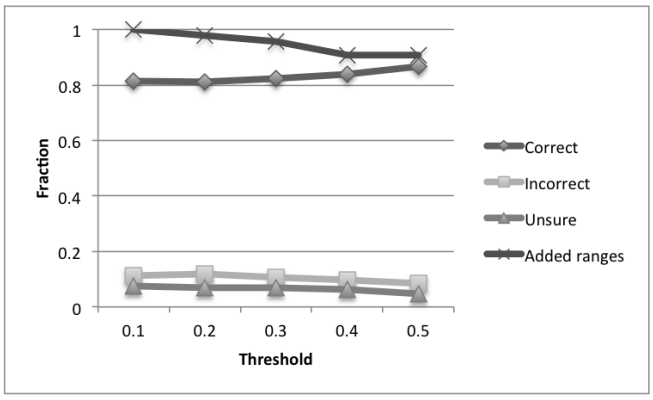


Fig. 2. Correctness of new range axioms, and fraction of properties that still receive a range axiom as threshold increases.

However, this increase in precision comes at a price of fewer suggested range axioms. In the figure we have therefore included also the "loss" of range axioms, in terms of the

fraction of the properties where (correct) range axioms were proposed at threshold 0.1, but which when the threshold is increased no longer will have any range axiom in the SKP (called "Added ranges" in the diagram). When increasing the threshold above 0.3, this drop starts to become significant, e.g. going from 96% at the 0.3 threshold down to 91% at 0.4.

An additional drawback when raising the threshold, which is not directly visible in the figure, is the level of abstraction of the included range classes. In general, the agreement between evaluators is quite poor when it comes to evaluating the level of abstraction, and it varies quite a lot between the 8 SKPs that were assessed, hence, we do not provide any numerical results of this part of the evaluation. Instead, based on the cases when the evaluators do agree, and the trends in their individual assessments, we try to summarise some tendencies. The trend is that as the threshold increases, the first (correct, but not necessarily appropriate with respect to abstraction level) range axioms that are removed seem to be the ones that are considered too specific (c.f. `dbpedia:Actor` in the example above) by at least some evaluator. However, continuing to further increase the threshold, i.e., from 0.4 and onwards, seems to remove a significant amount of (agreed on) appropriate range classes as well as the overly general ones, hence, increasing the threshold too much seems to come with too much negative side-effects in terms of increasing the fraction of overly general range classes compared to the appropriate ones.

Based on these results, we conclude that, both from the perspective of including as many correct range axioms as possible without introducing too many errors, and from the (somewhat inconclusive) indications on appropriate level of generality, a selection threshold around 0.3 seems to be a reasonable pick. This threshold has been used for generating the SKPs in the current catalogue.

6 Discussion

Originally, the notion of Ontology Design Patterns (ODPs) referred solely to a top-down view on modelling best practices, and constituted manually designed patterns representing those best practices. More recently, however, the more general notion of KPs has been proposed, and such patterns have also been created in a bottom-up fashion, i.e., representing the way information on the Web or Linked Data is actually represented, rather than how it "should" be. It is highly relevant in this context to discuss the relation between best practices and patterns. Although we do agree that actual modelling patterns, found in data, do not necessarily conform to best practices, we also acknowledge that determining what is a "best practice" is very difficult. By investigating real-world data we observe actual practices, and by storing these as SKPs users are able to understand the current practice. For many use cases (e.g., querying or linking to data) it is more important to understand and adhere to *current practices*, rather than best practices that may not at all be used in the data at hand. Since our SKPs are dynamic, i.e., can be re-generated as soon as data changes, we envision that assuming data and model quality increases over time, the gap between best practices and actual practices is reduced.

Another general aspect of the SKPs that is worth mentioning is their generalisability over different datasets. Our experiments have so far been limited to DBpedia data, however, the method we are using is in no way restricted to this particular data. Although

DBpedia may be a particularly tricky dataset (due to its semi-automatic construction, and large coverage), we have observed that similar problems with duplicated properties and lack of ranges and other axioms do exist also in other datasets. However, the most interesting problem arises when starting to extract cross-dataset SKPs, which will be our next step. To find "synonymous" properties across vocabularies and datasets, and to be able to compare patterns between overlapping datasets is where we envision that the substantial benefits arise. The methods presented here are sufficiently general to be applied to this extended scenario with only minor modifications to the current implementation.

7 Conclusions and Future Work

KPs are more and more being extracted bottom-up, e.g., from Linked Data, rather than only being hand-crafted in a top-down fashion, e.g., as ODPs. This new kind of KPs is important since they can assist in making sense of datasets, and allow users and systems to formulate appropriate queries over data, while managing the diversity of properties used in datasets. Diversity of data representation, and lack of agreement on schemas and ontologies, is currently a major obstacle towards taking full advantage of the Semantic Web and Linked Data. Therefore, approaches like ours, for characterising and structuring data (e.g., by identifying synonymous properties and property ranges), are of essence.

This paper has provided an overview of our method for generating SKPs from Linked Data (details on the synonymy detection and property selection in [14,15]) focusing particularly on the final part; characterising the properties, e.g., through range axioms. Generally, SKPs can characterise classes from any reference ontology, by presenting their most frequent properties and property characteristics, based on analysing the underlying data. SKPs are stored as OWL ontologies but can be continuously updated in a completely automated fashion to reflect changes in the underlying data. We have exemplified the method by applying it to classes of the DBpedia ontology, and in particular we have thereby evaluated our method for extracting range axioms. Results show that by setting appropriate thresholds, SKPs can be generated that cover (i.e., allow us to query, using the properties of the SKP) over 94% of the triples about individuals of that class, while only needing to care about 27% of the total number of distinct properties that are used in the data. The range extraction method results in range axioms that are on average correct in 82% of the cases (merely 10% are clear errors), at the selected threshold level. These results clearly show that it is possible to make sense of data, and manage the diversity of Linked Data, by analysing the data and identifying the underlying patterns.

The catalogue of SKPs for the DBpedia classes is being published at the moment. While this will be an important resource, it is simply one example of a reference ontology that can be used. As future work we intend to publish the method described in the paper as a software component to be reused by others, over their dataset of choice. We also intend to extend the generated set of DBpedia-based SKPs, by taking into account other datasets that align to DBpedia, creating cross-dataset SKPs that can be used to formulate queries (and distribute queries) over several dataset. Another interesting line of future work is to use the SKPs in order to analyse data quality, similar to what is described for "key properties" in [1], by studying the triples that do not adhere to the pattern.

Acknowledgements

Part of this research has been sponsored by the EPSRC funded project LODIE: Linked Open Data for Information Extraction, EP/J019488/1.

References

1. Atencia, M., David, J., Scharffe, F.: Keys and pseudo-keys detection for web datasets cleansing and interlinking. In: Proc. of the 18th International Conference, EKAW 2012, Galway City, Ireland, October 8-12, 2012. LNCS, vol. 7603, pp. 144–153. Springer (2012)
2. Auer, S., Demter, J., Martin, M., Lehmann, J.: Lodstats - an extensible framework for high-performance dataset analytics. In: Proc. of the 18th International Conference, EKAW 2012, Galway City, Ireland, October 8-12, 2012. LNCS, vol. 7603, pp. 353–362. Springer (2012)
3. Basse, A., Gandon, F., Mirbel, I., Lo, M.: DFS-based frequent graph pattern extraction to characterize the content of RDF Triple Stores. In: Proc. of the WebSci10: Extending the Frontiers of Society On-Line, April 26-27th, 2010, Raleigh, NC: US [Online proc.] (2010)
4. Blomqvist, E.: Ontocase-automatic ontology enrichment based on ontology design patterns. In: Proc. of the 8th International Semantic Web Conference (ISWC 2009). LNCS, vol. 5823, pp. 65–80. Springer (2009)
5. Blomqvist, E., Sandkuhl, K.: Patterns in ontology engineering: Classification of ontology patterns. In: ICEIS 2005, Proc. of the Seventh International Conference on Enterprise Information Systems, Miami, USA, May 25-28, 2005. pp. 413–416 (2005)
6. Gangemi, A.: Ontology Design Patterns for Semantic Web Content. In: The Semantic Web ISWC 2005. LNCS, vol. 3729. Springer (2005)
7. Gangemi, A., Presutti, V.: Handbook on Ontologies, chap. Ontology Design Patterns. Springer, 2nd edn. (2009)
8. Gangemi, A., Presutti, V.: Towards a pattern science for the Semantic Web. Semantic Web 1(1-2), 61–68 (2010)
9. Musetti, A., Nuzzolese, A., Draicchio, F., Presutti, V., Blomqvist, E., Gangemi, A., Ciancarini, P.: Aemoo: Exploratory Search based on Knowledge Patterns over the Semantic Web (2011), [Finalist of the Semantic Web Challenge 2011]
10. Nuzzolese, A.G.: Knowledge Pattern Extraction and Their Usage in Exploratory Search. In: Proc. of the 11th International Semantic Web Conference (ISWC 2012). LNCS, vol. 7650, pp. 449–452. Springer (2012)
11. Nuzzolese, A.G., Gangemi, A., Presutti, V., Ciancarini, P.: Encyclopedic knowledge patterns from wikipedia links. In: Proc. of the 10th International Semantic Web Conference (ISWC 2011). pp. 520–536. LNCS, Springer (2011)
12. Presutti, V., Aroyo, L., Adamou, A., Schopman, B.A.C., Gangemi, A., Schreiber, G.: Extracting Core Knowledge from Linked Data. In: Proc. of the Second International Workshop on Consuming Linked Data (COLD2011), Bonn, Germany, October 23, 2011. vol. 782. CEUR-WS.org (2011)
13. Presutti, V., Blomqvist, E., Daga, E., Gangemi, A.: Pattern-based ontology design. In: Ontology Engineering in a Networked World, pp. 35–64. Springer (2012)
14. Zhang, Z., Gentile, A.L., Augenstein, I., Blomqvist, E., Ciravegna, F.: Mining equivalent relations from linked data. In: Proc. of the annual meeting of the Association for Computational Linguistics (ACL) 2013 (2013)
15. Zhang, Z., Gentile, A.L., Blomqvist, E., Augenstein, I., Ciravegna, F.: Statistical knowledge patterns: Identifying synonymous relations in large linked datasets. In: (To appear) Proceedings of ISWC2013. LNCS, Springer (2013)

Ontology Patterns: Clarifying Concepts and Terminology

Ricardo A. Falbo[1], Giancarlo Guizzardi[1,2], Aldo Gangemi[2], Valentina Presutti[2]

[1]Federal University of Espírito Santo, Vitória, Brazil
{falbo, gguizzardi}@inf.ufes.br
[2]ISTC, National Research Council, Italy
{aldo.gangemi,valentina.presutti}@cnr.it

Abstract. Ontology patterns have been pointed out as a promising approach for ontology engineering. The goal of this paper is to clarify concepts and the terminology used in Ontology Engineering to talk about the notion of ontology patterns taking into account already well-established notions of patterns in Software Engineering.

Keywords: ontology pattern, ontology design pattern, ontology engineering

1 Introduction

Although nowadays ontology engineers are supported by a wide range of ontology engineering methods and tools, building ontologies is still a difficult task even for experts [1]. In this context, reuse is pointed out as a promising approach for ontology engineering. Ontology reuse allows speeding up the ontology development process, saving time and money, and promoting the application of good practices [2]. However, ontology reuse, in general, is a hard research issue, and one of the most challenging and neglected areas of ontology engineering [3]. The problems of selecting the right ontologies for reuse, extending them, and composing various ontology fragments have not been properly addressed yet [4].

Ontology patterns (OPs) are an emerging approach that favors the reuse of encoded experiences and good practices. OPs are modeling solutions to solve recurrent ontology development problems [5]. Experiments, such as the ones presented in [4], show that ontology engineers perceive OPs as useful, and that the quality and usability of the resulting ontologies are improved. However, compared with Software Engineering where patterns have been used for a significant time [6, 7, 8], patterns in Ontology Engineering are still in infancy. The first works are from the beginning of the 2000s (e.g. [9, 10]), and only recently this approach has gained more attention, especially by the communities of Ontology Engineering [2, 3, 4, 5] and Semantic Web [1, 11].

In this paper, we discuss the notion of ontology pattern by means of an analogy to the notion of pattern in Software Engineering. A premise underlying the discussion made in this paper is that, for developing a domain ontology, an ontology engineer should follow an ontology development process that is quite similar to the software development process in Software Engineering. i.e., in our view, a domain ontology

should be developed following an ontology development process comprising activities such as ontology requirements elicitation, ontology conceptual modeling, ontology design, ontology implementation, and ontology testing.

This paper is organized as follows. In Section 2, we present some important pattern-related concepts as used in Software Engineering, and also the current view of patterns in Ontology Engineering. In Section 3, we revisit some notions related to ontology patterns by means of an analogy to patterns in Software Engineering. In Section 4, we show, by means of examples, how some types of ontology patterns can be used during the ontology development process. Finally, in Section 5, we present our final considerations.

2 Software Engineering Patterns and Ontology Patterns

Patterns, in general, are vehicles for encapsulating knowledge. They are considered one of the most effective means for naming, organizing, and reasoning about design knowledge. "Design knowledge" in this sentence is applied in a general sense, meaning design in several different areas, such as Architecture and Software Engineering. According to [12], "a pattern describes a particular recurring design problem that arises in specific design contexts and presents a well-proven solution for the problem. The solution is specified by describing the roles of its constituent participants, their responsibilities and relationships, and the ways in which they collaborate".

In Software Engineering, patterns help to alleviate software complexity in several phases of the software development process [13]. During the software development process, regardless of the method or process model adopted, there are some activities that should be performed, namely: requirements elicitation, conceptual modeling, architectural design, detailed design, and implementation. There are different types of patterns covering different abstraction levels related to these phases. Analysis patterns are to be used during conceptual modeling. They describe how to model (in the conceptual level) a particular kind of problem in an application domain. They comprise of conceptual model fragments that represent knowledge of the problem domain, and their goal is to aid developers in understanding the problem rather than showing how to design a solution. According to [14], there are two main types of analysis patterns: domain-specific and domain-independent analysis patterns. Domain-specific analysis patterns model problems that only appear in specific domains. They capture the core knowledge related to a domain-specific problem and, therefore, they can be reused to model applications that share this core knowledge. On the other hand, domain-independent analysis patterns capture the core knowledge of atomic notions that are not tied to specific application domains and, hence, can be reused to model the same notions whenever they appear in any domain. Patterns such as the ones proposed by Fowler [7] are examples of analysis patterns. Architectural patterns describe selected types of components and connectors (the generalized constituent elements of all software architectures) together with a control structure that governs system execution [15]. They can be seen as templates for concrete software architectures [8], and thus are to be used during the architectural design phase. Several of the patterns proposed in the Pattern Oriented Software Architecture (POSA) approach [8], and most of the

patterns presented in [16] are architectural patterns. Design Patterns provide a scheme for refining subsystems or components of a software system, or the relationships between them. They describe commonly-recurring structures of communicating components that solves general design problems within a particular context [6]. Design patterns are medium-scale patterns. They are smaller in scale than architectural patterns, but they are independent of a particular programming language (implementation-independent patterns) [8]. Moreover, they are used in the detailed design phase. The Gang of Four (GoF) patterns [6] are the most known examples of a design patterns catalog. Finally, idioms (or programming patterns) represent the lowest-level patterns. They are specific to a programming language (patterns at a source code level). An idiom describes how to implement particular aspects of components or the relationships between them, using the features of a given language [8]. Idioms are used in the implementation phase. Coplien's C++ patterns [17] are examples of idioms.

The Ontology Engineering community has also tackled the notion of patterns, especially for aiding developing domain ontologies. Domain ontologies aim at describing the conceptualization related to a given domain, such as electrocardiogram in medicine [18].

According to [3], an Ontology Design Pattern (ODP) is a modeling solution to solve a recurrent ontology design problem. Gangemi and Presutti [3] have identified several types of ODPs, and have grouped them into six families: Structural, Reasoning, Correspondence, Presentation, Lexico-Syntactic, and Content ODPs.

Structural ODPs include Logical and Architectural ODPs. Logical ODPs provide solutions for solving problems of expressivity, while architectural ODPs affect the overall shape of the ontology either internally or externally. Reasoning ODPs inform about the state of an ontology, and let a system decide what reasoning has to be performed on the ontology in order to carry out queries and evaluation, among others. Correspondence ODPs include Reengineering ODPs and Mapping ODPs. Reengineering ODPs provide solutions to the problem of transforming a conceptual model (which can even be a non-ontological resource) into a new ontology. Mapping ODPs are patterns for creating semantic associations between two existing ontologies. Presentation ODPs deal with usability and readability of ontologies from a user perspective. They are meant as good practices that support the reuse of ontologies by facilitating their evaluation and selection (e.g. naming conventions). Lexico-syntactic ODPs are linguistic structures or schemas that consist of certain types of words following a specific order, and that permit to generalize and extract some conclusions about the meaning they express. They are useful for associating simple Logical and Content ODPs with natural language sentences, e.g., for didactic purposes. Finally, Content ODPs are small fragments of ontology conceptual models that address a specific modeling issue, and can be directly reused by importing them in the ontology under development. They provide solutions to domain modeling problems [3].

As pointed by Gangemi [19], Content ODP can extract a fragment of either a foundational or a core ontology, which constitutes its background. Based on this fact, Falbo et al. [20] consider two types of Content ODPs: Foundational ontology patterns, which are extracted from foundational ontologies, and tend to be more generally applied, and Domain-related ontology patterns, which are domain-specific patterns, and thus are applicable to solve problems in specific domains.

The Manchester's Ontology Design Patterns Catalog [21] is a public catalog of ODPs focused on the biological knowledge domain. In this catalog, there types of ODPs are considered, namely [21, 22]: Extensional ODPs, which provide ways of extending the limits of the chosen knowledge representation language; Good practice ODPs, which are used to produce more modular, efficient and maintainable ontologies; and Domain Modeling ODPs, which are used to model a concrete part of the knowledge domain.

In contrast with the case of patterns in Software Engineering, patterns in Ontology Engineering are not properly linked to the development phase in which they can be applied. Moreover, it is important to highlight that the term "design" in "ontology design patterns" does not have the same meaning of "design" in "design patterns" of Software Engineering. In "ontology design patterns", the term "design" is applied in a general sense, meaning the creation (building) of the ontology. In Software Engineering, in the other hand, "design" refers to the software development phase in which developers cross the border from the problem space to the solution space. While requirements elicitation and analysis deal with the problem domain, aiming at understanding the problem to be solved and its domain, in the design phase, the focus is on providing a solution, what requires taking technological aspects into account. Conceptual models, built during requirements analysis, are only concerned with modeling a view of the domain for a given application, and thus are independent of the technology to be applied in the solution. Design models, on the other hand, are committed to translating the conceptual view to the most suitable implementation according to the underlying implementation environment and also considering a number of non-functional (technological) requirements (such as efficiency, usability, reliability, portability, etc.). Thus, designers should know a priori features of the implementation environment to properly address the non-functional requirements in a given solution. In fact, the same conceptual model can lead to several design solutions, and the design phase involves choosing the most adequate solution for the problem.

Guizzardi [23] defends an analogous process for Ontology Engineering. In an ontology conceptual modeling phase, a *reference domain ontology* should be produced, whose aim is to make a clear and precise description of the domain elements for the purposes of communication, learning and problem solving. Reference ontologies are to be used in an off-line manner to assist humans in tasks such as meaning negotiation and consensus establishment. In the design phase, this conceptual specification should be transformed into a design specification by taking into account a number of issues ranging from architectural issues and non-functional requirements, to target a particular implementation environment. The same reference ontology can potentially be used to produce a number of (even radically) different designs. Finally, in the implementation phase, an ontology design is coded in a target language to be then deployed in a computational environment. This implementation version is frequently termed an *operational ontology* [20]. Unlike reference ontologies, operational ontologies are not focused on representation adequacy, but are designed with the focus on guaranteeing desirable computational properties. A design phase, thus, is necessary to bridge the gap between the conceptual modeling of reference ontologies and the coding of them in terms of a specific operational ontology language (such as, for instance, OWL and RDFS, but other DL-based languages [24], Datalog-based languages [25], relational

databases [26], etc.). Issues that should be addressed in the design phase are, for instance: determining how to deal with the differences in expressivity of the languages that are used in each of these phases; or how to produce lightweight specifications that maximize specific non-functional requirements, such as reasoning performance. Figure 1 illustrates this Ontology Engineering view.

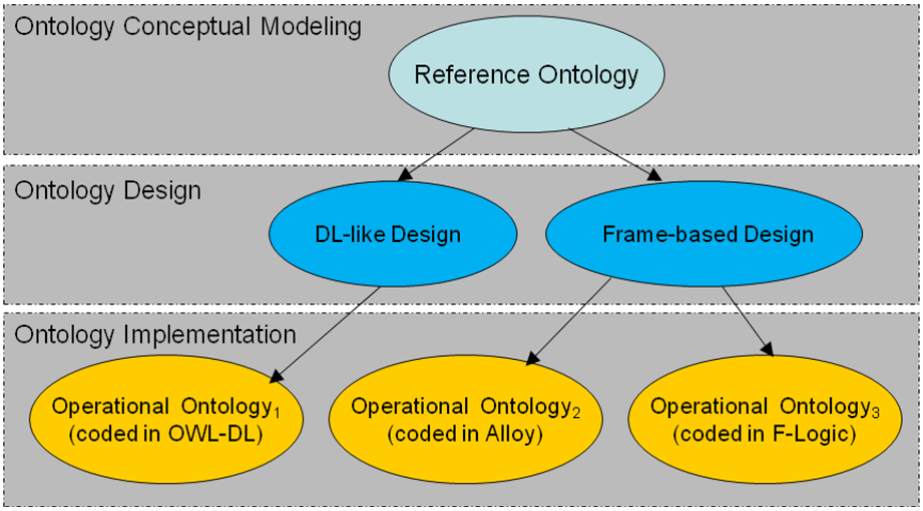


Fig. 1. Ontology Engineering as a Software Development Process

Based on this view of Ontology Engineering, in the next section, we revisit the notion of ontology patterns in order to clarify some concepts and the terminology used. Moreover, we compare the types of ontology patterns identified by Gangemi and Presutti [3] with the types of patterns related to the development phases in Software Engineering.

3 Ontology Patterns: Aligning Concepts and Terminology to Software Engineering Patterns

Once we have discussed a view of ontology development that is analogous to the well established view of software development in Software Engineering, we can revisit the notion of Ontology Pattern. Gangemi and Presutti [3] define Ontology Design Pattern as a modeling solution to solve a recurrent ontology design problem. As discussed in the previous section, "design" in this definition is used in a broad sense. In order to avoid confusion with the term "design" referring to the "design phase" of ontology development, we prefer to name patterns applied to Ontology Engineering as Ontology Patterns (instead of Ontology Design Patterns).

We can now further elaborate the definition of ontology pattern: *an Ontology Pattern (OP) describes a particular recurring modeling problem that arises in specific ontology development contexts and presents a well-proven solution for the problem*.

Taking this definition into account, we can now look at the types of ontology patterns identified by Gangemi and Presutti [3], and try to introduce them in another classification of ontology patterns that considers the ontology development phase depicted in Figure 1.

First, there are some types of patterns in [3] that, according to the definition presented above, do not qualify as OPs. They capture best practices, but do not conform to this definition. This is the case, for instance, of the lexico-syntatic and presentation patterns, which do not refer to a modeling problem. Mapping OPs also do not fit well to the definition above, since they do not address a modeling problem in a specific ontology development context. Mapping OPs are useful for integrating ontologies, and not for developing a new one. Although presentation OPs cannot be properly said to be ontology pattern, they have a counterpart in Software Engineering: naming conventions. In Software Engineering, there are name conventions that apply to different development phases. For instance, language-specific naming conventions, such as Java naming conventions, are to be applied during the implementation phase; naming conventions for classes, attributes and operations in general can be applied during conceptual modeling and design phases. Thus, analogously, ontology name conventions that are language-independent are to be applied during ontology conceptual modeling and design phases; language-specific ontology name conventions (such as OWL name conventions) are to be used during ontology implementation.

The Reengineering OP type is a case apart. Reengineering OPs are defined as transformation rules applied in order to create a new ontology (target model) starting from elements of a source model [3]. Based on this definition, we can say that they can be applied in several ontology development phases. However, most of the existing Reengineering patterns are language-dependent, such as patterns to transform non-OWL models to OWL DL operational ontologies, or refactoring patterns for refactoring an existing OWL DL source ontology into a new OWL DL target ontology. Such alleged patterns are, in fact, not proper patterns but Idioms.

In the other hand, content, architectural, logical and reasoning OPs can be related to ontology development phases. Content OPs are analogous to analysis patterns in Software Engineering; Architectural OPs to architectural patterns; and Logical and Reasoning OPs to design patterns, although some of them are, again, in fact, idioms.

Regarding the Manchester's Catalog, according to [21], "this catalog is generated from OWL files". Thus, its patterns better classify as Idioms.

Figure 2 shows a taxonomy of OPs, reorganizing some of the ontology pattern types identified in [3], and introducing subtypes of Content OPs (renamed as Conceptual OPs), as defined in [20].

Ontology Conceptual Patterns are fragments of either foundational ontologies (Foundational OPs) or domain reference ontologies (Domain-related OPs). They are to be used during the ontology conceptual modeling phase, and focus only on conceptual aspects, without any concern with the technology or language to be used for deriving an operational ontology. Ontology Conceptual Patterns are analogous to Analysis Patterns in Software Engineering. Foundational OPs are analogous to Domain-independent Analysis Patterns, while Domain-related OPs are analogous to Domain-specific Analysis Patterns.

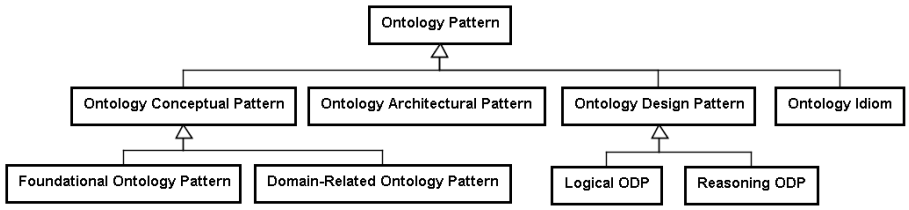


Fig. 2. Ontology Pattern Types

Foundational OPs (FOPs) are reusable fragments of foundational ontologies. Since foundational ontologies span across many fields and model the very basic and general concepts and relations that make up the world [18], FOPs can be applied in any domain. They are reused by analogy between the pattern and the problem in hand. An example of a FOP is the pattern for the problem of specifying roles with multiple disjoint allowed types, which were extracted from the ontology of substantial universals of the Unified Foundational Ontology (UFO) [27]. Another example is the appointment pattern, which were extracted from the ontology of social entities of UFO [28]. These two FOPs and how they can be reused are discussed in Section 4.

Domain-related OPs (DROPs) are reusable fragments extracted from reference domain ontologies. DROPs should capture the core knowledge related to a domain, and thus they can be seen as fragments of a core ontology of that domain. In contrast with FOPs, DROPs are reused by extension, i.e. concepts and relations of the pattern are specialized when the pattern is reused. In Section 4, we present a DROP for the domain of software processes and discuss its reuse by extension in the development of a software testing ontology.

Ontology Architectural Patterns are patterns that describe how to arrange an ontology (generally a large one) in terms of sub-ontologies or ontology modules, as well as patterns that deal with the modular architecture of an ontology network, where the involved ontologies play the role of modules [3]. These patterns can be used both during the conceptual modeling phase, and at the beginning of the ontology design phase. Since modularity is recognized as an important quality characteristic of good ontologies, we advocate for their use since the first stages of ontology development, for splitting the ontology into smaller parts, allowing tackling the problems one at a time. When applied at the beginning of the design phase, the purpose is to reorganize the ontology modules for addressing technological aspects, in special by taking non-functional requirements into account.

Ontology Design Patterns (ODPs) are patterns that address problems that occur during the ontology design phase. Based on the Gangemi and Presutti's taxonomy of types of ontology patterns [3], we identified two main types of ODPs: logical and reasoning ODPs. Reasoning ODPs, as the name suggests, aims at addressing specific design problems related to improving reasoning with ontologies (and qualities related to reasoning, such as computational tractability, decidability and reasoning performance). Logical ODPs, in turn, regards problems related to the expressivity of the formalism to be used in ontology implementation. They help to solve design problems that appear when the primitives of the implementation language do not directly sup-

port certain logical constructs. Logical ODPs are extremely important for ontology design, since most languages for coding operational ontologies are not focused on representation adequacy, but are designed with the focus on guaranteeing desirable computational properties [23]. We should highlight, however, that many patterns that address reasoning and logical problems are, in fact, Ontology Idioms (or Ontology Coding Patterns), since they describe how to solve problems related to reasoning or to expressivity of a specific logical formalism (e.g. OWL). According to the notion defended here, ODPs are more widely applied than ontology idioms, since they address problems related to various languages that share the same characteristics. For instance, a pattern addressing the problem of how to express n-ary relation semantics by only using class and binary relation primitives is a Logical ODP. It applies to all logical formalisms that do not have a construct for representing n-ary relationships, but have constructs for representing classes and binary relations. A pattern addressing the problem of how to express n-ary relation semantics in OWL is an Ontology Idiom.

Figure 3 shows the relationships between the types of ontology patterns shown in Figure 2 and the ontology development activities, shown in Figure 1. In the next section, we illustrate some Ontology Conceptual Patterns, and discuss how they can be reused.

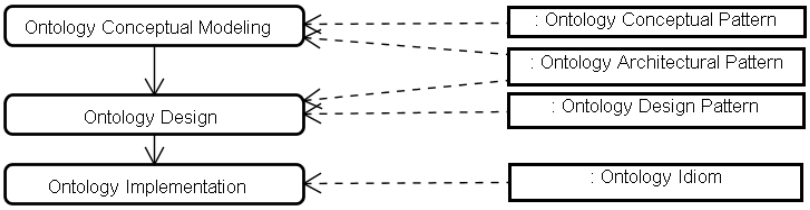


Fig. 3. Applicability of Ontology Patterns with respect to Ontology Development Phases

4 Reusing Ontology Conceptual Patterns

There are two main ways of reusing ontology patterns: by analogy and by extension. Moreover, several patterns can be reused when developing a domain ontology. Thus, pattern composition is also an important mechanism for combining patterns.

In reuse by analogy [29], with an ontology modeling problem at hands, we look for OPs that describe knowledge related to the type of situation we are facing. Once selected the pattern, we have to identify which concepts in our domain correspond to the concepts in the pattern, and we reproduce the structure of the pattern in the domain ontology being developed. Most of the OPs are reused by analogy (including most Ontology Architectural and Design Patterns, Idioms, but also Foundational Ontology Patterns (FOPs)). Domain-related OPs (DROPs), in turn, are typically reused by extension. In reuse by extension, the OP is incorporated in the domain ontology being developed, and it can be extended by means of specialization of its concepts and relations, and also by including new properties and relationships with the extended concepts.

Figure 4 shows the conceptual model of the FOP for the problem of specifying roles with multiple disjoint allowed types [27]. In this picture, the abstract class A is the *role mixin* that covers different role types. Classes B and C are the disjoint subclasses of A that can have direct instances, representing the *roles* (i.e., sortal, anti-rigid and relationally dependent types) that carry the principles of identity that govern the individuals that fall in their extension. Classes D and E are the ultimate *kinds* that supply the principles of identity carried by B and C, respectively. The association R represents the common specialization condition of B and C, which is represented in A. Finally, class F represents a type that class A is *relationally dependent* of. *Kind*, *Role* and *Role mixin* are concepts from the Unified Foundational Ontology (UFO), part A, an ontology of endurants. This pattern is also embedded as a higher-granularity modeling primitive in the ontology-driven conceptual modeling language OntoUML. For details, see [27].

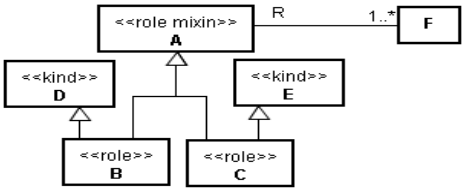


Fig. 4. Foundational OP for roles with multiple and disjunctive kinds [27].

Figure 5 shows two FOPs extracted from UFO-C, an ontology of social entities [28], namely: atomic/complex commitment types and appointments. The Atomic/Complex Commitment Types pattern, as the name suggests, models the distinction between atomic and complex commitments. According to UFO-C, *Commitments*, as intentional moments, inhere in *Agents*. Commitments can be atomic (*Atomic Commitment*) or complex (*Complex Commitment*). Complex commitments are composed of other commitments. The Appointment Pattern models a special type of commitment, named *Appointment* in UFO-C. An appointment is a commitment whose propositional content explicitly refers to a time interval.

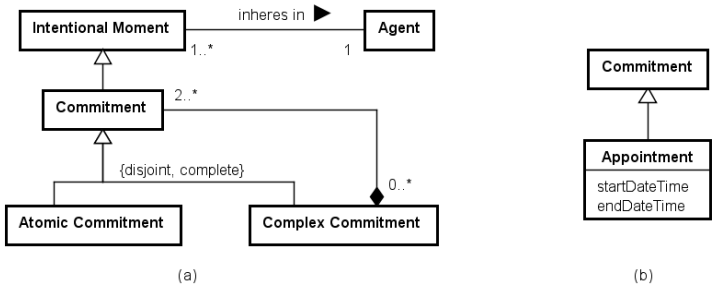


Fig. 5. (a) Atomic/Complex Commitment Types FOP; (b) Appointment FOP

In Figure 6, we apply the three aforementioned patterns when developing a domain ontology on the domain of Agendas. The Agenda Ontology was developed to support

the semantic integration of the Google Calendar API and the Google Contacts API to a Software Engineering Environment. In an agenda we are mainly interested in registering contacts and scheduled appointments. Contacts can be either organizations or people, which have different principles of identity. For addressing the modeling problem regarding contacts, we reuse the FOP presented in Figure 4; for addressing the modeling problem regarding appointments, we reuse the FOPs presented in Figure 5. The Agenda Ontology fragment produced applying these three FOPs is shown in Figure 6. In this model, the fragment that was solved by reusing the FOP for roles with multiple and disjunctive kinds (Figure 4) is shown detached in grey. As one can observe, the structure of the FOP is exactly applied to solve the problem in the Agenda Ontology.

Regarding the reuse of the Atomic/Complex Commitment Types FOP and the Appointment FOP, the structure of the models, although not exactly the same, is analogous. An *Appointment* in the Agenda Ontology is an *Appointment* in the sense of UFO-C. Like *Commitments*, *Appointments* can be classified as *Atomic* and *Complex Appointments* (see Atomic/Complex Commitment Types FOP). In the case of the Agenda Ontology, all complex appointments (said *Multiple Appointment*) are composed of atomic appointments (said *Single Appointment*). *Multiple Appointments* are further categorized according to their frequency of occurrence into: *Diary Appointment*, *Weekly Appointment* and *Monthly Appointment*. Since in the agenda domain we are not interested in commitments that are not appointments, we did not represent the concept of commitment. However, since appointment is a special type of commitment, appointments inhere in an agent (*Agenda Owner*).

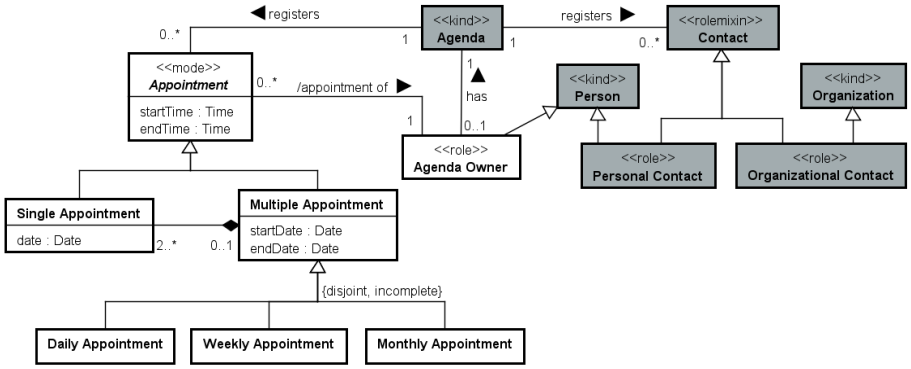


Fig. 6. A fragment of an Agenda Ontology.

Concerning the reuse of DROPs, which occur by extension, Figure 7 shows a fragment of the Reference Ontology on Software Testing (ROoST) [30], which were developed reusing DROPs organized as an ontology pattern language [20]. This picture shows a small (and simplified) fragment of ROoST dealing with artifacts produced and used by three software testing activities. This fragment of ROoST was built by composing and extending two DROPs: Work Product Participation (WPPA) pattern and Work Product Types (WPT) pattern. In Figure 7 the concepts from these patterns are shown detached in grey. As shown in this model fragment, concepts and relations

from the DROPs (concepts: *Activity Occurrence*, *Document* and *Code*; relations: *uses* and *produces*) are extended for the testing domain. *Test Execution*, *Test Coding* and *Test Case Design* are subtypes of *Activity Occurrence*. *Test Case* and *Test Result* are subtypes of *Document* (sub-kinds in UFO). *Test Code* is a sub-kind of *Code*, while *Code To Be Tested* is a role played by a *Code* when used in a *Test Execution*. Regarding relation specializations, *Test Case Design produces Test Case*. *Test Coding uses Test Case* and *produces Test Code*. Finally, *Test Execution uses Test Code* and *Code To Be Tested*, and *produces Test Results*.

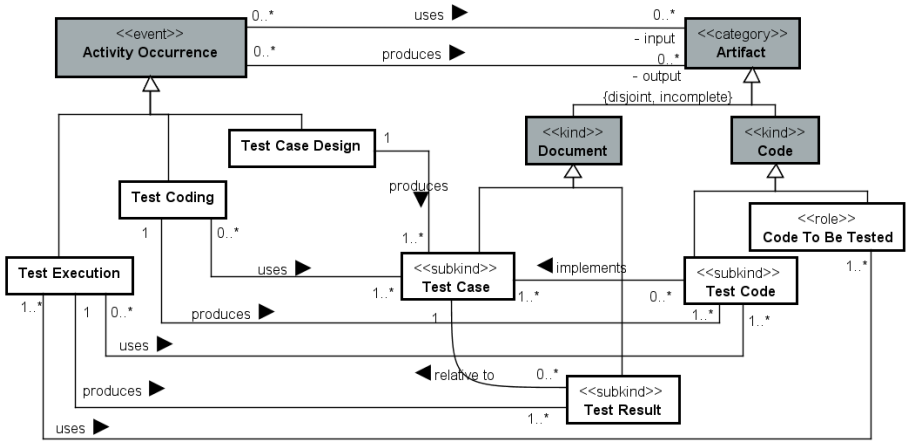


Fig. 7. A Fragment of the Reference Ontology on Software Testing

It is worthwhile to point out that new concepts and relations can also be added to the domain ontology being developed. In the case of ROoST model fragment shown in Figure 7, two relations were added: *Test Code implements Test Case*, and *Test Result is relative to Test Case*.

5 Final Considerations

Ontology patterns (OPs) are currently recognized as a beneficial approach for ontology development [3, 4]. In this paper we made an analogy between patterns in Software Engineering and patterns in Ontology Engineering, in order to clarify and harmonize the terminology used in both areas. Since patterns in Software Engineering have already been studied and used longer than in Ontology Engineering, we revisited some notions in the latter. In particular, by revisiting the classification proposed by Gangemi and Presutti in [3], we propose another way of classifying OPs, which is strongly related to the ontology development phase in which they are to be used. Moreover, we discuss ways of reusing OPs, namely by analogy and by extension. Domain-related OPs are typically reused by extension; while the other types of OPs are typically reused by analogy. It is worthwhile to point out that a third way is complementary to both: patterns composition. By means of examples, we illustrated how

different ontology conceptual patterns can be reused for developing domain reference ontologies.

Acknowledgments. This research is funded by the Brazilian Research Funding Agencies FAPES (PRONEX # 52272362/11) and CNPq (#311578/2011-0).

References

1. Noppens, O., Liebig, T., Ontology Patterns and Beyond - Towards a Universal Pattern Language. In: Proceedings of the Workshop on Ontology Patterns (WOP 2009), Washington D.C., USA, 2009.
2. Poveda-Villalón, M., Suárez-Figueroa, M.C., Gómez-Pérez, A., Reusing Ontology Design Patterns in a Context Ontology Network. In: Second Workshop on Ontology Patterns (WOP 2010), Shangai, China, 2010.
3. Gangemi, A., Presutti, V., Ontology Design Patterns. In: Handbook on Ontologies, Second edition, Staab, S., Studer, R. (Eds.), Springer, 2009, pp. 221 - 243.
4. Blomqvist, E., Gangemi, A., Presutti, V. *Experiments on Pattern-based Ontology Design*. In Proceedings of K-CAP 2009, pp. 41-48. 2009.
5. Presutti, V., Daga, E., Gangemi, A., Blomqvist, E. *eXtreme Design with Content Ontology Design Patterns*. In: Proceedings of the Workshop on Ontology Patterns (WOP 2009), Washington D.C., USA, 2009.
6. Gamma, E., Helm, R., Johnson, R.E., Vlissides,J. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.
7. Fowler, M. *Analysis Patterns: Reusable Object Models*. Addison-Wesley Professional Computing Series, 1997.
8. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns,* John Wiley & Sons, 1996.
9. Clark, P., Thompson, J., Porter, B. Knowledge patterns. In: Proceedings of the 7th International Conference on *Principles of Knowledge Representation and Reasoning (KR 2000)*, pp. 591–600, San Francisco, 2000.
10. Reich, J.R. Ontological design patterns: Metadata of molecular biological ontologies, information and knowledge. Proceedings of the 11th International Conference on Database and Expert Systems Applications, DEXA 2000, pp. 698–709, London, 2000.
11. Svatek, V., Design Patterns for Semantic Web Ontologies: Motivation and Discussion. In: Proceedings of the 7th Conference on Business Information Systems, Poznan, Poland, 2004.
12. Buschmann, F., Henney, K., Schmidt, D.C., *Pattern-Oriented Software Architecture: On Patterns and Pattern Language*s, John Wiley & Sons Ltd, 2007.
13. Schmidt, D. C., Fayad, M., Jonhson, R.E., Software Patterns, Communications of the ACM, vol. 39, no. 10, October 1996.
14. Hamza, H., Mahdy, A., Fayad, M.E., Cline, M. Extracting domain-specific and domain-independent patterns. In Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (OOPSLA '03), New York, USA, pp. 310-311, 2003.
15. Devedzic, V. Software Patterns, In: Handbook of Software Engineering and Knowledge Engineering Vol.2 -Emerging Technologies, S.K. Chang (Ed.), World Scientific Pub Co Inc., 2002.

16. Fowler, M., *Patterns of Enterprise Application Architecture*, Addison Wesley, 2003.
17. Coplien, J.O., Advanced C++ Programming Styles and Idioms, Addison-Wesley, 1992.
18. Guarino, N.: Formal Ontology and Information Systems. In: Guarino, N. (ed.) Formal Ontology and Information Systems, pp. 3–15, IOS Press, Amsterdam, 2008.
19. Gangemi, A. Ontology Design Patterns for Semantic Web Content, In: Proc. of the 4th International Semantic Web Conference – ISWC'2005, p. 262 – 272, Galway, Ireland, 2005.
20. Falbo, R. A., Barcellos, M.P., Nardi, J.C., Guizzardi, G. Organizing Ontology Design Patterns as Ontology Pattern Languages, 10th Extended Semantic Web Conference, Montpellier, France, 2013.
21. Ontology Design Patterns (ODPs) Public Catalog. http://www.gong.manchester.ac.uk/odp/html/. Last access in 2nd September 2013.
22. Aranguren, M.E., Antezana, E., Kuiper, M., Stevens, R. Ontology Design Patterns for bio-ontologies: a case study on the Cell Cycle Ontology. BMC bioinformatics, 9(Suppl 5):S1, 2008.
23. Guizzardi, G., On Ontology, ontologies, Conceptualizations, Modeling Languages and (Meta)Models, In O. Vasilecas, J. Edler, A. Caplinskas (Org.). *Frontiers in Artificial Intelligence and Applications, Databases and Information Systems IV*. IOS Press, Amsterdam, 2007.
24. Guizzardi, G., Zamborlini, V., A Common Foundational Theory for Bridging two levels in Ontology-Driven Conceptual Modeling. In: 5th International Conference of Software Language Engineering (SLE 2012), Dresden. Germany, 2012.
25. Angele, J., Kifer, M., Lausen, G., Ontologies in F-Logic, In: Handbook on Ontologies, Second edition, Staab, S., Studer, R. (Eds.), Springer, 2009, pp. 45 – 70.
26. Silbernagl, D., Reasoning with Ontologies in Databases: including optimization strategies, evaluation and example SQL code, VDM Verlag, 2011.
27. Guizzardi, G. *Ontological Foundations for Structural Conceptual Models*, Netherlands: Universal Press, 2005.
28. Guizzardi, G., Falbo, R.A., Guizzardi, R.S.S. Grounding software domain ontologies in the Unified Foundational Ontology (UFO): the case of the ODE software process ontology. In: Proceedings of the XI Iberoamerican Workshop on Requirements Engineering and Software Environments, IDEAS'2008, Recife, Brazil, pp. 244-251, 2008.
29. Rittgen, P., Translating Metaphors into Design Patterns, In: Advances in Information Systems Development, Springer, pp. 425 – 436, 2006.
30. Souza, E.F., Falbo, R.A., Vijaykumar, N.L., Using Ontology Patterns for Building a Reference Sofware Testing Ontology, The 8th International Workshop on Vocabularies, Ontologies and Rules for the Enterprise (VORTE 2013), Vancouver, Canada, 2013.

Reasoning Performance Indicators for Ontology Design Patterns

Karl Hammar

Jönköping University
P.O. Box 1026
551 11 Jönköping, Sweden
karl.hammar@jth.hj.se

Abstract. Ontologies are increasingly used in systems where performance is an important requirement. While there is a lot of work on reasoning performance-altering structures in ontologies, how these structures appear in Ontology Design Patterns (ODPs) is as of yet relatively unknown. This paper surveys existing literature on performance indicators in ontologies applicable to ODPs, and studies how those indicators are expressed in patterns published on two well known ODP portals. Based on this, it proposes recommendations and design principles for the development of new ODPs.

Keywords: OWL, Ontology Design Pattern, reasoning, performance

1 Introduction

The Semantic Web is built upon, and depends on, the use of OWL DL ontologies. The adoption rate of such ontologies among practitioners is as of yet limited. Ontology Design Patterns (hereafter ODPs) attempt to simplify ontology development for everyday users, by packaging recurring ontology problems, along with recommended solutions, as recipes or small reusable building blocks [3]. By providing users with such ready-made solutions to common design problems, ontologies can be developed with less risk of inconsistencies and errors [2], a key factor if uptake of ontology technology is to be improved. Since their introduction in 2005, more than 170 Ontology Design Patterns have been published on the two most prominent ODP portals on the web[1]. Patterns and pattern-based ontology engineering methods have been the topic of a number of research projects and publications, e.g., pattern typologies [12], agile ontology development with patterns [11], pattern-based ontology learning [1], ontology transformation [13].

However, less effort has been spent on studying the effects of ODP design on reasoning performance over resulting ontologies. Certain structures occurring in Ontology Design Patterns for meronomy modelling have been shown to impact reasoning performance [9], but which other commonly used ODP features or

[1] http://ontologydesignpatterns.org/ and http://odps.sourceforge.net

structures that affect performance characteristics is as of yet unknown. The importance of establishing such a list of performance-affecting indicators becomes obvious when studying use cases in which reasoning with semantic technologies is performed (complex event processing with stream reasoning, ubiquitous computing scenarios, etc.). In many of these cases, immediate or rapid system responses are critical requirements. Consequently, the ontologies employed must be designed to provide appropriate reasoning performance characteristics.

While obtaining a better understanding of performance-altering structures in ODPs is an important goal in itself, if the work is to provide practical recommendations on the use of published ODPs, one needs also study how the developed performance indicators appear in those ODPs commonly used by the community. With this in mind, the following research questions were selected for study:

1. Which existing performance indicators from literature known to affect the performance of reasoning with ontologies are also applicable to ODPs?
2. How do these performance indicators vary across published ODPs?
3. Which recommendations on reasoning-efficient ODP design can be made, based on the answers to the above two questions?

2 Method

In order to answer the research questions, a two-step approach was employed. To begin with, a literature study on existing ontology performance indicators that are reusable in describing Ontology Design Patterns was performed, the results of which answer the first research question. Thereafter, the distribution of these indicators over published Ontology Design Patterns from two ODP portals was studied, in order to answer the second question. In the end, recommendations for ODP developers based on the findings of both these steps were developed.

In the literature study, publications at the main tracks and the associated workshops of four high-impact conferences dealing with formal knowledge modelling, from 2005 to 2012, were studied. The conferences in question were the International and Extended Semantic Web Conferences (ISWC and ESWC), the International Conference on Knowledge Capture (K-CAP), and the International Conference on Knowledge Engineering and Knowledge Management (EKAW).

All papers matching the above criteria were downloaded, and their abstracts studied. Abstracts mentioning metrics, indicators, language expressivity effects, classification performance improvements or performance analyses (in total, 16 papers) were selected for thorough reading. Of these, eight were found to identify performance-altering structures likely to exist or be relevant in Ontology Design Patterns. These papers and their contributions are discussed in Section 3.1.

The second research question concerned the study of how performance-altering indicators varied among the patterns available in the pattern repositories used by the community. To this end, the reusable OWL building blocks of the patterns from two well known ODP repositories, `http://ontologydesignpatterns.org`

and `http://odps.sourceforge.net`, were downloaded and studied. A modular expandable tool for measuring ontology or ODP metrics was developed[2] specifically for this purpose. The Java-based tool parses an input ontology (or in this case, ODP module) and based on which metric measurement plugins are located in the tool's classpath, measures different aspects of said ontology. It generates as output CSV data suitable for post-processing in a spreadsheet or statistics tool. Plugins for all of the performance related indicators under study (with two exceptions, detailed in Section 3.2) were developed for this tool, and it was then executed over the downloaded pattern set.

In analysing the data from the execution of the indicator measurements, a simple four step process was repeated for each indicator under study:

1. Sort all ODPs by the studied indicator.
2. Observe correlation effects against other indicators. Can any direct or inverse correlations be observed for whole or part of the set of patterns?
3. Observe distribution of values. Do the indicator values for the different patterns vary widely or not? Is the distribution even or clustered?
4. For any interesting observation made above, attempt to find an underlying reason or explanation for the observation, grounded in the OWL ontology language and established ODP usage or ontology engineering methods.

In performing the above analysis, some interesting correlations were discovered and studied, as shown in Section 3.2. In some cases, an explanation for the correlations based on the structure of the OWL language and the constructs within it could also be generated.

3 Findings

The below two sections summarise the key findings of the literature study and the subsequent indicator variance study.

3.1 Literature Review

In the studied papers, three main types of indicators and corresponding effects could be identified, namely expressivity profile indicators (i.e., indicators related to profiles or constraints of ontology language structures available for use), inheritance hierarchy structural indicators (i.e., indicators related to the structure of the subsumption tree), and axiom usage indicators (i.e., general indicators related to the logical axioms employed in an ontology). Each of these categories and the indicators found to be associated with them are discussed in the following subsections.

[2] `https://github.com/hammar/OntoStats`

Profile indicators Urbani et al. discuss the issue of scaling out description logic reasoning on parallel computing clusters using the MapReduce framework. They show in [15] that materialising the closure of an RDF graph using RDFS semantics can be performed using MapReduce, due to certain characteristics of the RDFS semantics. As shown in [14], the increased expressivity of OWL means that implementing such parallelisable scalable reasoning over datasets based on OWL ontologies is significantly more difficult than when using RDFS. Limiting themselves to ontologies within the OWL Horst fragment of OWL, the authors manage to work around these issues and present a resulting solution that enables reasoning with OWL Horst significantly faster than previous solutions [14].

In [7] Horridge et al. analyse the characteristics of the three OWL 2 profiles, OWL 2 RL, OWL 2 EL, and OWL 2 QL, and study the adherence to these profiles among ODPs published on the Web. The three profiles are subsets of OWL 2, developed by the W3C specifically for particular usages [16]. By limiting the semantics used, both in terms of actual axioms allowed and the positioning and use of those axioms, computational properties suitable to different uses are achieved. Horridge et al. [7] find that relatively few ODPs fit in these profiles, and that this may in part be due to modelling practices and recommendations (e.g., to always declare an inverse for an object property, or the use of cardinality restrictions where existential restrictions could be used instead).

Developing an ontology that lies solely within OWL Horst or one of the OWL 2 profiles, requires that no axioms exist in the ontology that lies outside of the target language restriction. Therefore, it is obviously important that ODP users be aware of the language profile of the patterns that they consider for reuse. Consequently, the profile indicators are highly relevant when describing ODP performance characteristics.

Structural indicators Kang et al. [8] perform a thorough evaluation of the effects of a number of different ontology metrics on performance in different commonly used reasoners. While most of their observations are on effects of axiomatic indicators, one interesting finding concerns the subsumption hierarchy. Kang et al. find that the indicator that they denote *tree impurity* has a measurable impact on reasoner performance, such that a high degree of tree impurity in an ontology correlates to slower reasoning over that same ontology. This tree impurity metric measures how far the ontology's inheritance hierarchy deviates from being a tree, by calculating how many more `owl:subClassOf` axioms are present in the ontology than are needed to structure a pure tree. Kang et al. [8] find that tree impurity has a clear negative impact on computational efficiency over an ontology. This finding mirrors the predictions of Gangemi et al. [4,5] regarding the computational efficiency effects of subsumption hierarchy tangledness, which they define as the number of classes in an ontology with multiple superclasses. While tree impurity and tangledness are measured differently, they both capture the same underlying design structure (that is, subsumption hierarchy deviation from a simple one-parent tree).

In [10], LePendu et al. study the characteristics of ontologies in the biomedicine domain. One of the metrics studied, and found to have a high impact on materialisation performance, is the depth of the subsumption hierarchy. They note that for every asserted instance of a subclass, all of the logic axioms pertaining to each and every superclass must also be calculated. For a shallow ontology, this may be a matter of just a few classes before the top level class is reached. For a deeper ontology however, this may be a quite significant amount of entailments that need to be computed. Kang et al. [8] also study the depth indicator, and like LePendu et al. find that it contributes to slower reasoning performance.

The structure of the subsumption hierarchy (both depth-wise and in terms of tree impurity/tangledness) is often affected by extensive pattern use. Given that a common pattern usage method involves importing and subclassing a reference building block, and given that patterns often build upon and refine one another such that this usage method is repeated, extensive pattern usage may quickly lead to an increased ontology depth or tangledness.

Axiomatic indicators The majority of performance-affecting indicators discussed in the studied literature concerns the use of particular types of axioms or structures in an ontology. Two papers in particular contribute to this knowledge, namely Goncalves et al. [6] and the aforementioned work by Kang et al. [8].

Goncalves et al. [6] investigate performance variability in ontologies, and details a developed method for isolating performance-degrading sections of ontologies, by the authors denoted "hot spots", for different reasoners. The removal of hot spots were found to cut reasoning times by between 81 and 99 %. As a side effect of their work, the authors notice that the removal of hotspots correlate with the removal of General Concept Inclusions, GCIs, from the ontologies. GCIs are subclass or equivalency axioms that have a complex class expression on their right hand side, for instance `(HeartDisease and hasLocation some HeartValve) SubClassOf CriticalDisease`. These results suggest that the number of GCI axioms in an ontology are useful as indicators of reasoning performance. Given the relatively high impact of the hot spots/GCIs seen by Goncalves et al. [6], and given that the existence of a GCI axiom in a single pattern could give rise to many such hot spots if the pattern is instantiated multiple times, it is important to study the prevalence of this type of modelling in ODPs.

As mentioned above, Kang et al. [8] evaluate performance effects of a number of metrics. They find four indicators that show a measurable performance-altering effect and that can easily be applied to ODPs also:

- Existential quantifications – the number of existential quantification axioms in an ontology or ODP. This is easiest measured by counting the number of `ObjectSomeValuesFrom` axioms in the ontology.
- Cyclomatic complexity – inspired by the same metric as used in software engineering complexity calculations, this indicator measures the number of linearly independent paths through the RDF graph, including not only subclass relations but any directed edges, which a reasoner needs to traverse in classifying said graph.

– Class in-degree – the average number of incoming edges to classes in the ontology. This gives an indication as to how interconnected an ontology is.
– Class out-degree – the inverse of the above indicator, i.e., the average number of outgoing edges to classes in the ontology.

Summary The performance-altering indicators found via the literature study, and further examined in the following section, are summarised in Table 1.

Table 1. ODP performance indicators found via literature study.

Indicator	Source
Average class in-degree	[8]
Average class out-degree	[8]
Cyclomatic complexity	[8]
Depth of inheritance	[10]
Existential quantification count	[8]
General concept inclusion count	[6]
OWL Horst adherence	[15,14]
OWL 2 EL adherence	[7,16]
OWL 2 QL adherence	[7,16]
OWL 2 RL adherence	[7,16]
Tree impurity / Tangledness	[8,4]

3.2 Indicator Variance in ODP Repositories

The results of the study of indicator variance are detailed below, with the exception of a few indicators that were not included, namely cyclomatic complexity, and the set of OWL profile adherence indicators. It proved practically infeasible to develop software for reliably measuring the former, and rather than make assertions based on possibly inexact data, it has been left out of this analysis. The latter set of indicators has as mentioned above been discussed extensively in Horridge et al.[7], to which the interested reader is referred. In total, 104 patterns were studied, with an average size in number of classes of about seven, and in number of properties about 15.

Average class in-degree The values for the average class in-degree indicator vary between 0.75 and 8, with a median value of 2.39 and an average value of 2.6. The distribution of indicator values over the whole pattern set is shown in Figure 1. The large majority of patterns (93 %) have a class in-degree of less than four, whereas a small group of patterns differ quite significantly and have an average value of around six.

Comparing some of the patterns exhibiting high and low values for the average class in-degree indicator, it was observed that they tended to differ in terms of the number of object properties contained within the patterns. The patterns exhibiting a high level of class in-degree seemed to contain a larger number of object properties than those patterns displaying a low level of this indicator. To

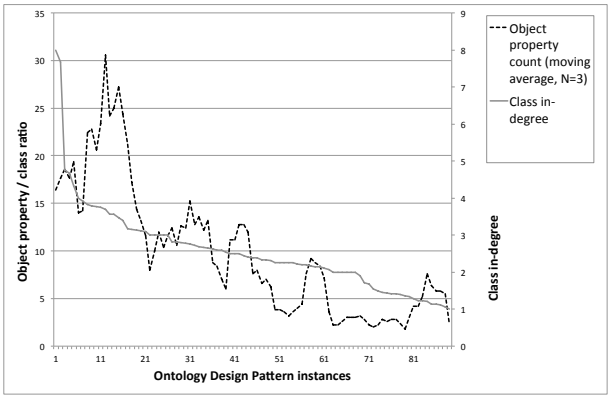


Fig. 1. Class in-degree and object property per class distributions

test whether this held for the entirety of the pattern set, the object property counts were mapped against the values of the class in-degree indicator. Such a mapping should if the observation holds indicate the existence of an correlation between the two mapped indicator value series. The results, as shown in Figure 1, while not indicating a correlation of the indicators across the entire studied pattern set, does indeed indicate that the patterns towards the high end of the spectrum in terms of class in-degree also often contain a higher number of object properties than the patterns with a lower class in-degree.

A possible explanation for this observation is the use of domain and range definitions on many object properties in patterns with high average class in-degree. It is considered good practice to establish such definitions for properties one adds to an ontology. However, each domain or range definition gives rise to one incoming RDF edge to the class in question, raising the average class in-degree indicator. Based on this observation, a recommendation to the effect of limiting the number of domain and range definitions used in performance-dependent ontologies can be made. However, there is likely to also be other as yet unknown causes beside domain and range definitions that that give rise to high average class in-degrees, for which reason ODP users and developers are recommended to limit the use of structures that needlessly raise class in-degree.

Average class out-degree The values for the average class out-degree indicator vary between 1 and 3.83, with a median value of 2.75 and an average of 2.64. The distribution of indicator values over the whole pattern set is shown in Figure 2. The reason why all patterns exhibit a value of at least one is simply that all defined classes by definition have at least one outgoing `subClassOf` edge to another class.

In studying some patterns displaying low or high values, it was observed that the patterns displaying a higher value seem to be patterns in which class

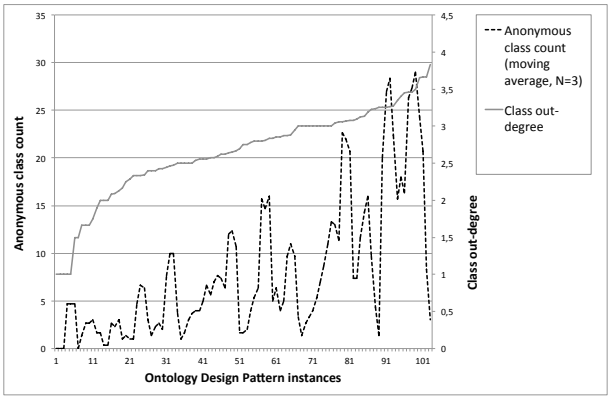


Fig. 2. Class out-degree and Anonymous class count distributions

restrictions are used extensively. Such restrictions are written as a class being asserted to be either a subclass of or equivalent to a restriction axiom, which would explain this observation – each `subClassOf` or `equivalentClass` axiom adds an outgoing edge, increasing the value of the indicator. To test whether this explanation is supported by further evidence, the number of anonymous class definitions (i.e., restrictions) were plotted against the value of the class out-degree indicator. The results are presented in Figure 2 which indicates a possible, if slight, correlation between class out-degree and anonymous class count.

Since the presence of class restrictions can, in the author's experience, help guide novice users understand the purpose of classes in an ontology or ODP, this unexpected performance-related effect of using such restrictions is of particular interest. Also, given the variation of this indicator's values over the studied set of published ODPs, a recommendation as to limiting its use in performance-critical ontology applications is made.

Depth of inheritance Due to the difficulty of measuring the inferred indicators across the transitive import closure graph of an ODP using the tools and APIs available at the time of writing, the values below were only calculated over the asserted depths of patterns, excluding imports. Moreover, as even this is quite a difficult task (due to different practices on how subclass relations to the top-level `owl:Thing` class are expressed), certain simplifications had to be made. These simplifications include the assumption of a subclass relation to Thing if no other superclass is asserted within the particular OWL file.

The subsumption hierarchy depth of the patterns varies from 0 to 5.3, with a median value of 1.5 and an average value of 1.7. In other words, most of the patterns are not very deep. At the bottom end of this distribution is a fairly large group (38 of 103 patterns) that have a depth of one or less. This particularly shallow group appears to consist of two types of patterns. The first type

consist of simpler domain specific vocabularies that do not employ much expressive logics, but rather act as schemas for simple datatypes that may be reused. Examples include patterns for species habitats, invoices, etc. The second type consist of very general patterns that define abstract or intangible phenomena without going into specific details. Examples include patterns modelling phenomena like participation and situation. A large part of the latter group seems to result from refactoring of top-level ontologies like DOLCE, whereas many of the patterns in the former group seem to be developed for more concrete and applied purposes. The patterns from the `http://odp.sourceforge.net` repository are generally deeper (with an average depth of 3.29) than those from the `http://ontologydesignpatterns.org` portal. However, the latter patterns generally contain more example classes that would likely be removed before instantiation in real cases, reducing this difference.

While ontology depth is associated with poor performance as discussed by LePendu et al. [10], the observations above indicate that very shallow patterns are often either lacking in specificity or logic expressivity, which implies that they may not be suitable for representing many types of medium-complexity situations in which patterns may be more useful. The recommendation here is for ODP developers to not shy away from subsumption hierarchy depth if needed for modelling the concepts of a domain, but to otherwise avoid constructing patterns that cause excessively deep ontologies.

Existential quantification count About half the patterns, 51 of 103, contain no explicit existential quantification axioms. If cardinality restrictions are rewritten into semantically equivalent existential restrictions as suggested in [7], the number of patterns containing no existential quantification axioms drops to 43. Of the 60 patterns that contain such axioms half, 31, contain one or two existential quantification axioms each. Studying a number of such patterns it was observed that the axioms are used sparingly and only when required.

However, in studying the patterns that contained a higher number of existential quantification axioms (i.e., three or more, as seen in 29 of the patterns), it was observed that these axioms were sometimes used in seemingly unneeded ways. For instance, subclasses restating such axioms as were already asserted on their superclasses, and existential quantification used to assert the coexistence of two individuals where it seems one individual might well exist on its own. These observed suboptimal uses of computationally expensive existential quantification axioms motivates a recommendation on limiting their use – if pattern users wish to add such axioms to restrict their model, the recommended axioms can instead be included in pattern documentation.

General concept inclusion count GCIs were not displayed by any of the studied ODPs. The author believes that this is because such constructs are generally not well supported by ontology engineering tools such as Protégé or TopBraid Composer. The lack of any patterns displaying values for this indicator implies that the performance effects of the use of this type of structure in ODPs

is negligible in practice. All the same, a recommendation can be made to the effect of limiting the use of these structures when developing new patterns.

Tree Impurity / Tangledness Due to the mentioned inconsistencies in how subclass relations to the `owl:Thing` concept are modelled, the tree impurity indicator is difficult to measure in a reliable manner. However, as mentioned, this indicator measures the same underlying structure as the tangledness indicator, i.e., how far an ontology inheritance hierarchy deviates from being a tree with one parent class per subclass. Therefore, observations regarding tangledness variation in the dataset should carry over to the tree impurity indicator also. Of the 103 studied patterns, only three display any degree of directly asserted tangledness at all. In all three of these cases, the number of multi-parent classes in the pattern was one. It appears that the use of asserted multiple inheritance in ODPs is rare. However, it should be noted that the number of *inferred* multi-parent classes may be significantly greater than this number. While inferred tangledness has for technical reasons been infeasible to measure in this study, its effect on the performance of reasoning may be considerable.

4 Discussion

Table 2 summarises the recommendations presented in earlier sections on how to develop and apply ODPs for uses in which reasoning performance matters. These recommendations suggest some design principles for ODP development and use:

- When developing patterns, *don't overspecify*. While an ODP creator's understanding of a domain may warrant adding domain and range restrictions to properties, or to implement some existential or cardinality restriction (because that is how the real world concept being modelled actually works), doing so will possibly have detrimental performance effects. Ensure that requirements on the ODP are made explicit, and implement only such axioms as are required to fulfill those requirements. Do not aim for further completeness for the sake of neatness.
- If the reusable OWL building block associated with a certain ODP does not display suitable reasoning characteristics, i.e., if it is overspecified, has a needlessly deep subsumption hierarchy, or is outside of a computation-friendly OWL 2 profile, *don't be afraid to rewrite it*. Many patterns encode a good practical solution to some modelling problem, which may be reused even if the associated OWL file is not directly suitable for one's purpose.
- The problem/solution mapping presented by a pattern may be more reusable than the OWL file building block provided with the pattern, as suggested above. Therefore, when when publishing a new Ontology Design Pattern, *ensure sufficient documentation exists* on the pattern's requirements, the problem that it solves, and how it solves that problem, such that users actu-

ally can reengineer the pattern if needed. Presently several published ODPs are only partially described, e.g., in the NeOn ODP portal [3].

Table 2. Recommendations on performance efficient ODP design.

Indicator	Recommendation
Average class in-degree	Avoid designs that give a high count of ingoing edges per node.
Average class out-degree	Avoid designs that give a high count of outgoing edges per node.
Class restrictions	Limit the use of class restrictions (i.e., enumerations, property restrictions, intersections, unions, or complements) to the minimum required by the ODP requirements.
Depth of inheritance	Avoid developing ODPs that cause a deep subsumption hierarchy.
Existential quantification count	Limit the use of existential quantification axioms to the minimum required by the ODP requirements. Even if the addition of such an axiom makes "real world" intuitive sense, first consider whether it is strictly necessary for the purpose of the ODP.
General concept inclusion count	Rewrite GCI axioms into property restrictions if possible. While such restrictions also cause reasoning performance effects, they are not known to be associated with the type of performance "hot spots" which GCIs can give rise to.
OWL Horst adherence	If the pattern is specifically intended to be used in ontologies which will be reasoned over using MapReduce, constrain the axioms allowed to the OWL Horst fragment of OWL.
OWL 2 EL adherence	If the pattern is specifically intended to be used in large ontologies, constrain the axioms allowed to the OQL 2 EL profile.
OWL 2 QL adherence	If the pattern is specifically intended to be used in ontologies and systems where query answering capability over large data sets is prioritised, constrain the axioms allowed to the OWL 2 QL profile.
OWL 2 RL adherence	If the pattern is specifically intended to be used in systems where predictable reasoning performance is prioritised, or rule engines used, constrain the axioms allowed to the OWL 2 RL profile.
Property domain and range restrictions	Limit the number of property domain and range definitions to the minimum required by the ODP requirements, as these may otherwise give rise to inefficient high class in-degree values.
Tree impurity / Tangledness	Avoid the use of multi-parent classes. In constructing class restriction axioms, avoid restrictions that give rise to inferred tangledness, i.e., axioms which give rise to unions or intersections of classes from different branches in the subsumption tree.

5 Conclusions

In this paper we have studied which published indicators of reasoning performance for ontologies that carry over to Ontology Design Patterns, and how those indicators are expressed in the ODPs published in two well-known pattern portals on the web. The results indicate that certain structures occurring in published ODPs can give rise to unfavourable performance when reasoning over ontologies built using these ODPs. Recommendations and design principles have been presented regarding trade-offs and prioritisations that ODP users and developers may need to make in employing or constructing Ontology Design Patterns for usage in systems where performance efficiency is of importance.

[3] `http://ontologydesignpatterns.org/`

References

1. Blomqvist, E.: Semi-automatic Ontology Construction based on Patterns. Ph.D. thesis, Linköping University (2009)
2. Blomqvist, E., Gangemi, A., Presutti, V.: Experiments on Pattern-based Ontology Design. In: Proceedings of the Fifth International Conference on Knowledge Capture. pp. 41–48. ACM (2009)
3. Gangemi, A.: Ontology Design Patterns for Semantic Web Content. In: The Semantic Web–ISWC 2005. pp. 262–276. Springer (2005)
4. Gangemi, A., Catenacci, C., Ciaramita, M., Lehmann, J.: Modelling Ontology Evaluation and Validation. In: The Semantic Web: Research and Applications. pp. 140–154. Springer (2006)
5. Gangemi, A., Catenacci, C., Ciaramita, M., Lehmann, J., Gil, R., Bolici, F., Strignano, O.: Ontology evaluation and validation. Tech. rep., Laboratory for Applied Ontology, ISTC-CNR (2005)
6. Goncalves, R.S., Parsia, B., Sattler, U.: Performance Heterogeneity and Approximate Reasoning in Description Logic Ontologies. In: The Semantic Web – ISWC 2012. pp. 82–98 (2012)
7. Horridge, M., Aranguren, M.E., Mortensen, J., Musen, M., Noy, N.F.: Ontology Design Pattern Language Expressivity Requirements. In: Proceedings of the 3rd Workshop on Ontology Patterns (2012)
8. Kang, Y.B., Li, Y.F., Krishnaswamy, S.: Predicting Reasoning Performance Using Ontology Metrics. In: The Semantic Web – ISWC 2012. pp. 198–214 (2012)
9. Lefort, L., Taylor, K., Ratcliffe, D.: Towards Scalable Ontology Engineering Patterns: Lessons Learned from an Experiment based on W3C's Part-whole Guidelines. In: Proceedings of the Second Australasian Workshop on Advances in Ontologies. pp. 31–40. Australian Computer Society, Inc. (2006)
10. LePendu, P., Noy, N., Jonquet, C., Alexander, P., Shah, N., Musen, M.: Optimize First, Buy Later: Analyzing Metrics to Ramp-up Very Large Knowledge Bases. In: The Semantic Web – ISWC 2010. pp. 486–501. Springer (2010)
11. Presutti, V., Daga, E., Gangemi, A., Blomqvist, E.: eXtreme Design with Content Ontology Design Patterns. In: Proceedings of the Workshop on Ontology Patterns (WOP), collocated with International Semantic Web Conference (ISWC) (2009)
12. Presutti, V., Gangemi, A., David, S., Aguado de Cea, G., Suárez-Figueroa, M.C., Montiel-Ponsoda, E., Poveda, M.: D2.5.1: A Library of Ontology Design Patterns: Reusable Solutions for Collaborative Design of Networked Ontologies. Tech. rep., NeOn Project (2007)
13. Svátek, V., Šváb-Zamazal, O., Vacura, M.: Adapting Ontologies to Content Patterns using Transformation Patterns. In: Workshop on Ontology Patterns (2010)
14. Urbani, J., Kotoulas, S., Maassen, J., Van Harmelen, F., Bal, H.: OWL reasoning with WebPIE: calculating the closure of 100 billion triples. In: The Semantic Web: Research and Applications. Springer (2010)
15. Urbani, J., Kotoulas, S., Oren, E., Van Harmelen, F.: Scalable Distributed Reasoning using MapReduce. In: The Semantic Web - ISWC 2009. Springer (2009)
16. W3C: OWL 2 Web Ontology Language Profiles (Second Edition), `http://www.w3.org/TR/owl2-profiles/`, checked on: 2013-02-27

Detecting Good Practices and Pitfalls when Publishing Vocabularies on the Web

María Poveda-Villalón[1], Bernard Vatant[2], Mari Carmen Suárez-Figueroa[1], Asunción Gómez-Pérez[1]

[1]Ontology Engineering Group. Universidad Politécnica de Madrid. Spain.
[2]Mondeca, Paris, France.

mpoveda@fi.upm.es, bernard.vatant@mondeca.com, {mcsuarez, asun}@fi.upm.es

Abstract. The uptake of Linked Data (LD) has promoted the proliferation of datasets and their associated ontologies bringing their semantic to the data being published. These ontologies should be evaluated at different stages, both during their development and their publication. As important as correctly modelling the intended part of the world to be captured in an ontology, is publishing, sharing and facilitating the (re)use of the obtained model. In this paper, 11 evaluation characteristics, with respect to publish, share and facilitate the reuse, are proposed. In particular, 6 good practices and 5 pitfalls are presented, together with their associated detection methods. In addition, a grid-based rating system is generated showing the results of analysing the vocabularies gathered in LOV repository. Both contributions, the set of evaluation characteristics and the grid system, could be useful for ontologists in order to reuse existing LD vocabularies or to check the one being built.

Keywords: ontology, vocabulary, linked data, ontology publication, ontology evaluation, pitfalls, good practices

1 Introduction

Vocabularies or Ontologies[1] bring their semantics to Linked Data (LD)[2] [3], by formally defining shared sets of classes and properties, using semantic standards such as RDFS or OWL. When a vocabulary element is used in a RDF dataset through its URI, nothing more is generally declared in this dataset about this element, and that is a good practice since datasets have not to re-define URIs already defined in external vocabularies. In order to understand the meaning of such an URI, both humans and applications should be able to de-reference it, discover the context in which it has

[1] At this moment, *there is no clear division between what is referred to as "vocabularies" and "ontologies"* (http://www.w3.org/standards/semanticweb/ontology). For this reason, we will use both terms indistinctly in this paper.

[2] http://www.w3.org/standards/semanticweb/ontology

been formally defined. This context is typically a RDFS or OWL file and the matching HTML documentation. Both files are, in the best of cases, available from the ontology URI through proper content negotiation implementation over HTTP protocol. Both human-readable and machine-consumable information should provide, not only the semantics of the elements defined in its namespace, but also a reasonable amount of metadata about the vocabulary (dates, creator, publisher, versions, etc.).

The Linked Open Vocabularies project (LOV)[3] is intended to gather and describe those vocabularies used or potentially usable by LD and to provide indicators of their relevancy. Each vocabulary in LOV is described by metadata gathered either from its formal publication, or from the vocabulary documentation or communication with the publishers, or from the vocabulary content itself. Two years after its launch, LOV has been widely acknowledged and embraced by the LD community.

A fundamental feature for the scope of our research is that each entry in LOV is uniquely identified by a vocabulary URI, and is generally associated with a unique namespace URI. Given the variety of interpretations and so many different implementation practices we have discovered in LOV, either OWL standards have underspecified the definition and relationship between those two URIs, or the specification has been largely either ignored or misunderstood. The simplest configuration is to have these two URIs being the same. But many other configurations are possible and are indeed observed. Moreover, content negotiation on the namespace does not necessarily leads to vocabulary URI.

An application dedicated to consume vocabularies is not likely to be prepared to such a variety of configurations. It is likely to identify the vocabulary by either or both the namespace URI or the vocabulary URI. Vocabulary publishing practices can be classified as "good" or "bad" insofar as they ease or impede such applications.

In this paper, we have conducted a detailed analysis of more than 350 vocabularies gathered in the LOV registry. Our aim is to automatize the detection of good practices and common pitfalls when publishing vocabularies in order to ease the work of applications willing to access and consume LOV vocabularies with no more initial information than the vocabulary URI, its namespace and the prefix assigned in LOV. The results of such scan is: (1) a non exhaustive list of best practices and common pitfalls about publishing LD vocabularies, (2) specific methods for detecting such good practices and common pitfalls, (3) some metadata about ontology quality (regarding the appearance or lack of good practices and pitfalls) that could be added to the vocabulary metadata stored in LOV, and (4) the inclusion of pitfalls in services such as OOPS![4] to help eager vocabulary managers to check the quality of their vocabularies prior to their publication.

The structure of the paper is the following. Section 2 introduces and describes the framework with the evaluation characteristics to be used in the evaluation of LOV vocabularies. Section 3 presents the detection methods implemented for checking the characteristics presented in Section 2. The results of executing such detection methods over 355 vocabularies registered in LOV and an analysis of the obtained results are shown in Section 4. Finally, Section 5 exposes related research efforts and Section 6 presents some concluding remarks and future lines of work.

[3] http://lov.okfn.org

[4] http://www.oeg-upm.net/oops

2 Good practices and pitfalls for publishing vocabularies

Main guidelines for publishing data over the web are the extremely well-known Linked Data principles and the Linked Open Data 5 Star rating system defined by Tim Bernes-Lee[5]. More precisely, the rating system defines the following levels (taken literally from the source):

- ***LOD1.*** *Available on the web (whatever format) but with an open licence, to be Open Data*
- ***LOD2.*** *Available as machine-readable structured data (e.g. excel instead of image scan of a table)*
- ***LOD3.*** *As (2) plus non-proprietary format (e.g. CSV instead of excel)*
- ***LOD4.*** *All the above plus, Use open standards from W3C (RDF and SPARQL) to identify things, so that people can point at your stuff*
- ***LOD5.*** *All the above plus Link your data to other people's data to provide context*

More specific recommendations about publishing ontologies on the web have been proposed inspired by the above-mentioned 5-star linked data scale. We will refer to it along this paper as the "Linked data vocabulary 5-start rating system"[6] that defines the following recommendations (taken literally from the source):

- ***LDV1.*** *Publish your vocabulary on the Web at a stable URI*
- ***LDV2.*** *Provide human-readable documentation and basic metadata such as creator, publisher, date of creation, last modification, version number*
- ***LDV3.*** *Provide labels and descriptions, if possible in several languages, to make your vocabulary usable in multiple linguistic scopes*
- ***LDV4.*** *Make your vocabulary available via its namespace URI, both as a formal file and human-readable documentation, using content negotiation*
- ***LDV5.*** *Link to other vocabularies by re-using elements rather than re-inventing*

Along the rest of the paper we will refer to the points stated in these two rating systems as LOD or LDV plus its ordinal numeration according to the lists above. We will use some of these points or recommendations to support the good practices and pitfalls proposed in this paper. We will also point to the 10 rules [1] for designing persistent URI, since some points are also applicable.

In the following, we describe the 11 characteristics we have identified when publishing ontologies on the Web. It should be noted that in the remaining the term "characteristics" will be used for referring to the set of both good practices and pitfalls. That is, there are 11 characteristics described here, 6 of them represent good practices and 5 of them represent pitfalls. Each characteristic has an identifier, a description and one example of an ontology holding that characteristic. The identifiers are on the form of GPX for good practices where the X is a numerical identifier, in this case starting in 1. For pitfalls, the identifiers are on the form of PY where Y is a numerical identifier. In this case, as the pitfalls here defined will be included in OOPS! catalogue[7], the numeration follows to the one given in the catalogue to avoid confusion and help the reader to find each pitfall both along this paper and within the

[5] http://www.w3.org/DesignIssues/LinkedData.html
[6] http://bvatant.blogspot.fr/2012/02/is-your-linked-data-vocabulary-5-star_9588.html
[7] http://www.oeg-upm.net/oops/catalogue.jsp

catalogue by the same identifier**.** For the examples, we refer to the vocabularies registered in LOV.

2.1 Good practices proposal

The following six characteristics represent our proposal of good practices in ontologies regarding publishing issues and metadata in an online ontology.

GP1. Provide RDF description: In order to make an ontology more reusable one should publish it on an stable URI (LDV1) providing machine-readable formats using open standards from W3C to identify things (LOD4).

- **Example:** the "Configuration ontology (cold)" ontology with URI http://purl.org/configurationontology provides a turtle serialization when looking up its URI.

GP2. Provide HTML documentation: It is important to provide human-readable documentation (LDV2) so that third parties (data publishers, ontology developers, etc.) can understand the ontology more easily, boosting, therefore, its use (e.g. describing data from) and reuse (e.g. within another ontology).

- **Example:** "Accommodation Ontology Language Reference (acco)", which URI is http://purl.org/acco/ns, provides HTML documentation by redirecting to http://ontologies.sti-innsbruck.at/acco/ns.html.

GP3. Content negotiation for RDF: According to (a) LDV4, (b) the best recipes for publishing vocabularies[8] "*It is accepted as a principle of good practice that HTTP clients SHOULD include an 'Accept:' field in a request header, explicitly specifying those content types that may be handled*." and (c) the rule "*Implement 303 redirects for real-world objects*" proposed in [1], it is a good practice to provide RDF description of the vocabulary using content negotiation mechanisms to retrieve it when the Accept header indicates this format.

- **Example: "**ACM Classification Ontology (acm)" with URI http://www.rkbexplorer.com/ontologies/acm provides correct content negotiation mechanism when asking for RDF content.

GP4. Content negotiation for HTML: According to (a) LDV4, (b) the best recipes for publishing vocabularies "*It is accepted as a principle of good practice that HTTP clients SHOULD include an 'Accept:' field in a request header, explicitly specifying those content types that may be handled*." and (c) the rule "*Implement 303 redirects for real-world objects*" proposed in [1], it is a good practice to provide HTML description of the vocabulary using content negotiation mechanisms to retrieve it when the Accept header indicates this format.

- **Example:** "Agent Relationship Ontology (agrelon)" with URI http://d-nb.info/standards/elementset/agrelon.owl# implements correct content negotiation mechanism when requesting (X)HTML.

GP5. Provide *vann* metadata: As an ontology URI does not necessarily corresponds to the namespace where the ontology elements are defined it is a good practice to indicate by means of metadata the namespace used for defining

[8] http://www.w3.org/TR/swbp-vocab-pub/

them. In this sense, we also consider a good practice to indicate a preferred prefix used when referring to the given ontology. This good practice is related to LDV2 as it is related with the metadata provided within the ontology.

- **Example:** "The Lingvoj Ontology (lingvo)" with URI http://www.lingvoj.org/ontology it a good example of providing vann metadata to indicate the preferred namespace and prefix for the ontology.

GP6. Well-established prefix: Even though it is no crucial, it would be desirable that a prefix used for a given vocabulary is well-established and there is consensus about it across applications. For example, in the case of "foaf" there is no doubt to which vocabulary is this prefix referring to.

- **Example: "**Algorithms Ontology (algo)" with URI http://securitytoolbox.appspot.com/securityAlgorithms has a consistent prefix across systems, in this case, LOV and prefix.cc[9].

2.2 Pitfalls proposed

The following five characteristics represent our proposal for pitfalls in ontologies regarding publishing issues and metadata. These five characteristics represent undesirable situations to be found in an online ontology, or in other words, a publisher team would not like to see these characteristics in its ontologies.

P36. URI contains file extension: Guidelines in [1] suggest avoiding file extension in persistent URIs, particularly those related to the technology used, as for example ".php" or ".py". In our case we have adapted it to the ontology web languages used to formalized ontologies and their serializations. In this regard, we consider as pitfall including file extensions as ".owl", ".rdf", ".ttl", ".n3" and ".rdfxml" in an ontology URI.

- **Example:** "BioPAX Level 3 ontology (biopax)" ontology's URI (http://www.biopax.org/release/biopax-level3.owl) contains the extension ".owl" related to the technology used.

P37. Ontology not available: This bad practice is about not meeting LOD1 from Linked Data star system that stars "On the web" and LDV1 that says "Publish your vocabulary on the Web at a stable URI".

- **Example: "**Ontology Security (ontosec)" which URI is http://www.semanticweb.org/ontologies/2008/11/OntologySecurity.owl is not available online as RDF nor as HTML[10].

P38. No OWL ontology declaration: The *owl:Ontology* tag aims at gathering metadata about a given ontology as version information, creation date, etc. It is also used to declare the inclusion of other ontologies. Not declaring this tag is consider as a bad practice for owl ontologies as it is a symptom of not providing useful metadata as proposed in LDV2.

- **Example:** "Creative Commons Rights Expression Language (cc)" ontology with URI http://creativecommons.org/ns does not have any

[9] http://prefix.cc/

[10] By the time of carrying out this study at 19th of June of 2013.

owl:Ontology declaration in its RDF file even though there are other OWL elements used as, for example, *owl:equivalentProperty*.

P39. Ambiguous namespace: In the case of not having defined the ontology URI nor the *xml:base* namespace, the ontology namespace is matched to the file location. This situation is not desirable as the location of a file might change while the ontology should remain stable as proposed in LDV1.

- **Example:** "Basic Access Control ontology (acl)" with URI http://www.w3.org/ns/auth/acl has no *owl:Ontology* tag nor *xml:base* definition.

P40. Namespace hijacking: This bad practice refers to the situation when an ontology is reusing or referring to terms from other namespaces that are not defined in such namespace. This is an undesirable situation as no information could be retrieve when looking up those undefined terms, in addition, there would be no meaning or semantic behind them. In addition this practice is against Linked Data publishing guidelines provided in [3] "*Only define new terms in a namespace that you control*."

- **Example:** the **"**WSMO-Lite Ontology (wl)" which URI is http://www.wsmo.org/ns/wsmo-lite#, uses http://www.w3.org/2000/01/rdf-schema#Property" that is not defined in the rdf namespace (http://www.w3.org/2000/01/rdf-schema#) instead of using http://www.w3.org/1999/02/22-rdf-syntax-ns#Property**,** that is actually defined in the rdfs namespace (http://www.w3.org/1999/02/22-rdf-syntax-ns#).

2.3 Dependencies between good practices and pitfalls

It is obvious that some good practices and pitfalls appearance is conditional upon the appearance of another one. In this sense, some characteristics block the potential appearance of others, for example, if it not possible to retrieve the RDF description of an ontology it cannot be checked whether it has *vann* metadata defined in it. These connections are shown in Figure 1 by means of the relation "X depends on Y".

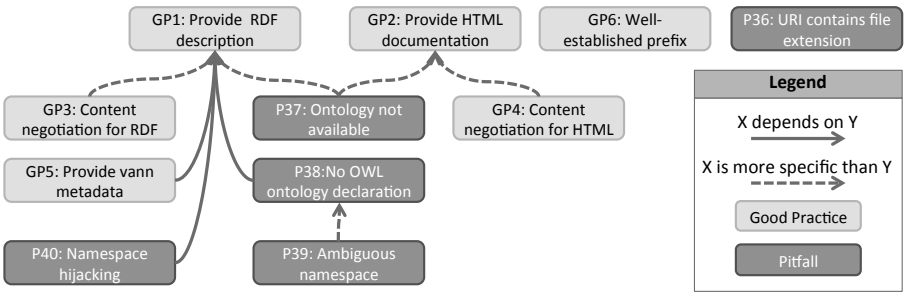


Figure 1. Dependencies between good practices and pitfalls.

Another dependency between characteristics is the case of a good practice or a pitfall being more specific than other. For example, providing HTML documentation

implementing correct content negotiation mechanisms is more specific than just serving HTML documentation. These associations are shown in Figure 1 by means of the relation "X is more specific than Y". For these cases we need the most general characteristic to be true in order to check a more specific one. The opposite is also possible, for example, for "P37. Not available" to be possible "GP1. Provide RDF description" and "GP2. Provide HTML description" have to be false. This information is important from the publisher point of view as it indicates which detections could be affected when correcting another issue.

3 Description of the methods used to identify good practices and pitfalls in ontologies

In this section, the detection methods used within this study for each good practice (Section 3.1) and pitfall (Section 3.2) are detailed. These methods have been coded and applied over the 355 vocabularies registered in LOV at the moment of carrying out this study. The results and analysis of such execution are shown in Section 4.

3.1 Good practices detection methods

Detection method for GP1. Provide RDF description: To check whether the ontology, given its URI it, can be loaded and processed by means of an RDF API, in our case we use JENA[11].

Detection method for GP2. Provide HTML documentation: To check whether, given an ontology URI, an HTML document is retrieved when requesting HTML in the accept header. This is checked by means of looking for HTML tags in the retrieved content. We do not use any HTML parser as they add the tag needed to make a valid HTML page from sources that do not really follow this syntax.

Detection method for GP3. Content negotiation for RDF: To check whether, given an ontology URI, it provides an rdf/xml serialization when asking for RDF in the accept header and it implements the redirections mechanism: 303-200. We use Vapour[12] for checking this point and adapted its behaviour for purl ontologies considering also the sequence 302-303-200.

Detection method for GP4. Content negotiation for HTML: To check whether, given an ontology URI, it provides an HTML document when asking for HTML in the accept header and it implements the redirections mechanism: 303-200. We use Vapour for checking this point and adapted its behaviour for purl ontologies considering also the sequence 302-303-200.

Detection method for GP5. Provide *vann* metadata: To check whether there is at least one result for the following SPARQL query executed over the ontology model loaded in JENA:

[11] http://jena.apache.org/

[12] http://validator.linkeddata.org/vapour

```
SELECT ?prefPrefix ?prefNS WHERE{
       OPTIONAL {?s vann:preferredNamespacePrefix ? prefPrefix.}
       OPTIONAL {?s vann:preferredNamespaceUri ?prefNS.}}
```

Detection method for GP6. Well-established prefix: To check that the prefix defined in LOV for a given ontology matches with the one defined in prefix.cc. The detection method first, checks if given the ontology namespace we obtain from prefix.cc the same prefix as declared in LOV. If no prefix is retrieved, the service is used the other way around, the namespace recorded in prefix.cc for the prefix given in LOV is requested. If the two prefixes (the one from LOV and the one obtained, if any, from prefix.cc) are equal we say that the ontology meets this characteristic, otherwise it does not.

3.2 Pitfalls detection methods

Detection method for P36. URI contains file extension: To check whether the ontology URI contains the string ".owl" or ".rdf" or ".n3" or ".ttl".

Detection method for P37. Ontology not available: To check whether neither GP1 nor GP2 hold, that is, if they both are false.

Detection method for P38. No OWL ontology declaration: To check whether there is an "owl:Ontology" tag defined in the ontology or not. It is worth mentioning that this check is done over the raw text containing the RDF code and applying the following seven regular expressions:

```
<owl:Ontology rdf:about="
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Ontology"/>
a(\\s+)owl:Ontology(\\s*);
rdf:type(\\s+)owl:Ontology(\\s*);
a(\\s+)owl:Ontology(\\s*),
rdf:type(\\s+)owl:Ontology(\\s*),
<owl:Ontology>
```

Detection method for P39. Ambiguous namespace: To check whether the RDF code of a given ontology matches at least one of the following cases:

a. There is no "owl:Ontology" tag declaration nor "xml:base" defined.
b. There is no "owl:Ontology" tag declaration and the "xml:base" is empty.
c. The "rdf:about" in the "owl:Ontology" tag declaration is empty and there is no "xml:base" defined.
d. The rdf:about in the "owl:Ontology" tag and the "xml:base" are empty.

Detection method for P40. Namespace hijacking: For detecting this pitfall we rely on Triple Checker[13]. It should be noted that we only consider as error the case of an ontology using undefined terms in a namespace even though Triple Checker also warns about other issues. For example, analysing "Appearances Ontology Specification"[14] we consider as P40 the case of the term

[13] http://graphite.ecs.soton.ac.uk/checker/

[14] A copy of the result given by Triple Checker for the "Appearances Ontology Specification" at 11th of June of 2013 is available at http://goo.gl/MD9FDo

"http://swrc.ontoware.org/ontology#date-added" however other warnings are not, for example, the one given for the term "http://rdf.muninn-project.org/ontologies/muninn#wikipedia_version".

4 Results and Analysis over LOV vocabularies

In this section, results and statistics for the LOV ecosystem status at 19th of June of 2013 are shown. Main motivations for choosing vocabularies in LOV as vocabulary registry for carrying out this study are the facts that (a) the ecosystem is updated and manually curated, (b) contains a reasonable number of vocabularies registered (more than 350) and (c) provides complete information and trustable values for the data needed (in our case we need: namespace, URI and prefix) for each vocabulary.

Figure 2 shows for each characteristic (good practice or pitfall) how many times it has been detected within the 355 analyzed vocabularies. For example, the first column shows that "GP1. Provide RDF description" appears in 308 ontologies (marked as 'Good Practice detected') and it does not appear in 47 (marked as 'Good Practice not detected'). For the pitfalls, in the seventh column we observe that "P36. URI contains file extension" appears in 39 ontologies (marked as 'Pitfall detected'), while it does not appear in 316 (marked as 'Pitfall not detected').

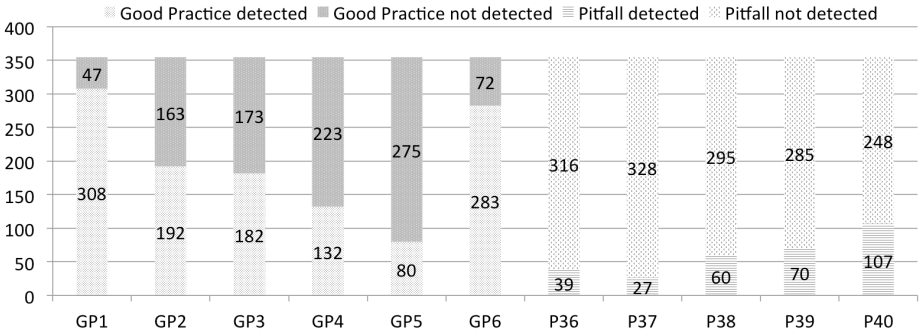
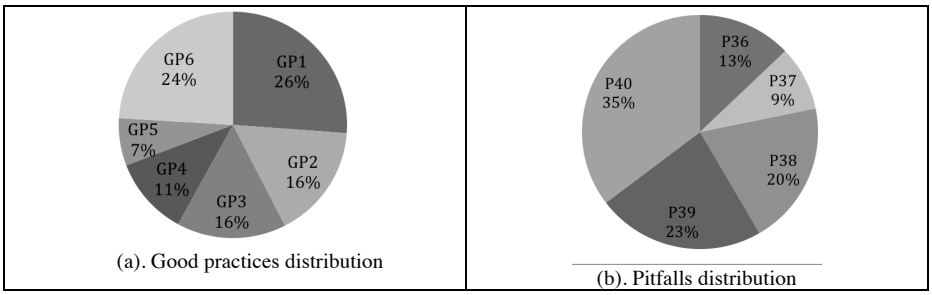


Figure 2. Good practices and pitfalls frequency.

The information shown in Figure 3 represents the distribution of good practices (a) and pitfalls (b) among the total number of appearance. That is, looking at the pie chart in the right, the slice for P40 means that among all the pitfalls appearances over the 355 ontologies (a total of 303: the sum of all the values for 'Pitfall detected' in Figure 2), 35% it has been a case of "P40. Namespace hijacking".

From Figure 2 and Figure 3 we see that most of the good practices are present in more than half of the ontologies analysed, being the most popular "GP1. Provide RDF description" and "GP6. Well-established prefix". Even though GP1 is the good practice appearing most it is still alarming that in more than 40 ontologies they could not have been processed programmatically. This is clearly a problem as it impedes the the ontology (re)usability and, in case some data is annotated with such an ontology, it semantics could not be retrieved, turning it into meaningless data. The high appearance of GP6 might be surprising as it is quite specific and requires and extra effort

from ontology managers. This high frequency is due to the efforts from LOV curators editing prefix.cc content to keep as many prefixes as possible equal in both systems. Regarding the pitfalls, we can observe that they are scarcely present apart from "P40. Namespace hijacking" that have a high frequency. Which is also alarming as defining terms in namespaces out of our control would lead to de-referenceability and lack of semantics issues, indeed it clearly goes against main guidelines for publishing LD [3].



(a). Good practices distribution

(b). Pitfalls distribution

Figure 3. Good practices and pitfalls appearance distribution.

Figure 4 shows the number of ontologies that have a given number of good practices and pitfalls. For example, the bubble in the top row and third column starting from the left means that there are 32 ontologies having 2 good practices and 0 pitfalls. In this grid we see that most of the ontologies have none, one or two pitfalls while most of the ontologies have between 2 and 5 good practices. Even though the general landscape is not bad, there is still work to do in order to achieve the ideal situation where all the vocabularies are placed in the right corner at the top, that is, having the maximum number of good practices implemented and none pitfalls. It should be noted that the information shown in Figure 4 has been condensed and that a detailed grid showing the name of the ontologies is available at http://goo.gl/zu9ZbW.

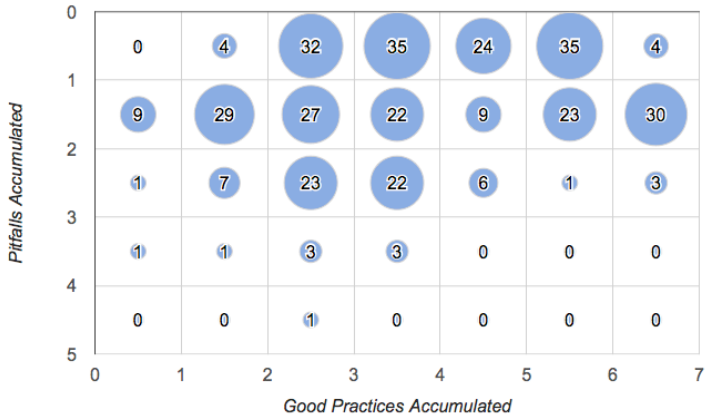


Figure 4. Number of vocabularies by good practices and pitfalls accumulated grid.

5 Related work

Ontology evaluation is a key process that should be performed at different stages of the ontology development and deployment. As important as correctly modelling the intended part of the world to be captured in an ontology, is publishing the model following good practices and avoiding bad practices.

However, apart from the aforementioned publishing recommendations (See Section 2), to the best of our knowledge, most of the evaluation approaches are focused on the ontology content quality or syntax checking and there is not too much research on approaches for validating the ontology publication process.

Regarding ontology content quality evaluation, and not directly related to LD features, it is important to mention plug-ins for desktop applications as XDTools plug-in[15] for NeOn Toolkit and OntoCheck plug-in[16] for Protégé; the wiki-based ontology editor MoKi [5] that incorporates ontology evaluation functionalities; and the online tool OOPS! [6] that detects potential pitfalls in ontologies.

In addition, validation services for RDF and LD have also been developed. One of the most popular tools is the W3C RDF Validation Service[17] that checks the syntax of RDF documents. In this regard, RDF:Alerts[18] also checks for syntax errors, undefined terms, among others. Regarding protocol issues, the online tools Vapour[19] [2] and Hyperthing[20] aim at validating the compliance of a resource according to LD publication rules. These tools check the de-referenceability of a given URI.

We can also mention evaluation works with respect to SKOS vocabularies where several tools have been proposed. Those that check characteristics related to LD, are qSKOS [4] that checks missing in and out links, broken links, undefined SKOS resources and HTTP URI scheme violation; and PoolParty[21] that also checks URI correctness.

6 Conclusions and future work

Along this paper 6 good practices and 5 pitfalls have been proposed and described. Detection methods for each of them have also been suggested and implemented[22]. With this contribution, ontology evaluation tools and quality features catalogues could be extended. In addition, an evaluation of the good practices and pitfalls detection has been carried out over 355 vocabularies registered in LOV.

A grid-based rating system has also been proposed. In this grid[23] the vocabularies are positioned according to the total number of good practices and pitfalls appearing.

---

[15] http://neon-toolkit.org/wiki/2.3.1/XDTools
[16] http://protegewiki.stanford.edu/wiki/OntoCheck
[17] http://www.w3.org/RDF/Validator
[18] http://swse.deri.org/RDFAlerts
[19] http://validator.linkeddata.org/vapour
[20] http://www.hyperthing.org/
[21] http://demo.semantic-web.at:8080/SkosServices/check
[22] Complete execution results are provided at http://goo.gl/zu9ZbW
[23] It refers to the detailed grid available at http://goo.gl/zu9ZbW instead of the one in section 4.

This grid could be used by (a) LOV curators in order to identify which vocabularies need to be reviewed and (b) vocabulary authors and publishers in order to detect possible improvements by meeting more good practices and avoiding pitfalls.

First conclusion we can draw is that vocabularies in LOV seem to be well maintained and likely to be high quality. It could be due to the fact that the LOV ecosystem is reviewed and conflictive vocabularies authors are contacted when a problem is encountered and, in the worst case, the vocabularies are deleted from the ecosystem. In this way, LOV administrators keep a high standard for the vocabularies registered. That is, it is a goodness of a semi-handcrafted registry against crawlers gathering vocabularies and ontologies over the web with little or no review and maintenance.

Second, it is worth mentioning that some practices that one would not expect to find in a stable and well-established ontology are surprisingly quite present within the analysed ontologies, e.g. making the RDF code of the ontology available online or not using terms from other namespaces that are not actually defined in such namespace.

Third, it is worth mentioning that it is difficult to define the division line between good practices and pitfalls as in some cases the absence of a good practice (e.g. "GP1. Provide RDF description") could be taken as a pitfall and the other way round. However, it does not hold for all of the good practices and pitfalls defined in this work. For example, the lack of some pitfalls (e.g. "P40. Namespace hijacking") does not really represent a good practice or a high quality point for the ontology.

Future lines of work will include to deal with the detection of (a) metadata about licences in order to check LDV1; (b) other kind of metadata apart from *vann* annotation, for example, creators, authors, dates, languages, etc. as proposed in LDV2; (c) linguistic information in order to check LDV3 and (d) reused terms within the analysed ontology in order to check LDV5. In addition different importance levels could be attached to each good practice or pitfall, as it is obvious that, for example, an ontology containing the file extension in its URI is not as critical as a case of namespace hijacking. This information would be useful to assess and rank ontologies weighting the evaluation results for the good practices and pitfalls observed.

As complement to this work, we propose, as future work, to provide guidelines to solve the problems when a good practice in not implemented or a pitfall is detected.

Finally, we propose to execute described methods over an ontology registry as LOV in regular basis in order to observe the evolution of the quality of the ecosystem as a whole and for each particular vocabularies in particular and draw trends and patterns when publishing vocabularies.

Acknowledgments. This work has been partially supported by the Spanish project *BabelData* (TIN2010-17550), the mobility and internationalization program by the *Consejo Social* of the *Universidad Politécnica de Madrid* and the French project Datalift (ANR-10-CORD-009).

References

1. Archer, P., Goedertier, S., and Loutas, N. *D7.1.3 – Study on persistent URIs, with identification of best practices and recommendations on the topic for the MSs and the EC*. Deliverable. December 17, 2012.

2. Berrueta, D., Fernández, S., and Frade, I. *Cooking HTTP content negotiation with Vapour*. ESWC2008 workshop on Scripting for the Semantic Web (SFSW2008), Tenerife, Spain. June 2, 2008.
3. Heath, T., Bizer, C.: *Linked data: Evolving the Web into a global data space* (1st edition). Morgan & Claypool (2011)
4. Mader, C., Haslhofer, B., & Isaac, A. *Finding quality issues in SKOS vocabularies*. In Theory and Practice of Digital Libraries. Springer Berlin Heidelberg. 2012.
5. Pammer, V. *PhD Thesis: Automatic Support for Ontology Evaluation Review of Entailed Statements and Assertional Effects for OWL Ontologies*. Engineering Sciences. Graz University of Technology.
6. Poveda-Villalón, M., Suárez-Figueroa, M.C., Gómez-Pérez, A. *Validating ontologies with OOPS!*. 18th International Conference on Knowledge Engineering and Knowledge Management. (EKAW2012) Galway, Ireland, 8 - 12 October 20

Event Processing in RDF

Mikko Rinne[1], Eva Blomqvist[2], Robin Keskisärkkä[2], and Esko Nuutila[1]

[1] Department of Computer Science and Engineering,
Aalto University, School of Science, Finland
firstname.lastname@aalto.fi
[2] Linköping University, 581 83 Linköping, Sweden
firstname.lastname@liu.se

Abstract. In this study we look at new requirements for event models based on concepts defined for complex event processing. A corresponding model for representing heterogeneous event objects in RDF is defined, building on pre-existing work and focusing on structural aspects, which have not been addressed before, such as composite event objects encapsulating other event objects. SPARQL querying of event objects is also considered, to demonstrate how event objects based on the model can be recognized and processed in a straightforward way with SPARQL 1.1 Query-compliant tools.

Keywords: Complex event processing, ontologies, SPARQL, RDF

1 Introduction

Event models (e.g. [17, 19]) currently available for tools using Semantic Web technologies do not address all necessary aspects of event processing, for example, composite event objects where higher-level event objects encapsulate lower-level event objects. Work on stream processing using Semantic Web technologies has initially focused on processing streams of individual triples [3, 11, 10] rather than events, but the use of larger subgraphs with more heterogeneous structures using RDF[3] and SPARQL[4] has also been described [15]. We extend current event models to incorporate more aspects of (complex) event processing and demonstrate how streams of structured and heterogeneous event objects, represented based on our model, can be processed using SPARQL 1.1.

Complex event processing, as pioneered by Luckham, Etzion and Niblett [13, 6], is based on layered abstractions of events. An event is defined by [14] as "anything that happens, or is contemplated as happening". A complex event is "an event that summarizes, represents, or denotes a set of other events". Real-world events are observed by sensors, which translate them to simple *event objects*, i.e., records of the observations in the system environment, which constitute the representation of an event that the system processes. Interconnected rule processors, denoted "event processing agents" (EPA) [14], transform patterns of

[3] http://www.w3.org/RDF/
[4] http://www.w3.org/TR/sparql11-query/

simple event objects, potentially from very heterogeneous sources, to complex event objects of higher abstraction levels. As an example, we may have sensors measuring the water level and flow in different parts of a network of rivers and lakes. That information combined with a weather forecast for heavy rain could be used to derive a flood warning, which in this case would be an abstract complex event object. So far, none of the existing event ontologies address any of the challenges of treating complex and composite event objects.

The solution has been modelled in the form of a Content Ontology Design Pattern [7] (hereafter simply denoted ODP), which is a reusable ontology component that can be used independently of the event models it is built upon, but which is also aligned to several important models. The proposed ODP is available in the ODP Portal[5]. By defining a comprehensive and structured model for representing event objects, this work addresses challenges on the level of abstraction as well as integration [5], and constitutes a novel and necessary step for further research in event processing based on Semantic Web technologies. The proposed model can be used as a tool for event processing systems to structure, integrate and manage such streams (whatever their original vocabularies), and for interchange of event objects. To use the model, it is not necessary that an incoming stream is already structured according to the ODP, refactoring of the streamed data can also be a task performed by the event processing system itself.

The paper starts by an in-depth discussion of the state of the art, existing solutions and tools, which are reviewed in Section 2. Section 3 reviews the requirements for an event model for event processing. Our solution is described in Section 4, with a discussion on benefits and shortcomings, as well as future work, in Section 5. Conclusions are presented in Section 6.

2 Background and Related Work

When something happens in the material world, it may be detected by mechanical, electric, or human sensors. These sensors emit streams of observations. A particular pattern of observations, detected by an event processing system, triggers the creation of an *event object* mirroring and/or describing the real-world event, as illustrated in Figure 1. Traditionally, observation streams have been handled by data stream management systems, with their roots in databases, where the processing unit is a row of a table [2]. Parameters, such as time, are used to portion infinite streams into windows, which are processed by aggregate operators to derive numerical conclusions descriptive of the contents. This heritage has later been applied to the Semantic Web by constructing streams out of time-annotated RDF triples [8, 9, 3] and extending SPARQL with window operators [3, 11], which isolate portions of the streams based on the timestamps.

Processing based on individual triples is, however, very limited. Most data sources, including sensors, can attach various measurements and other attributes than time (e.g., location) to the units of data they provide [12]. Moreover, an

[5] http://ontologydesignpatterns.org/wiki/Submissions:EventProcessing

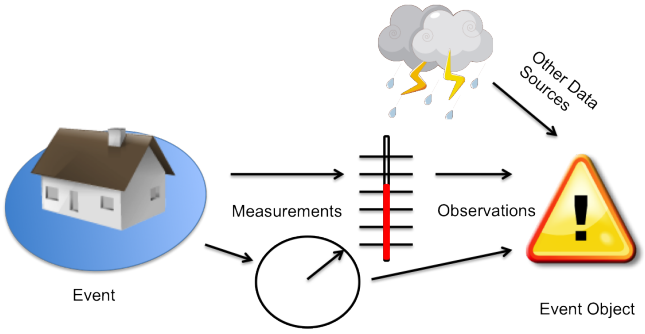


Fig. 1: An event object compiled from sensor observations and other data sources.

event object can be associated with multiple time-related parameters, e.g., time of sampling based on the clock of the sensor, time of entry to the data stream, time of arrival in the event processing system and time of event object validity [8], which need to be understood by the system to produce the desired result. Computing aggregate values such as minimum, maximum, sum and average from a single parameter is a practical way to summarize and clean noisy data, but it is not yet a means to derive layered conclusions for complex event processing.

The SPARQL query language has the capability to match and isolate subgraphs of data, with significant new functionality added in v. 1.1, e.g., property path handling. Processing heterogeneous event objects consisting of multiple RDF triples with a common timestamp has been demonstrated in [15, 16]. In this paper we review those aspects of (complex) event modelling, which in our view have not been fully addressed in the event (and semantic sensor) ontologies currently available [17, 19, 12].

The final report of the W3C Semantic Sensor Network XG [12] reviews numerous existing event and sensor ontologies, and subsequently describes a comprehensive ontology[6] (hereafter denoted the SSN ontology) for conveying the output of sensors. The concepts "Observation" and "SensorOutput" in the SSN ontology are, however, restricted to describing the output of exactly one sensor, so they do not extend to the concept of event objects nor abstractions into complex events. Our model extends the SSN ontology with such concepts, and follows a common baseline by using the DOLCE Ultra Light[7] (hereafter denoted DUL) top-level ontology as a formal basis, to be compatible with the SSN ontology.

When extending the SSN ontology with the concept of (complex) event objects, we have reviewed and considered to reuse existing event ontologies. The Event Ontology[8], rooted in describing music events, and the LODE ontology[9],

[6] http://www.w3.org/2005/Incubator/ssn/ssnx/ssn
[7] http://www.loa-cnr.it/ontologies/DUL.owl
[8] http://motools.sf.net/event
[9] Linking Open Descriptions of Events: http://linkedevents.org/ontology/

are both general enough to be applicable also for event processing, but lack some of the structures needed, for instance, the notion of complex events as abstractions over simple events. Taylor and Leidinger define an ontology [19] for complex event processing[10], but it is highly specific to the problem domain, containing references to particular observations, such as wind speed, which makes it unsuitable as a general pattern. The Event-F ontology[11] [17] on the other hand is a comprehensive framework, also derived from DUL, which is general enough to serve our purpose, but which still lacks the specifics of complex events. However, since Event-F is directly compatible with the SSN ontology, through DUL, it provides a good foundation for our extension, hence, we align our concepts also to Event-F. Another alignment between the two ontologies was made in the SPITFIRE project, producing an extended ontology[12] for describing sensor contexts as well as energy requirements, but this extension still lacks classes and properties for describing complex events. Also note that both SSN and Event-F are large ontologies (as is the SPITFIRE extension), and being based on DUL quite heavily axiomatized. In contrast to this, our model is published as an ODP, without importing either ontology, but rather simply aligning to them.

3 Requirements of an Event Model for Complex Event Processing

When developing the Event-F ontology, the WeKnowIt project collected a comprehensive set of requirements of general event models [18]. Such generic requirements include: participation of objects in events, temporal duration of events, spatial extension of objects, relationships between events (mereological, causal, and correlations), as well as documentation and interpretation of events. Additionally, a number of non-functional requirements, such as extensibility, formal precision (axiomatization), modularity, reusability, and separation of concerns. Even though the requirements are covered by Event-F, it does not cover all the needs of modelling complex events, hence, we here add the requirements that have not yet been addressed. The non-functional requirements have influenced the design of the model we are proposing, while taking some of the requirements even further, such as providing an ODP rather than a large core ontology of complex events, which takes the modularity requirement even one step beyond the design of the Event-F ontology.

Requirements not directly addressed by Event-F (nor any other current event model, or the SSN ontology):

1. *Events and event objects:* In the commonly agreed terminology of [14] there is a clear separation between *events*, as something occurring in the real world, and *event objects*, which are representations of the real-world events as described within some computer system that may be used to detect or process

[10] http://research.ict.csiro.au/conferences/ssn/EventOntology_no_imports.owl
[11] http://west.uni-koblenz.de/Research/ontologies/events
[12] http://spitfire-project.eu/ontology/ns/

the real-world events in some way. Although one could argue that an event model will never contain the real-world events themselves, i.e., whatever is modeled by an ontology will always be a representation of an event, we find it important to allow this distinction in a model for complex events because the breakdown of events into their parts and related events that a human user finds reasonable may considerably differ from the *event objects* that are actually present in the system for detecting or describing the event. Hence two parallel modelling structures should be used for this purpose. For instance, consider a music festival night as an event that occurs in the real world. Intuitively we may, for instance, describe this event as a set of concerts by different artists that are held in sequence on the same stage. However, a system representation of this event, i.e., the event objects, may very well display a completely different content and structural breakdown. For instance, we may use sound level sensors to detect that there is some activity on the stage, and make readings every minute, then the music festival night is actually represented by a "loud period" event object in the system, which consists of sub-events that are the individual sound level readings, together with some mechanism detecting that what is heard is actually music.

2. *Payload support:* Following [6], an event object is split into an event object header, which contains necessary information for processing the event object, and optional payload, which may not be fully understood or processed by the event processing network but needs to be kept associated with the event object. We incorporate optional "header" and "body" segments for event objects to demonstrate the capability of handling unknown components, e.g., unknown vocabularies used for the body of the event object.
3. *Encapsulated event objects:* The structurally most demanding type of event object in [14] is a "composite event object", which contains the event objects it is composed of, in a separable form. As the event objects constituting a composite event object may themselves be composite event objects, the recursion of composite event objects within composite event objects should be supported in any number of layers. It is worth noting that such a structural relation between *actual events* is already present in Event-F through the mereological relations borrowed from DUL. However, as we have already noted in the first requirement we need to clearly separate events from event objects, hence, when aligning event objects to the SSN ontology, i.e., modelling them as information objects in a system, we will need an additional such structure for the event objects (in addition to the one in Event-F).
4. *References to triggering events:* Complex event objects resemble composite event objects, since both are referencing other event objects, but while the parts of a composite event object are wholly dependent on the encapsulating event object (through partonomy) a complex event object can also simply be an event object that somehow is related to other event objects (referencing other event objects), e.g., by being an abstraction of a set of low-level event objects. The relation between complex event objects and their related event objects is therefore a kind of constituency (in the DUL terminology) rather than partonomy. An important use of such a relation is the ability to point

to the triggering event objects, so that it is possible to trace what triggered the abstraction, and hence the appearance of this complex event object.

5. *Multiple time-stamps:* An event object can be associated with multiple time-related parameters, e.g., time of sampling based on the clock of the sensor, time of entry to the data stream, time of arrival to the stream processing system and time of event object validity [8], which need to be understood by the system to arrive at the desired outcome. An event model needs to be able to distinguish between different kinds of timestamps.
6. *Querying ability:* An aspect that has been overlooked in, for instance, Event-F is the usage of the model for supporting queries over event objects. Although Event-F is logically sound and well-designed, it is not modelled particularly with querying in mind. Several structures in Event-F involve n-ary relations in several layers, modelled as OWL classes, which contributes to very long and complex query expressions. Although this may be an acceptable price to pay for increased reasoning capabilities, we also raise the importance of being able to easily query the represented event objects, and being able to formulate generic "query templates" for managing event objects.

In addition to these specific requirements, we have also tried to adhere to the general characteristics of Content ODPs, as described in [7], which can be seen as a list of desired features, including the provision of a reusable computational component representing the ODP, making the ODP small and autonomous, enabling inferencing on the ODP, and making it cognitively and linguistically relevant as well as a representation of best practices (including to adhere to a commonly agreed terminology, such as [14]).

4 Proposed Solution

4.1 Event Model in OWL

As a starting point for our proposed ODP model, we have taken the SSN ontology [12] and the Event-F ontology [18]. Event-F can be used together with our proposed ODP to connect to more detailed or user-friendly descriptions of the event itself, e.g., for reasoning purposes, when the rich axiomatization of events from Event-F is desirable, or for describing the event in a more user-oriented fashion. Both the SSN ontology and Event-F are based on DUL, and by extending and aligning to both these models, our ODP also relies on the DUL ontology. From our point of view, and in accordance with the terminology of [14], the *event object* (Req. 1) is a central concept (see Figure 2), which is the system representation, or record, of an event (real or system generated). An event object can then be related to a "real" event, i.e., a `dul:Event`, which through the alignment to the Event-F model, then has to be a documented event, i.e., a `dul:Event` involved in some `eventf:EventDocumentationSituation`.

An event object can then be either a *simple event object* or a *complex event object*, depending on if it abstracts (summarizes or represents) other, more low-level, event objects or not. A complex event object is something that has some

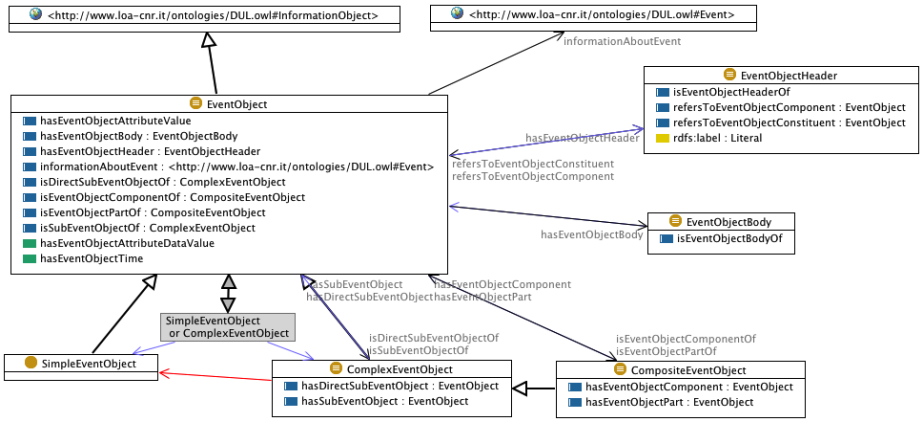


Fig. 2: The core classes of the Event Processing ODP and their relations to DUL concepts (using the UML-like notation of TopBraid Composer, where some details have been omitted due to readability reasons).

"sub-event objects" (Req. 4). A special case of a complex event object is a *composite event object* (Req. 3), which is a complex event object that is actually made up of a set of other event objects, i.e., acting as its parts. As noted by [14] a composite event object is always a complex event object, but every complex event object is not necessarily a composite event object, if it only represents or references other related event objects but does not include them as components. Encapsulated and referenced event objects can be modelled using two separate sets of properties, so that each type of relation can be treated independently. The structure does not require OWL reasoning per se, but gives the opportunity to reason over the structures, using transitivity and inverse properties.

Payload support (Req. 2) is provided through introducing classes for the header and body of an event object, making it possible to distinguish between the known parts of the information and the body, i.e., the payload that may not use any known vocabulary. Nevertheless, we feel that this legacy from earlier event processing systems may not be ideal to include in all RDF stream processing systems, whereby we have modeled the pattern in such a way that it is an optional feature. Event objects can be modelled directly, without header and body parts distinguished, which is more in line with the modelling "freedom" and simplicity of Linked Data and RDF graph data in general.

Multiple timestamps (Req. 5) are supported through a set of separate datatype properties: `hasEventObjectSamplingTime`, `hasEventObjectApplicationTime`, `hasEventObjectSystemTime`, and `hasEventObjectExpirationTime`, corresponding to the time points when the event object was sampled (e.g., recorded by a sensor), entered the data stream, arrived in the event processing system via the stream, and any known end time for the event objects validity, respectively. Although this does not solve the problem of different time references in general, at

least one can now explicitly say what time is actually recorded, and if needed, record several timestamps for each object. An additional desirable feature would be the ability to express which of the timestamps should be the default for time window operations, however, we feel that this lies outside the scope of our current pattern. Rather such capabilities should lie in a vocabulary for describing RDF streams, or event processing systems, not event objects themselves.

Finally, event objects can be effectively processed through SPARQL queries (Req. 6). In particular, we have made sure that some of the most common queries can be expressed in a generic manner, i.e., as "query patterns" (discussed in the following section), to facilitate reuse and to make event processing as uniform as possible between systems. More details on the modelling decisions and the detailed structure of the ODP can be found in the annotations of the ODP model itself, and in the pattern abstract that accompanies it [4].

4.2 Processing Events with SPARQL

Having an ODP for describing the event objects handled by an event processing system is a (practical) contribution in itself, since such a model has not existed before. However, for this contribution to be significant in the future it needs to be practically usable and beneficial to a large class of systems. As also stated in our list of requirements an important aspect of the work is to be able to effectively query the event data structured according to the model, which will make it useful in a system setting. Ideally, the ODP presented above would come with its set of generic query patterns that represent common operations on event objects, which can be reused within any domain. In this section, we show a step towards such generic query patterns, although we also point at some limitations with the current model and SPARQL standard that restrain us from providing a completely generalized solution.

The following example set of four event objects is used to demonstrate operations on composite complex event objects containing a header and a body and having a capability to reference other event objects without encapsulating them:

```
:floodWarning0001 a ep:EventObject ;
  ep:hasEventObjectHeader [
    rdfs:label "Flood Warning composite event" ;
    ep:hasEventObjectTime "2013-07-03T08:18:21"^^xsd:dateTime ;
    ep:refersToEventObjectConstituent :weather0001 ;
    ep:refersToEventObjectComponent :waterAlert0001 ;
    floodex:forecast floodex:ImminentDanger ;
  ] ; #end of Header
  ep:hasEventObjectBody [
    rdfs:comment "Exemplifies a composite event." ;
  ] . #end of Body

:waterAlert0001 a ep:EventObject ;
  ep:hasEventObjectHeader [
     rdfs:label "Water-related alert composite" ;
     ep:hasEventObjectTime "2013-07-03T08:17:21"^^xsd:dateTime ;
     ep:refersToEventObjectComponent:waterLevel2341 ;
     floodex:waterLevelChangeRate floodex:high ;
  ] ; #end of Header
  ep:hasEventObjectBody [
    rdfs:comment "Information external to our system." ;
```

```
    foaf:mbox <mailto:contactrelevanttoanothersystem@example.org> ;
  ] .

:waterLevel2341 a ep:EventObject ;
  ep:hasEventObjectHeader [
     ssn:isProducedBy  [
       a ssn:SensingDevice ;
       rdfs:label "Water Level Measurement" ;
     ] ;
     ep:hasEventObjectTime "2013-07-03T08:17:15"^^xsd:dateTime ;
     ssn:hasValue [
       dul:hasRegionDataValue 22 ;
     ] ;
  ] .

:weather0001 a ep:EventObject ;
  ep:hasEventObjectHeader [
     rdfs:label "Weather forecast for London" ;
     ep:hasEventObjectTime "2013-07-03T08:17:21"^^xsd:dateTime ;
     ep:hasEventObjectAttributeValue floodex:rain ;
  ] ; #end of Header
  ep:hasEventObjectBody [
     ...
  ] . #end of Body
```

The full example is available with the pattern and prefixes in the ODP portal. The `floodWarning` is a composite event object, encapsulating a `waterAlert`. The `waterAlert` is also composite, encapsulating a `waterLevel` measurement. The `floodWarning` also refers to a `weather` event object, but the `weather` event object is not encapsulated in the `floodWarning`. The challenge for querying is to match the `floodWarning` composite event object (for any move, copy, delete or other operation on the composite event object), with all levels of encapsulated event objects, excluding referenced information not integral to the event object.

After the addition of property paths in version 1.1, SPARQL has some new methods for supporting nested structures. Using the property path expression `(ep:hasEventObjectHeader / ep:refersToEventObjectComponent)*`, an arbitrary number of nested composite event objects can be supported. The referenced event object `weather0001` is not matched, as it is referred with `ep:refersToEventObjectConstituent`.

Matching more levels of depth for the header and body in a generic way is not as straightforward. Matching an arbitrary chain of unknown links would be a very powerful tool, which in the world of linked data could eventually end up matching that entire universe - not just the header and body triples we want. Using a property path of unspecified length with a known combination of predicates (`ep:hasEventObjectHeader` and `ep:refersToEventObjectComponent`) can be asserted safe in our controlled setting of an event object stream, where the pair of predicates can be guaranteed only to refer to direct sub-event objects in the same graph. Supporting an arbitrary number of levels of unknown predicates is, however, much more challenging and potentially dangerous. SPARQL doesn't currently offer means to follow such a property path. The same functionality would theoretically be available through negation, using a property path expression such as `(! :foobar)*`, where ":foobar" is a fabricated predicate, which should not appear in the data stream. In addition to the dangers explained above, tool support for this approach is uncertain.

In case of the header the support for more depth can always be achieved by making the SPARQL query more explicit, because the structure of the header is assumed to be known by our event processing application, but the structure of the body is assumed to be unknown. One way to support deeper structures is to restrict such structures to be linked only through blank nodes, since blank nodes cannot point to nodes outside the current graph. Linking through blank nodes allows more depth in the event object structure without setting an explicit requirement to know the contents.

Using these tools we can write a query, which correctly constructs a copy of the `floodWarning0001` composite event object including the encapsulated event objects `waterAlert0001` and `waterLevel2341` without prior knowledge of their existence or contents:

```
CONSTRUCT { # Create a copy of the matched event object, with encapsulated event objects
   ?event a ep:EventObject ;
          ep:hasEventObjectHeader ?header .
   ?header ?hp ?hv .                          # First level headers
   ?header2 ?hp2 ?hv2 .                       # Second level nested headers
   ?event ep:hasEventObjectBody ?body .
   ?body ?bp ?bv .                            # First level body
} WHERE { # Match an event object with all nested levels of encapsulated event objects
   :floodWarning0001 ( ep:hasEventObjectHeader /
      ep:refersToEventObjectComponent )* ?event .
   ?event a ep:EventObject .
   ?event ep:hasEventObjectHeader ?header .   # Mandatory header
   OPTIONAL { ?header ?hp ?hv                 # Optional first-level headers
     #Optional second-level nested headers, only through blank nodes
     OPTIONAL { BIND ( IF (isBlank(?hv), ?hv, 0) as ?header2)
                ?header2 ?hp2 ?hv2 } }
   OPTIONAL {
     ?event ep:hasEventObjectBody ?body .     # Optional body
     OPTIONAL { ?body ?bp ?bv } }             # Optional first-level body content
}
```

To keep the query compact the amount of nested levels in the header and the body has been set to match our example. More nested levels can be added by adding more nested OPTIONAL-clauses, checking that linking is taking place specifically through blank nodes. To the best of our knowledge matching an unspecified number of nested levels through blank nodes only is not possible with SPARQL 1.1 without explicit knowledge of the predicates, in which case the query would again need to be explicitly defined for each nesting level. Apart from the explicit subject (`:floodWarning0001`) the example query is generic and would work with a structurally compliant event object independent of the content. In a real-world application the explicit subject could be replaced by some other criteria to match the desired event object in an event object stream.

5 Discussion and Future Work

Our proposed model has been published as an ODP, which comes with several advantages. For instance, both the SSN and Event-F ontologies may be perceived as quite large and "heavy" to understand and use, while our small model only contains a handful of classes and properties that can be grasped quite easily. It can also be used independently of the DUL axiomatization, if this is not desirable

or compatible for a specific use case. On the other hand, the lack of upper level axiomatization can be easily amended through our careful alignments (included in the model as axioms), by simply adding the missing imports (SSN and Event-F), if the upper level is needed for a particular use case.

A generic way of matching event objects using SPARQL 1.1 was demonstrated, supporting:

- the distinction between events and *event objects* (Req. 1), if desired,
- inclusion of a header and an optional body (Req. 2), with unknown content, in the event object structure,
- composite event objects encapsulating other event objects (Req. 3), potentially over any number of nested layers (limited by the SPARQL implementation),
- referencing of other event objects without encapsulating them (Req. 4), and
- generation of query templates to process compliant event objects (Req. 6), observing related restrictions.

Support for multiple timestamps (Req. 5) is built into the model, but selection of the timestamp to use for a particular purpose was considered to be a stream-specific parameter and outside the scope of this paper.

As a restriction to the generality of queries, knowledge of the maximum number of nested levels of RDF triples supported within the header or body was observed to be required a priori, with every level adding some complexity to the query needed to match an event object. To avoid following links outside event objects, nesting of the body should only be done using blank nodes. Deviations of the format, such as allowing event objects both with and without explicit header, or objects of other classes (such as `ssn:SensorOutput`) add complexity to queries. The recommended approach would be to convert all event objects to a uniform format upon entry to the event processing network. The specific conversions are outside the scope of this document, but due to the flexibility of RDF most formats used for describing events can be converted to RDF representations compliant with the presented model in a straightforward way.

On a more general note, currently available commercial complex event processing tools[13] use different means of defining the event processing network and the agents within, apart from systems based on a common root. The introduction of an event processing model meeting the requirements of complex event processing enables tools based on Semantic Web technologies to address the same application space. Compared to proprietary approaches, RDF and SPARQL have the benefit of a specified definition language, paving the way for improved tool compatibility. Integrated processing of event streams with static, linked and open datasets in the cloud and the built-in reasoning capabilities of Semantic Web tools are also strong benefits.

A longer-term target is to make semantic stream processing systems configurable to understand and process heterogeneous, layered event objects both on

[13] e.g. http://www.thetibcoblog.com/wp-content/uploads/2011/12/cep-market-dec2011.png

live streams as well as recorded data. For recorded data, the systems will need to follow the same stream parameters (e.g. time) to be used for both algebra and relational operators. Descriptions of the operational semantics of flow processing systems should be developed so that it is possible to know a priori, where results of processing the same data using the same set of queries will be different.

To pave the way, harmonization of tools and specifications could be improved, e.g., on the following aspects:

- Common event processing model (vocabulary). This paper makes a contribution, but to be effective on a broad scale, further community consensus on the model is needed.
- Common stream description vocabulary, and publication mechanisms. In addition to describing the event objects inside the stream, and in the event processing system, streams themselves need to be described and published, e.g., similar to other web services, in an agreed upon format.
- Representations and handling of time. Understanding of a common time reference between systems using a recorded stream should be possible with a reasonable amount of configuration rather than requiring reprogramming of system components.
- Harmonized descriptions of the operational semantics of semantic flow processing systems [1]
- Benchmarking of semantic flow processing systems. There should be tests both for data stream management as well as layered event processing. Correct results, in light of operational semantics, should be defined so that performance and correctness of operation can be compared.

6 Conclusions

In this paper we propose a novel Event Processing ODP, i.e., a vocabulary for representing and reasoning over complex and composite event objects, which is needed for further progress in the area of RDF stream processing. In addition to the model itself, another contribution is the demonstration of generic query patterns for event object management using SPARQL 1.1, which facilitates event processing. The model is aligned to important standards, such as the SSN ontology, and compatible with other event models, such as Event-F, and it also meets all the requirements for representing and processing event objects that were discussed in Section 3.

Acknowledgments

This work was supported by European Commission through the SSRA (Smart Space Research and Applications) activity of EIT ICT Labs[14], and CENIIT at Linköping University through the grant 12.10.

[14] http://eit.ictlabs.eu/ict-labs/thematic-action-lines/smart-spaces/

References

1. Aglio, D.D., Balduini, M., Valle, E.D.: On the need to include functional testing in RDF stream engine benchmarks. In: 1st International Workshop On Benchmarking RDF Systems (BeRSys 2013) Co-loc. with ESWC 2013. Montpellier, FR (2013)
2. Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and issues in data stream systems. In: Proc. of the 21st ACM SIGMOD-SIGACT-SIGART symp. on Principles of database systems - PODS '02. ACM Press, New York, USA (2002)
3. Barbieri, D.F., Braga, D., Ceri, S., Grossniklaus, M.: An execution environment for C-SPARQL queries. In: Proceedings of the 13th International Conference on Extending Database Technology. p. 441. Lausanne, Switzerland (2010)
4. Blomqvist, E., Rinne, M.: The Event Processing ODP. In: Proceedings of WOP2013 - Pattern track. CEUR Workshop Proceedings, CEUR-WS.org (2013)
5. Corcho, O., García-Castro, R.: Five challenges for the semantic sensor web. Semantic Web 1(1), 121–125 (2010)
6. Etzion, O., Niblett, P., Luckham, D.: Event Processing in Action. Manning Publications (Jul 2010)
7. Gangemi, A., Presutti, V.: Ontology Design Patterns. In: Handbook on Ontologies, 2nd Ed. International Handbooks on Information Systems, Springer (2009)
8. Gutierrez, C., Hurtado, C., Vaisman, R.: Temporal RDF. In European Conference on The Semantic Web (ECSW 2005) pp. 93—107 (2005)
9. Gutierrez, C., Hurtado, C.A., Vaisman, A.: Introducing time into RDF. IEEE Transactions on Knowledge and Data Engineering 19(2), 207–218 (Feb 2007)
10. Komazec, S., Cerri, D., Fensel, D.: Sparkwave : Continuous Schema-Enhanced Pattern Matching over RDF Data Streams. In: Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems. pp. 58–68. ACM (2012)
11. Le-Phuoc, D., Dao-Tran, M., Parreira, J.X., Hauswirth, M.: A native and adaptive approach for unified processing of linked streams and linked data. In: ISWC'11. pp. 370–388. Springer-Verlag Berlin (Oct 2011)
12. Lefort, L., Henson, C., Taylor, K.: Semantic Sensor Network XG Final Report (2011), http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/
13. Luckham, D.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley, 1 edn. (2002)
14. Luckham, D., Schulte, R.: Event Processing Glossary Version 2.0 (2011), http://www.complexevents.com/2011/08/23/event-processing-glossary-version-2-0/
15. Rinne, M., Abdullah, H., Törmä, S., Nuutila, E.: Processing Heterogeneous RDF Events with Standing SPARQL Update Rules. In: Meersman, R., Dillon, T. (eds.) OTM 2012 Conferences, Part II. pp. 793–802. Springer-Verlag (2012)
16. Rinne, M., Törmä, S., Nuutila, E.: SPARQL-Based Applications for RDF-Encoded Sensor Data. In: 5th International Workshop on Semantic Sensor Networks (2012)
17. Scherp, A., Franz, T., Saathoff, C., Staab, S.: F – A Model of Events based on the Foundational Ontology DOLCE + DnS Ultralite. In: International Conference on Knowledge Capturing (K-CAP). Redondo Beach, CA, USA (2009)
18. Scherp, A., Papadopoulos, S., Kritikos, A., Schwagereit, F., Saathoff, C., Franz, T., Schmeiss, D., Staab, S., Schenk, S., Bonifacio, M.: D5.2.1 Prototypical Knowledge Management Methodology (2009), http://www.weknowit.eu/sites/default/files/D5.2.1.pdf
19. Taylor, K., Leidinger, L.: Ontology-Driven Complex Event Processing in Heterogeneous Sensor Networks. In: Proc. of the 8th Extended Semantic Web Conference (ESWC2011). pp. 285–299. Springer Berlin Heidelberg (2011)

Terminology-Based Patterns for Natural Language Definitions in Ontologies

Dagmar Gromann

Vienna University of Economics and Business, Austria
dgromann@wu.ac.at

Abstract. Natural language content in ontologies is crucial to any human interaction with them, but scarcely available. Terminology science centers on best practices in domain-specific natural languages. Hence, ontologies can benefit from the systematic approach of terminology to natural language definitions. This paper proposes an Annotation Ontology Design Pattern named "Natural Language Definition ODP" that provides natural language definitions for ontology classes. For this purpose, a (semi-)automated method for implementing this pattern combining ontology verbalization and information extraction is investigated herein and exemplified in the domain of finance.

Keywords: Annotation ODPs, Natural Language Definition, Terminology, Automatic Extraction of ODPs, Domain-Specific ODP Application

1 Introduction

A growing number of application scenarios for Semantic Web (SW) ontologies render reusable, high-quality solutions to their design increasingly important. For this purpose, Ontology Design Patterns (ODP) define a formal methodology for various aspects of ontological design, ranging from Logical to Presentation ODPs [1]. The latter seek to increase the usability and readability of ontologies from a user's perspective, which are vital to multi-lingual scenarios [2] and interactions with domain experts and users [3], and are divided into Annotation and Naming ODPs. Annotation ODPs provide best practices for homogeneous natural language (NL) expressions (`rdfs:label`) and definitions (`rdfs:comment`), while the latter focus on naming conventions [1]. A general paucity of an operational approach to NL definition authoring and its time-intensive nature led to scarce and frequently inconsistent NL definitions in ontologies. Thus, this paper investigates their (semi-)automated generation by means of the proposed Annotation ODP "Natural Language Definition ODP" based on established methods from terminology science.

Ever since its advent, terminology science has realized the need of providing a systematic approach to NL definitions. Concept-centered terminologies as defined by ISO 704:2009 and 1087:2000 consist of sets of terminology concepts in specialized domains. NL definitions are required to form these terminology

concepts and their interrelations. In contrast, ontology concepts are formally defined by means of logics. Combining the formal ontological definition with the terminological NL definition authoring method results in a multidimensional approach formalized as the proposed Annotation ODP. Thereby, the ISO 704 method of combining the denomination of the superordinate concept with (a) characteristic(s), delimiting the concept to be defined from its related concepts, can be (semi-) automated and is the foundation of the proposed pattern. Given domain-specific, axiomatized ontology elements with a minimum of NL coverage in labels or fragment identifiers, the superordinate concept's denomination can be identified and applied by using its subsumption hierarchy. For the non-trivial purpose of obtaining characteristics, three mutually complementing approaches are proposed: ontology verbalization, utilizing existing NL content, and information extraction. An example is provided by applying the pattern to the partially available NL content of Fadyart's Finance Ontology[1].

2 Natural Language Definition ODP

The objective of the NL Definition ODP is to define an ontology concept in natural language(s). Ontologies represent knowledge by formalizing vocabularies of terms as well as their interrelations and define their meaning formally. Terminologies mostly rely on NL characteristics to establish NL definitions for concepts and interrelate concepts designated by terms, appellations, or symbols. The two most important types of definitions as specified by ISO 704 are extensional definitions, listing the instances of a concept, and intensional definitions. The latter constitutes a combination of superordinate concept and manually identified delimiting characteristic(s) for concepts related generically.

The intensional approach offers the most explicit, consistent, and precise method to definition formation. It is intended to provide the minimum of information needed for human users to differentiate one terminology concept from another. To facilitate its automation, the basic textual description of ISO has been adopted and formalized for proposing the NL Definition ODP introduced in Definition 1 and illustrated in Example 1.

The pattern defines the NL definition of an `entry term`, which corresponds to the label of the ontology class. The singular form of the term is preferred, unless only available in plural, e.g. "liabilities". It utilizes the label or fragment identifier of the superordinate concept, which for the experiment herein is restricted to Noun Phrases (NP). Thereby, it obtains a context and implicitly inherits the characteristics of the superclass. The NP is connected to characteristics by utilizing a finite set of relative pronouns, verbs, and where applicable verbalized object properties. The same elements and a coordinating conjunction are needed to string together several characteristics.

Obtaining the characteristic(s) relies on a three-tiered mutually complementing approach of ontology verbalization, utilizing existing NL content, and infor-

[1] `http://fadyart.com/` version 3.04

mation extraction from structured Web resources. All three of them help specifying the relative pronoun and linking verb to be used for the concept to be defined.

Definition 1: NL Definition ODP

Entry Term [A/An] NP<superclass> [which/that/who/whose] [(can) be/include/belong to/classify as OR <objectProperty>] [(<characteristic(s)>)* and] <characteristic>

Example 1: NL Definition of Concept "Card" (Fadyart Finance Ontology)

Card [A] payment instrument [that] [has as card type] <a credit card or debit card> and [has as card data] <a start date, sequence number, holder name, expiry date, issuer name, card number, security code>

Ontology verbalization refers to the translation of ontology concepts, relations, and axioms to (controlled) natural languages, such as Attempto Controlled English [4]. In contrast to controlled natural language, the objective herein is to use verbalized ontology elements to identify the appropriate verb and relative pronoun linking the definition's characteristics. For this purpose, verbalization patterns have been identified, of which selected ones are provided below.

P1 - `ObjectUnionOf`: [a/an OR ObjectMinCardinalty] [(NP<class>,)* or] NP(class)
P2 - `ObjectMinCardinalty`: at least <number>
P3 - `ObjectSomeValuesFrom`: NP<class>(domain)[<ObjectSomeValuesFrom> that <ObjectProperty> [a/an] NP<class>(range(s))]
P4 - `ObjectProperty` with "has" is split into two parts: NP<class><(domain) <ObjectProperty:has> as <ObjectProperty:rest> [a/an] NP<class>(range)

Should the label of the object property already contain the concept label in the range, the concept label is not reiterated in the NL definition, e.g. "hasManager" pointing to "Manager". The above list is not exhaustive and requires NLP methods for its implementation, e.g. tokenization. The application of these patterns to characteristic formation will be exemplified in the next section.

In a next step, the existing `rdfs:comment` of the ontology class is linked to the NP and, where applicable, verbalized content by means of a coordinating conjunction and the identified relative pronoun, which, if not available in the comment, can be obtained from Wiktionary.

If no NL content is available, re-using existing structured Web resources, such as DBpedia, has been considered. The tentative information extraction process herein relies on string matching and an immediate subsumption to top DBpedia ontology concepts (e.g. Organization, Resource). Reducing NL definitions to DBpedia information might result in quality issues. For instance, circular definitions are frequent on DBpedia, i.e., a term is defined by itself or by a second term that refers back to the first term. For instance, "Debtor" is defined as "Debtor owes a debt to someone ...". Applying the proposed pattern ensures the proper context for the concept, i.e., superordinate concept, and DBpedia information provide useful additional details.

Due to its systematic nature, the described pattern enables a consistent formation of NL definitions, which strongly enhances the human readability of ontologies it is applied to. The proposed pattern is illustrated for the English language and requires minor adaptations for its realization in other NLs syntactically similar to English provided lexical resources are available.

3 Example Application

An OWL ontology serves as the input to the intended system design, here exemplified with Fadyart's Finance Ontology in English. By means of the OWL API the ontology can be parsed, the subclass relations and object properties identified, and an annotation property can be added. Starting from $\top$, the subsumption hierarchy is traversed to the first concept not directly subsumed by it. If its superordinate concept contains no label, its fragment identifier is tokenized (using e.g. the Stanford Core NLP) and represents the `NP<superclass>` of Definition 1. In Example 2, the class ClientPortfolio is the subclass of AccountsPayable.

To ensure the correct grammatical number and relative pronoun, the superclass term is queried in Wiktionary, e.g. Java-based Wiktionary Library[2]. Here, the query returns "plural only" and the relative pronoun "that"[3] for "Accounts Payable". Subsequently, tokenization and verbalization pattern P4 defined in the previous section are applied to the object property of Example 2. Its range consists of a union of three classes, which is verbalized using pattern P1. Finally, the existing comment is to be added to the already obtained definition. By means of Wiktionary "Clients" in the existing `rdfs:comment` is identified as countable noun, so its singular form can be combined with the obtained definition using a coordinating conjunction and the relative pronoun identified above. The derived definition can be added to the concept ClientPortfolio as `rdfs:comment`.

Example 2: Class ClientPortfolio in Manchester Syntax

Original Input in OWL

```
Ontology: <http://www.fadyart.com/Finance.owl> ...
ObjectProperty: hasClientPortfolioBeneficialOwnerOfIncome
  SubPropertyOf: hasAccountDomain
  Domain: ClientPortfolio
  Range: PartyHolder or PartyLegalRepresentative or
       PartyUsufructuary
Class: AccountsPayable
  Annotations: rdfs:label "Accounts payable"@en
  SubClassOf: ShortTermLiabilities
Class: ClientPortfolio
  Annotations: rdfs:label "Client accounts"@en,
  rdfs:comment "The clients of the financial institution for who's
       account the securities handling operations are performed."^^
       xsd:string
  SubClassOf: AccountsPayable
```

Resulting Definition

Client Account
Accounts payable that has as client portfolio beneficial owner of income at least one party holder, party legal representative, or party usufructuary and that is a client of the financial institution for who's account the securities handling operations are performed.

Several object properties and enormous unions render it necessary that domain experts decide which verbalization most adequately defines the concept. Additionally, at times comments are utilized for supplementary information rather than NL definitions of concepts, which is why for some cases the comments might not be re-used for the definition formation process.

4 Related Work

Glosses for ontology concepts reuse existing lexical resources, e.g. WordNet, to provide ontology engineers with various linguistic descriptions to choose from for a specific concept [5]. Approaches grounding existing ontologies in lexical and

[2] `http://code.google.com/p/jwktl/`
[3] "Money that is owed ..."

linguistic descriptions (e.g. *lemon*[4]) either re-use glosses for NL descriptions or derive meaning by pointing to the semantic object in the ontology. Ontology verbalization utilizes formalized knowledge in ontologies to derive NL descriptions. For instance, the SWAT project[5] facilitates the understandability of verbalized entailments by providing individual inference steps in the English language [6]. Instead of providing an external meta-model or ontology engineering support, the proposed pattern seeks to re-use existing resources and use verbalization patterns in order to provide NL definitions for existing domain-specific ontology classes. As a standard-based approach, it reflects established best practices and accepted semiotic theories.

5 Discussion and Future Work

This paper proposes a NL definition ODP on the basis of definition formation methods from terminology science. Subsequent to defining the pattern, a (semi-)automated design to obtaining NL definitions by means of ontology verbalization, utilization of existing NL comments, and information extraction has been exemplified in the financial domain. In terms of future work, the degree to which the pattern can be generalized to other domains will be tested. As regards, information extraction, a profound disambiguation process will be considered. Furthermore, its formalization for a submission to the ontology design pattern repository is planned.

References

1. Presutti, V., Blomqvist, E., Daga, E., Gangemi, A.: Pattern-Based Ontology Design. In Suárez-Figueroa, M.C., Gómez-Pérez, A., Motta, E., Gangemi, A., eds.: Ontology Engineering in a Networked World. Volume 12. Springer (2012) 35–64
2. Cimiano, P., Buitelaar, P., McCrae, J., Sintek, M.: LexInfo: A Declarative Model for the Lexicon-Ontology Interface. Web Semantics: Science, Services and Agents on the World Wide Web **9**(1) (2011) 29–51
3. Damljanovic, D., Agatonovic, M., Cunningham, H.: Natural Language Interfaces to Ontologies: Combining Syntactic Analysis and Ontology-Based Lookup through the User Interaction. In Sure, Y., Domingue, J., eds.: The Semantic Web: Research and Applications. Volume 19. Springer (2010) 106–120
4. Kaljurand, K., Kuhn, T.: A Multilingual Semantic Wiki Based on Attempto Controlled English and Grammatical Framework. In Corcho, P.C.O., Hollink, V.P.L., Rudolph, S., eds.: The Semantic Web: Semantics and Big Data. Volume 17. Springer (2013) 427–441
5. Jarrar, M.: Position paper: Towards the Notion of Gloss, and the Adoption of Linguistic Resources in Formal Ontology Engineering. In: Proceedings of the 15th international conference on World Wide Web, ACM (2006) 497–503
6. Nguyen, T.A.T., Power, R., Piwek, P., Williams, S.: Predicting the Understandability of OWL Inferences. In Corcho, P.C.O., Hollink, V.P.L., Rudolph, S., eds.: The Semantic Web: Semantics and Big Data. Volume 17. Springer (2013) 109–123

[4] `http://lemon-model.net`
[5] `http://swatproject.org`

Towards Diagrammatic Ontology Patterns

Gem Stapleton[1], John Howse[1], Kerry Taylor[2], Aidan Delaney[1], Jim Burton[1], and Peter Chapman[1]

[1] University of Brighton, UK, www.ontologyengineering.org
{g.e.stapleton,john.howse,a.j.delaney,j.burton,p.chapman}@brighton.ac.uk
[2] CSIRO Computational Informatics and Australian National University, Australia
kerry.taylor@csiro.au

Abstract. It has long been recognized that patterns can be a useful and important tool when building models. This is reflected by their adoption in the practice of ontology and software engineering. Similarly, visual representations of information are often seen as beneficial with, for example, software engineering making use of the suite of diagrammatic notations forming the UML. Likewise, ontology engineering has seen the development of a variety of different visualizations for classes and properties. This paper combines these two strands of work, making visual (diagrammatic) patterns available to ontology engineers.

1 Introduction

This paper ties together two strands of research: ontology engineering using patterns and ontology visualization. Major benefits of using patterns include the simplification of the modelling process and the provision of a consistent approach to specifying commonly occurring constructions. Patterns give ontology engineers convenient access to these constructions. Similarly to the design patterns of object-oriented software development [3], they provide a common language for analyzing and sharing reusable, composable design abstractions. The visual ontology engineering patterns we present in this paper are lower-level than a typical design pattern from software development but fulfil the same role and may serve as building blocks for more complex patterns. The aim of visualization is similar to that of patterns, in the sense that visualizations are intended to aid ontology engineering. Visualizations (such as OWLViz [5], OntoGraf [1] and CMap [4]) can bring about benefits by revealing information that could be unapparent when using traditional notations. The main contribution of this paper is a set of diagrammatic patterns for ontology engineering, defined using *concept diagrams* [7]. Section 2 an introduction to concept diagrams, focusing on the aspects needed for this paper. Section 3 defines patterns for commonly occurring constraints and section 4 applies the patterns.

2 Concept Diagrams for Ontology Modelling

This section provides an introduction to concept diagrams, designed as part of the Ontology Engineering with Diagrams project (`www.ontologyengineering.org`).

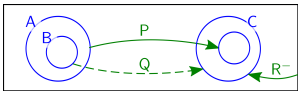


Fig. 1. A concept diagram

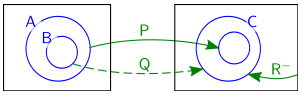


Fig. 2. Multiple boundary rectangles

Readers interested in the full notation and its formalization should see [7]. Concept diagrams represent classes using *closed curves* and properties using *arrows* which can be *solid* or *dashed*. The spatial relationships between the closed curves and the sources and targets of arrows convey semantic information.

Fig. 1 asserts that the class B is subsumed by A and both A and B are disjoint from C. The spatial properties of inclusion and exclusion correspond to the semantic properties of set inclusion and disjointness. The *solid* arrow P asserts that individuals in A are only related to elements in C: the target of the arrow is the set of things to which the elements in A are related and this set is subsumed by C. The solid arrow R^-, where R^- is the *inverse* of R, asserts that all things are, between them, related to exactly the individuals in C under R^-; the source of this arrow is the boundary rectangle which represents the set of all things, often denoted $\top$. Lastly, the *dashed* arrow provides *partial* information about property Q: under Q, the individuals in B are, between them, related to at least the elements in C. Under some circumstances, we may not wish to assert disjointness and subsumption relationships. Concept diagrams make this readily achievable, whilst avoiding clutter, by using multiple rectangles. Fig. 2 visualizes the same information as Fig. 1 except for the disjointness of C with A and B. Spatial relationships only convey information within a single rectangle.

3 Diagrammatic Patterns for Common Constructions

We now demonstrate how to express nine commonly occurring axioms using patterns, contrasting with Description Logic (DL). Common constraints that are imposed on classes are subsumption (subset), disjointness and equivalence. To express that one class is subsumed by another class, concept diagrams use curve containment, reflected in our first pattern; to express class subsumption in DL, one asserts $\mathsf{C}_2 \sqsubseteq \mathsf{C}_1$. Similarly, concept diagrams use curve disjointness (i.e. non-overlapping curves) to express class disjointness, captured in DL by $\mathsf{C}_1 \sqcap \mathsf{C}_2 \sqsubseteq \bot$.

Pattern 1: Class Subsumption Class C_1 subsumes class C_2, Fig. 3.

Pattern 2: Class Disjointness Classes C_1 and C_2 are disjoint, Fig. 4.

Pattern 2 has an obvious generalization to (concisely) assert that n classes are pairwise disjoint (that is, any pair of the n classes are disjoint). Using DL, one axiom is required for each pair of classes to capture this disjointness information: $\mathsf{C}_1 \sqcap \mathsf{C}_2 \sqsubseteq \bot$,...,$\mathsf{C}_1 \sqcap \mathsf{C}_\mathsf{n} \sqsubseteq \bot$, ..., $\mathsf{C}_{\mathsf{n}-1} \sqcap \mathsf{C}_\mathsf{n} \sqsubseteq \bot$; OWL has a more succinct representation: DisjointClasses$(\mathsf{C}_1, ..., \mathsf{C}_\mathsf{n})$. In the diagram below, the

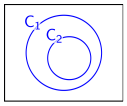


Fig. 3. Pattern 1: Class Subsumption

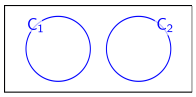


Fig. 4. Pattern 2: Class Disjointness

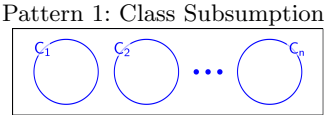


Fig. 5. Pattern 3: General Class Disjointness

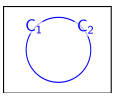


Fig. 6. Pattern 4: Class Equivalence

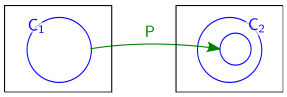


Fig. 7. Pattern 5: All Values From

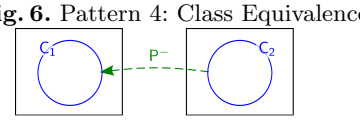


Fig. 8. Pattern 6: Some Values From

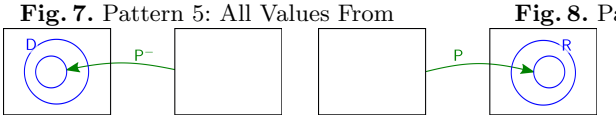


Fig. 9. Pattern 7: Domain **Fig. 10.** Pattern 8: Range

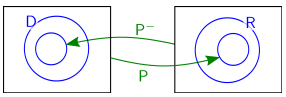


Fig. 11. Pattern 9: D & R

ellipsis indicates the presence of a further $\mathsf{n}-3$ circles labelled in the obvious fashion.

Pattern 3: General Class Disjointness $\mathsf{C}_1, .., \mathsf{C}_\mathsf{n}$ are pairwise disjoint, Fig. 5.

Ontology engineers often want to express that two classes are equivalent. As with the other patterns, there are many semantically equivalent, but syntactically different, concept diagrams that express class equivalence. The following pattern employs two overlaying (completely concurrent) curves.

Pattern 4: Class Equivalence Classes C_1 and C_2 are equivalent, Fig. 6.

In DL, Class Equivalence can be expressed by $\mathsf{C}_1 \equiv \mathsf{C}_2$. Again, the Class Equivalence pattern has an obvious generalization to the n-class case: to express that n classes are equivalent draw n overlaying curves. The number of DL axioms to express many classes are all equivalent to each other increases rapidly, whereas only a single diagram is needed, omitted for space reasons. A common property restriction is to enforce 'All Values From' and 'Some Values From' constraints.

Pattern 5: All Values From All individuals in class C_1 have all values, under property P, from class C_2, Fig. 7

The arrow in the above diagram formally asserts that the image of the property P (considering P as a binary relation), when its domain is restricted to C_1, is a subset of C_2. In other words, the only things that individuals in C_1 are related to, under P, must be in C_2. The use of multiple bounding boxes ensures that no unintended disjointness information between classes is asserted. In DL, the All Values From pattern is captured by $\mathsf{C}_1 \sqsubseteq \forall \mathsf{P}.\mathsf{C}_2$. We also present a pattern for 'Some Values From', expressed in DL by $\mathsf{C}_1 \sqsubseteq \exists \mathsf{P}.\mathsf{C}_2$.

Pattern 6: Some Values From Individuals in class $\mathsf{C_1}$ have at least one value, under property P, from class $\mathsf{C_2}$, Fig. 8.

The above diagram makes use of the *inverse* of property P. To justify the correctness of the Some Values From pattern, consider an individual, $\mathsf{c_1}$, in the class $\mathsf{C_1}$. The pattern must ensure that $\mathsf{c_1}$ has a value, $\mathsf{c_2}$, from $\mathsf{C_2}$ under property P. Well, $\mathsf{c_1}$ has such a value, $\mathsf{c_2}$, if and only if $\mathsf{c_2}$ has the value $\mathsf{c_1}$ under $\mathsf{P^-}$. Equivalently, the image of $\mathsf{P^-}$ when its domain is restricted to $\mathsf{C_2}$ includes at least all of the individuals in $\mathsf{C_1}$, captured by the dashed arrow.

Our last three patterns concern domains and ranges of properties. Firstly, consider the domain, D, of property P. The domain of P is D if and only if the range of the inverse, $\mathsf{P^-}$, of P is D.

Pattern 7: Domain of a Property The domain of property P is D, Fig. 9.

The corresponding DL formalization of this pattern is $\forall \mathsf{P}.\top \sqsubseteq \mathsf{D}$; the construction $\forall \mathsf{P}.\top$ builds the pre-image of the property P. The Domain of a Property pattern employs the same style of construction: the arrow builds the pre-image of P. In DL, the *range* is typically defined by $\top \sqsubseteq \forall \mathsf{P}.\mathsf{R}$. The range can also be defined in DL by constructing the pre-image of the inverse, $\mathsf{P^-}$, of P and asserting that this pre-image is subsumed by R: $\forall \mathsf{P^-}.\top \sqsubseteq \mathsf{R}$. Our Range of a Property pattern constructs the image of P, using an arrow, and asserts that this image is subsumed by the range, R.

Pattern 8: Range of a Property The range of property P is R, Fig. 10.

Pattern 9: Domain and Range of a Property The domain and range of property P are D and R respectively, Fig. 11.

4 Applying the Patterns

We demonstrate the application of the patterns to the Semantic Sensor Networks (SSN) Ontology [2] as a case study. The examples represent just a small fragment of that ontology, but have been chosen to illustrate the application of the patterns above. The SSN ontology, the class MeasurementCapability is subsumed by Property and there are four pairwise disjoint classes: Sensor, Stimulus, Property, and Sensing. SensorInput is equivalent to Stimulus. The Class Subsumption, General Class Disjointness, and Class Equivalence patterns yield the diagrams in Figs 12, 13 and 14 respectively. Regarding property restrictions, the SSN ontology includes the constraint that sensors *detect* only stimuli. The property detects relates individuals in the Sensor class only to individuals in the Stimulus class. This property restriction is an All Values From constraint and the corresponding diagram is in Fig. 15. The SSN ontology also makes plentiful use of Some Values From property restrictions. One example is that sensors *implement* some sensing. The property implements relates individuals in the Sensor class to some individual(s) in the Sensing class. The Some Values From diagrammatic pattern thus gives rise to Fig. 16. Lastly, we demonstrate instances of the Domain of a Property, Range of a Property, and the Domain and Range of a Property patterns. To do so, we make use of a further two classes in the SSN ontology:

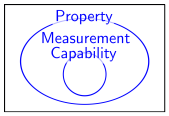


Fig. 12. Subsumption

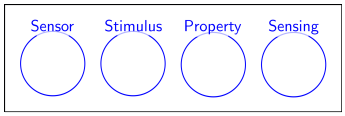


Fig. 13. General Disjointness

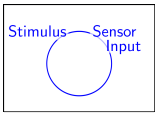


Fig. 14. Equivalence

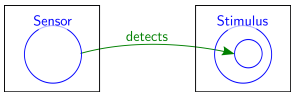


Fig. 15. All Val. From

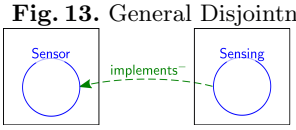


Fig. 16. Some Val. From

Fig. 17. D & R

Situation and Event. There is a property, includesEvent, with domain Situation and range Event shown in Fig. 17.

5 Conclusion

We have presented nine diagrammatic patterns for defining constraints that occur frequently in ontology engineering. These patterns are all formal, since concept diagrams have a fully defined syntax and semantics [7]. There are various avenues for significant future work. A particular goal is to provide tool support for ontology engineering using concept diagrams. This will treat concept diagrams as 'first-class' citizens in the model development process, rather than purely as a visualization of an ontology. We envisage producing a tool that allows the diagrammatic patterns to be accessed. A big challenge is to ensure that the resulting drawn (concrete) diagram has an effective layout. This will build on the now substantial body of work that solves Euler diagram layout problems [6].

References

1. OntoGraf. http://protegewiki.stanford.edu/wiki/OntoGraf, accessed July 2013.
2. M. Compton et al. The SSN ontology of the semantic sensor network incubator group. *Web Semantics: Science, Services and Agents on the World Wide Web*, 17(0):25–3, 2012.
3. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.
4. P. Hayes, T. Eskridge, M. Mehrotra, D. Bobrovnikoff, T. Reichherzer, and R. Saavedra. Coe: Tools for collaborative ontology development and reuse. In *Knowledge Capture Conference*, 2005.
5. M. Horridge. Owlviz. www.co-ode.org/downloads/owlviz/, accessed June 2009.
6. G. Stapleton, J. Flower, P. Rodgers, and J. Howse. Automatically drawing Euler diagrams with circles. *Journal of Visual Languages and Computing*, 23:163–193, 2012.
7. G. Stapleton, J. Howse, P. Chapman, A. Delaney, J. Burton, and I. Oliver. Formalizing concept diagrams. In *Visual Languages and Computing*. KSI, 2013.

The Event Processing ODP

Eva Blomqvist[1] and Mikko Rinne[2]

[1] Linköping University, 581 83 Linköping, Sweden
eva.blomqvist@liu.se
[2] Department of Computer Science and Engineering,
Aalto University, School of Science, Finland
mikko.rinne@aalto.fi

Abstract. In this abstract we present a model for representing heterogeneous event objects in RDF, building on pre-existing work and focusing on structural aspects, which have not been addressed before, such as composite event objects encapsulating other event objects. The model extends the SSN and Event-F ontologies, and is available for download in the ODP portal.

1 Introduction

Layered processing of heterogeneous events [2] with Semantic Web tools [4] sets new requirements to the event models to be used. For example, higher-level composite event objects may encapsulate the lower-level event objects, which triggered the composite abstraction. By defining a comprehensive and structured model for representing event objects, this work addresses current challenges, as discussed in [1], and constitutes a novel contribution in event processing based on Semantic Web technologies. The proposed Content ODP extends current event models to incorporate aspects of (complex) event processing, and additionally facilitates the use of generic SPARQL patterns for event object data management. For a detailed discussion of the requirements underlying the proposed ODP and related work that has been considered, see [5]. Our proposed Event Processing ODP is available in the ODP Portal[3]. The ODP is very general, and therefore applicable to any domain where event processing is to be performed. As with any Content ODP, if needed it can be specialized to include more domain-specific classes and properties suiting that domain, by importing the ODP and adding subclasses, subproperties, and additional domain-dependent axioms in the importing ontology (or ODP).

2 Structure of the Event Processing ODP

An important standardisation effort has been the Semantic Sensor Network (SSN) ontology[4], by the W3C Semantic Sensor Network Incubator group. It is

[3] http://ontologydesignpatterns.org/wiki/Submissions:EventProcessing
[4] http://purl.oclc.org/NET/ssnx/ssn

important that an event processing ODP is fully aligned with the SSN ontology, merely extending it with new concepts. Another important effort is the Event-F ontology[5], describing different aspects of events in the real world. However, what we describe are records of events (information about events in a system) rather than the event itself, c.f. `SensorOutput` of the SSN ontology. Nevertheless, Event-F can be used together with our ODP to connect to descriptions of the event itself, e.g., for exploiting the axiomatization of Event-F. Both the SSN ontology and Event-F are based on DOLCE Ultra Light [6] (DUL). By extending, and aligning to, both of these models, our ODP is also aligned to DUL. Another alignment between the two ontologies has already been made in the SPITFIRE project, producing an extended ontology[7] for describing sensor contexts as well as energy and network requirements, but this extension still lacks the necessary classes and properties for describing complex events, which we add through this ODP.

In the terminology of [3], the *event object* is the system representation, or record, of an event (real or system generated). An event object can be either a *simple event object* or a *complex event object*, depending on if it abstracts (summarizes or represents) other, more low-level, event objects or not. Additionally, a special case of a complex event object is a *composite event object*, which is a complex event object that is actually made up of a set of other event objects, i.e., acting as its parts. A composite event object is always a complex event object, but every complex event object is not necessarily a composite event object. A complex event object can be structurally equivalent with a simple event object, having only the semantic difference of representing other event objects in the network.

In the proposed ODP (core classes and properties illustrated in Fig. 1) we have modelled the `EventObject` class as a subclass of the `dul:InformationObject` (similarly as the `ssn:SensorOutput`). We have aligned our `EventObject` class to the SSN ontology, by expressing that `ssn:SensorOutput` is equivalent to `SensorOutput` in the local namespace, which in turn is a subclass of `EventObject`. The reason for introducing a new `SensorOutput` class is to make the ODP more self-contained, but still make the alignment explicit (even without importing the SSN ontology). A `SensorOutput` is normally a `SimpleEventObject`, however, when considering more complex sensors, such as human sensors entering information into a system, they can be directly entered into the system as complex event objects. `SimpleEventObject` may also consist of other kinds of event objects than a `SensorOutput`, e.g., simulated event objects produced inside a system.

To relate event objects to each other, we introduce a set of object properties (`hasSubEventObject` and `hasDirectSubEventObject`, and their inverses). The object properties are modelled in a way that allows us to keep both a hierarchical sub-event object structure through a non-transitive property (`hasDirectSubEvent-`

[5] http://west.uni-koblenz.de/koblenz/fb4/AGStaab/Research/ontologies/events/model.owl
[6] http://www.loa-cnr.it/ontologies/DUL.owl
[7] http://spitfire-project.eu/ontology/ns/

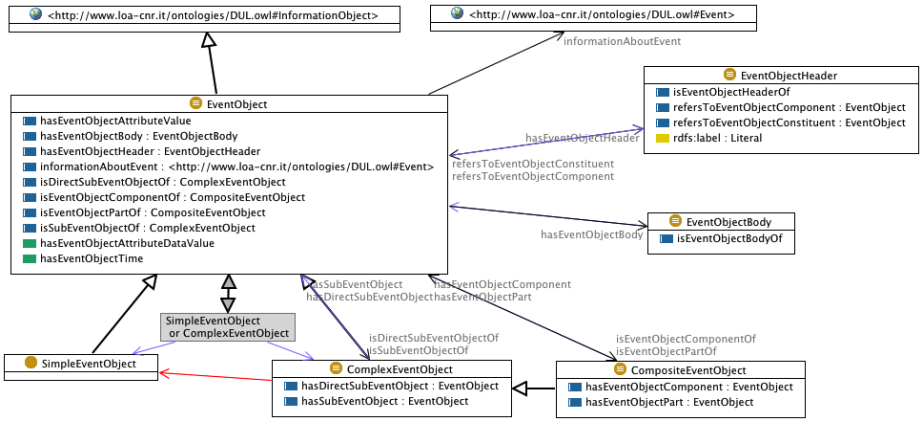


Fig. 1: The core classes of the Event Processing ODP.

`Object`) but (using a reasoner) also allows for directly retrieving all sub-event objects through its transitive superproperty (`hasSubEventObject`). The properties are aligned to DUL, i.e., sub-properties of `dul:hasConstituent` (and `dul:isConstituentOf`). These properties allow us to express event abstraction on the side of the event objects, while a similar structure exists on the side of actual events in DUL, i.e., directly for the `dul:Event` class using `dul:hasConstituent` and `dul:isConstituentOf`. These two parallel structures do not necessarily need to mirror each other, which is actually the main motivation for separating them. For instance, consider a music festival night as an event that occurs in the real world. Intuitively we may, for instance, describe this event as a set of concerts by different artists that are held in sequence on the same stage, which could be modelled using Event-F, the `dul:Event` class and its associated properties for mereological breakdown. However, a system representation of this event, i.e., the *event objects*, may very well display a completely different content and structural breakdown. For instance, we may use sound level sensors to detect that there is some activity on the stage, and make readings every minute, then the music festival night is actually represented by a "loud period" event object in the system, which consists of sub-events that are the individual sound level readings, together with some mechanism detecting that what is heard is actually music. The latter breakdown would then use the properties of the Event Processing ODP, to express the breakdown of the event objects from the system viewpoint. In addition, we also model event objects that are partitioned into components, i.e., whose parts are other event objects. This is modelled in a similar manner, i.e., again with a property structure parallel to Event-F, but this time exploiting the `dul:hasPart` and `dul:hasComponent` properties.

Since the notion of *event object header* and *event object body* is a prominent part of some systems (for a more detailed discussion on the aspect, see [5]), it is important to include in our model, but at the same time accommo-

date systems where this is not present. We include the concepts `EventObjectBody` and `EventObjectHeader` in the ODP (as `dul:InformationObject` subclasses), as well as object properties for relating an `EventObject` to its header and body, and further expressing the content of the header and body. For instance, if a header is used, the relation to other event objects (componency or constituency) can be expressed through a property of the header instead (`refersToEventObjectComponent` and `refersToEventObjectConstituent` respectively). By exploiting OWL property chains, this is then equivalent to directly stating the relation for an individual of `EventObject`.

The actual information content of an `EventObject`, whether organized into a header and body or not, can then be expressed through properties, such as `hasEventObjectAttributeValue` (subproperty of `dul:hasRegion`) and `hasEventObjectAttributeDataValue` (subproperty of `dul:hasDataValue`), or through arbitrary properties that the application case requires. To model different timestamps of the `EventObject`, there are a set of subproperties of `hasEventObjectAttributeDataValue` (in turn subproperty of `dul:hasDataValue`), grouped under a property called `hasEventObjectTime`, i.e., the following four OWL datatype properties: `hasEventObjectSamplingTime`, `hasEventObjectApplicationTime`, `hasEventObjectSystemTime`, and `hasEventObjectExpirationTime`, corresponding to the time points when the event object was sampled (e.g., recorded by a sensor), entered the data stream, arrived in the event processing system via the stream, and any known end time for the event objects validity, respectively. The alignment to Event-F is constructed through a restriction on the `EventObject` class, expressing that any `EventObject` describes some "real" event (`dul:Event`), which then according to the Event-F model has to be a documented event, i.e., a `dul:Event` involved in some `eventf:EventDocumentationSituation`.

3 Example of Usage

As an example usage of the ODP, we may have sensors measuring the water level and flow in different parts of a network of rivers and lakes. That information combined with a weather forecast for rain could be used to derive a flood warning, which would be a complex event object. An RDF file containing an instantiation of the ODP, realizing this small example scenario can be found in the ODP portal[8]. In this example, a flood warning is an instance of `EventObject`, which is a composite event object, encapsulating another `EventObject`, which is a water alert. The water alert, in turn is also composite, encapsulating a water level measurement. The flood warning also refers to a weather event object, but the weather event object is merely referenced (rather than encapsulated). If the flood warning did not encapsulate the triggering event objects, it would still be a complex event object but not a composite event object. Each of the `EventObject` instances have some data attached, and most of them use the header/body

[8] http://www.ontologydesignpatterns.org/cp/examples/eventprocessing/eventexample.owl

structure described previously. In addition, the example shows how external vocabularies can easily be used for expressing `EventObject` instances and data. For example, the weather data uses the meteo vocabulary[9], and to illustrate the alignment to the SSN ontology the water level measurement `EventObject` is modelled entirely using classes and properties from the SSN ontology.

4 Conclusions

We propose a novel Event Processing ODP, i.e., a vocabulary for representing and reasoning over complex and composite event objects, which is needed for further progress in the area of RDF stream processing. The model is aligned to important standards, such as the SSN ontology, and compatible with other event models, such as Event-F, and it is particularly designed so as to be flexible enough to accommodate different event object structures, yet generic enough to allow for expressing generic queries for management of the event objects. For a complete discussion of benefits and relations to the underlying requirements, see the accompanying full paper [5].

Acknowledgments

This work was partially supported by European Commission through the SSRA (Smart Space Research and Applications) activity of EIT ICT Labs[10], and CENIIT at Linköping University through the grant 12.10.

References

1. Corcho, O., García-Castro, R.: Five challenges for the semantic sensor web. Semantic Web 1(1), 121–125 (2010)
2. Etzion, O., Niblett, P., Luckham, D.: Event Processing in Action. Manning Publications (Jul 2010)
3. Luckham, D., Schulte, R.: Event Processing Glossary Version 2.0 (2011), http://www.complexevents.com/2011/08/23/event-processing-glossary-version-2-0/
4. Rinne, M., Abdullah, H., Törmä, S., Nuutila, E.: Processing Heterogeneous RDF Events with Standing SPARQL Update Rules. In: Meersman, R., Dillon, T. (eds.) OTM 2012 Conferences, Part II. pp. 793–802. Springer-Verlag (2012)
5. Rinne, M., Blomqvist, E., Keskisärkkä, R., Nuutila, E.: Event Processing in RDF. In: Proceedings of WOP2013 - Research paper track. CEUR Workshop Proceedings, CEUR-WS.org (2013)

[9] http://inamidst.com/sw/ont/meteo
[10] http://eit.ictlabs.eu/ict-labs/thematic-action-lines/smart-spaces/

The Object with States Ontology Design Pattern

Raúl García-Castro and Asunción Gómez-Pérez

Ontology Engineering Group, Center for Open Middleware
Universidad Politécnica de Madrid, Spain
{rgarcia,asun}@fi.upm.es

Abstract. This paper describes the Object with States content ontology design pattern that allows modeling the different states of an object and the restrictions on such object for its different states. It also presents an example of applying the pattern in a concrete use case in the ALM iStack ontology.

1 Introduction

An object can have different states over time for which different restrictions apply. Examples of objects with different states can be persons (which can be single, then married to another person and later become divorced or a widower) or research papers (which can be submitted for publication and after a round of reviews they can be accepted or rejected).

The goal of the Object with States content ontology design pattern described in this paper is to allow modeling the different states of an object and the restrictions on such object for its different states.

It is out of the scope of this pattern to model other information about object states such as: the time intervals in which an object is in a concrete state, transitions between states, or relationships or dependencies between states.

2 The Object with States Ontology Design Pattern

The Object with States ontology design pattern is a content pattern that aims to satisfy the following ontology requirements:

- Objects must have a unique state.
- Object states must belong to a single collection of non-duplicate elements (i.e., to a set).

Figure 1 depicts the Object with States pattern, which is available online as a reusable OWL ontology[1]. As can be seen, the pattern contains three classes, one for representing objects, another for representing object states, and a third one for representing sets of states. Besides, it contains object properties for relating objects and states (which are subproperties of those defined in the *Situation* pattern[2]) and for relating states and

[1] http://delicias.dia.fi.upm.es/ontologies/ObjectWithStates.owl
[2] http://ontologydesignpatterns.org/wiki/Submissions:Situation

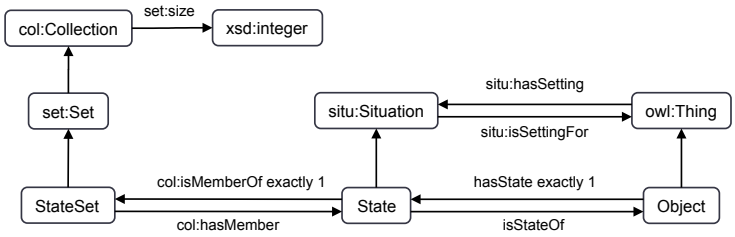


Fig. 1. The Object with states pattern.

sets of states (reused from the *CollectionEntity* pattern[3]) and a datatype property for defining the size of a set of states (reused from the *Set* pattern[4]).

Taking into account some fictitious classes and properties, which serve as an example of how to instantiate the pattern, the following ontology requirements can be satisfied with the pattern:

- The possible object states are: *StateA*, *StateB* and *StateC*.
- An object can have three different states.
- Objects in *StateA* must have at least one value for property *property1*.
- Objects in *StateB* must have at most one value for property *property2*.
- Objects in *StateC* must have exactly one value for property *property3*.

The following steps must be performed for instantiating the pattern (figure 2 presents a fictitious instantiation of it):

1. Represent all the possible states of the object as instances of the *State* class using the Value Partition pattern [1].
2. Define the set of states, which includes all the states, and its size.
3. Define classes to represent the object in each of the states.
4. Apply state-specific restrictions to these classes.
5. Define the *Object* class as a disjoint union of these classes.

Clearly, in the case when an ontology developer just needs to describe the state of an object, only the first of these steps is needed and an object description should just link to the state of the object. Alternatively, the different states could be modeled as literals but that is not recommended because it hinders extensibility; e.g., modeling them as individuals enables using these states in complex class descriptions as in our case.

One advantage of the pattern is that it allows to explicitly define the restrictions that an object must held in each of its states, instead of relying on documentation or software behavior to discover them. Furthermore, it allows simplifying the use of the ontology by hiding the disjoint union of state-specific object classes to end users, so they just have to deal with the *hasState* property and the *State* instances, while keeping the whole ontology for other purposes (e.g., data validation).

[3] `http://ontologydesignpatterns.org/wiki/Submissions:CollectionEntity`
[4] `http://ontologydesignpatterns.org/wiki/Submissions:Set`

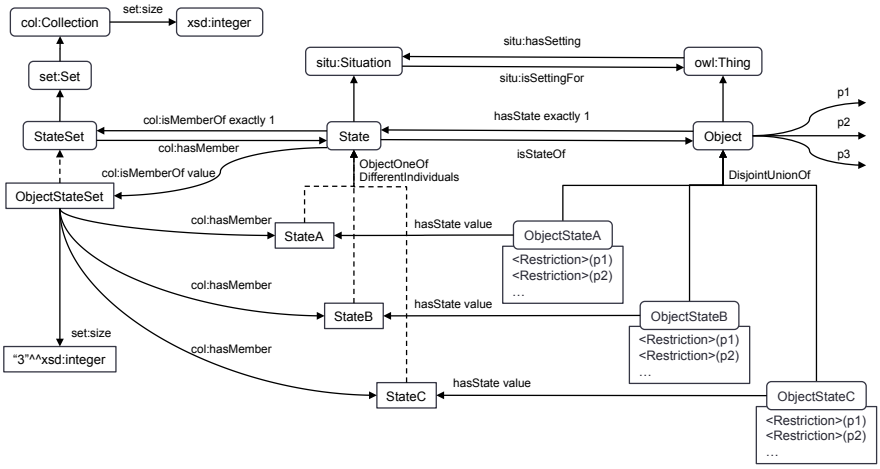


Fig. 2. A fictitious instantiation of the Object with states pattern.

Furthermore, regarding reasoning, the pattern allows checking the consistency of objects according to their states and classifying objects into their corresponding state by means of the *hasState* property (e.g., an object with the *hasState* property with a value of *StateB* can be automatically classified as an instance of *ObjectStateB*).

Even if it is not its primary intended use, the pattern does not disallow to define multiple states for a single object (e.g., a person can have a state of single or married and another state of child, adult, or senior). To instantiate the pattern to that end, the members of each state set should be defined in a subclass of *State* (which would define the value partition) and the classes for the object in each state should be defined in a separate disjoint union of classes.

3 Application of the Pattern

The Object with States ontology design pattern has been applied in the development of the ALM iStack ontology[5], which allows describing entities for software Application Lifecycle Management.

Software defects are one of the main concepts in that ontology. Defects have a concrete lifecycle, shown in figure 3, and in each of its states a defect has a set of required properties. Once a defect is registered into the ALM iStack platform it must have a certain status (usually the status will be *New* upon defect creation) and must satisfy a set of restrictions. In this case, a *New* defect must be assigned to some contributor and an *In-progress* defect must have a priority. These restrictions are propagated to the following states in the lifecycle.

As mentioned in the previous section, the class for registered defects and its subclasses do not need to be explicitly used when interchanging data between the differ-

[5] `http://delicias.dia.fi.upm.es/ontologies/alm-istack.owl`

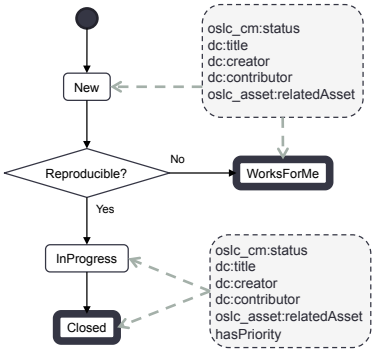


Fig. 3. The lifecycle of a defect.

ent components of the ALM iStack platform, i.e., components can simply talk about defects. These classes have been defined to explicitly specify the restrictions in each defect state and are mainly intended to be used for data validation.

Figure 4 depicts the subset of the ALM iStack ontology that is used to describe the different states of a defect. As can be seen, all those restrictions that are shared by a group of classes have been defined in the higher class in the hierarchy.

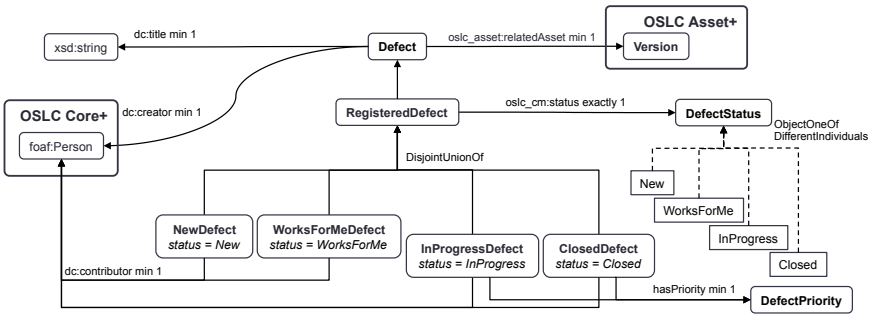


Fig. 4. The Object with states pattern applied in the ALM iStack ontology.

Acknowledgements

The authors are partially supported by the ALM iStack project of the Center for Open Middleware.

References

1. Rector, A.: Representing Specified Values in OWL: “value partitions” and “value sets”. W3C Working Group Note 17 May 2005. http://www.w3.org/TR/swbp-specified-values/

License Linked Data Resources Pattern *

Víctor Rodríguez-Doncel, Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez, and María Poveda-Villalón

Ontology Engineering Group, Universidad Politécnica de Madrid, Spain
{vrodriguez,mcsuarez,asun,mpoveda}@fi.upm.es

Abstract. Linked Data resources can be referenced by rights expressions or access control policies. Based on the common model found in six existing rights expression languages and revolving around the n-ary relation pattern, the License Linked Data Resources pattern is presented as a solution to describe existing licenses and ad-hoc rights expressions alike and valid for open and not open scenarios.

Keywords: Ontology design patterns, Linked Data, rights expressions

1 Introduction

Linked Data (LD) assets (RDF triples, graphs, datasets, mappings...) can be object of protection by the intellectual property (IP) law, the database law or its access or publication be restricted by other legal reasons (personal data protection, security reasons, etc.) [1]. Publishing a rights expression along with the digital asset, allows the rightsholder waiving some or all of the IP and database rights (leaving the work in the public domain), permitting some operations if certain conditions are satisfied (like giving attribution to the author) or simply reminding the audience that some rights are reserved. Additionally, LD resources can be conditionally available after the evaluation of access control policies [2], expressing who can act what actions under which circumstances.

After the comparison in [3] of six important rights expressions and policy languages (ODRL, MPEG-21 REL, XACML, ccREL, MPEG-21 MVCO and WAC), enough commonalities were found to extract a common underlying model, which could satisfy all of them. Based on that model, this poster paper describes a content ontology design pattern, named *License Linked Data Resources* (LLDR), to model licensing issues over Linked Data resources.

2 Pattern description

2.1 Intent and requirements

The intent of the content pattern Licence Linked Data Resources is to represent the relation that exists among a rights expression, an action, an agent, a LD

* This research is supported by the Spanish Ministry of Science and Innovation through a Juan de la Cierva fellowship and the project BabelData (TIN2010-17550).

resource and a condition. In particular, the core idea of the pattern is to model: *a rights expression which allows/prohibits/obliges to make an Action (Right) to an Agent over a LD resource under a condition.*

The LLDR pattern is commited to satisfy the following requirements:

- To keep the structure present in other Rights Expression Languages
- To be able to represent existing known licenses (Creative Commons licenses...)
- To support database rights: extraction and re-utilization
- To support privacy law (personal data handling) and the right to access
- To support IP law rights: reproduction, distribution, and transformation
- To support these right declarations: unconditionally waiving rights, restating that some rights are reserved, and licensing rights subject to conditions
- To support existing licensing practices for RDF resources
- To support these business models: open data business models, non open data business models, and hybrid models.

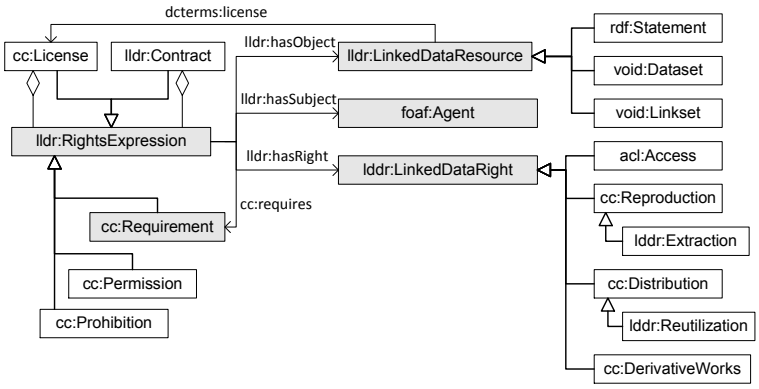


Fig. 1. License Linked Data Resource Pattern

2.2 Solution Description

The most convenient way to represent the information described before is to use the so-called n-ary relation pattern, which addresses these situations: *"(a) a binary relationship that really needs a further argument; (b) two binary relationships that always go together and should be represented as one n-ary relation; and (c) a relationship that is really amongst several things"* ([4]). One of the proposed patterns for representing n-ary relations consists of introducing a new class for the relation and links to all the participants in the relation. Indeed, the LLDR content pattern is inspired on the third consideration shown in the description of n-ary relations from the W3C Semantic Web Best Practices Group

Prefix	*Namespace*
void	http://www.w3.org/TR/void/
cc	http://creativecommons.org/ns#
foaf	http://xmlns.com/foaf/0.1/
acl	http://www.w3.org/ns/auth/acl
gr	http://purl.org/goodrelations/v1
dcterms	http://purl.org/dc/terms/

Table 1. Prefixes and namespaces

in [4]. Fig. 1 shows the LLDR pattern, where the core elements of the n-ary relation are grayed, Table 1 shows the namespace of some used vocabularies.

All the relations revolve around the `lldr:RightsExpression` element, this class being the *qualified relation*[1]. This class has the direct relations with the `lldr:LinkedDataResource` (a superclass declared to embrace the LD information units of `rdf:Statement`, `void:Dataset` and `void:Linksets`), the `lldr:LinkedDataRight`, the `cc:Requirement` and `foaf:Agent`. Prohibitions, permissions and requirements are rights expressions themselves. The `LinkedDataRights` is a superclass representing the applicable rights to Linked Data resources: IP rights (`cc:Reproduction`, `cc:Distribution` and `cc:DerivativeWorks`), database rights (`lldr:Extraction` and `lldr:Reutilization`) and the mere access (`acl:Access`).

Rights expressions appear naturally in groups and not separately. Common licenses and typical authorizations are actually aggregations of atomic rights expressions. For this reason, `cc:License` and `lldr:Contract` are both subclasses and containers of rights expressions. The aggregation relationship can be represented in OWL using a *partOf-whole relation* pattern , and consequently a `partOf` object property has been declared. As a final requirement, resources must be directly linkable to licenses, as this is a common practice already in use (through a `dcterms:license` or a `cc:license` property), and in which case the rights expression does not need to include a specific resource.

3 Pattern Usage Example

In the following examples, classes are represented with boxes, relations with arrows and individuals with ellipses. Fig. 2 represents how a RDF dataset is attributed a known license. This example is currently in use by the Linked Data community (albeit not massively). Fig. 3 represents how the right of extraction (copying a database) is waived exclusively to `myAgent`. Fig. 4 represents how the access to an important reificated statement (the forecast for the stock market price of Google) is offered for 100 €, using the GoodRelations vocabulary.

[1] http://patterns.dataincubator.org/book/qualified-relation.html

Fig. 2. Attributing a known license to an RDF dataset

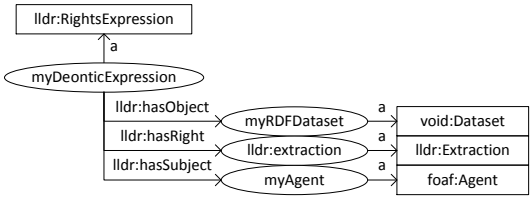


Fig. 3. Waiving a right on a dataset for a friend

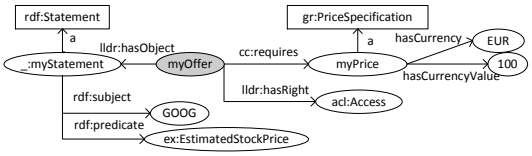


Fig. 4. Offering the access to an important RDF statement

4 Summary and Outlook

The content pattern Licence Linked Data Resources (LLDR) provides a mechanism to represent rights expressions to be applied for Linked Data resources. Having been recently published[2], the immediate goal is declaring its relationships with the ODRLv2 ontology and studying the compatibility with the key elements of other relevant vocabularies like LiMO, L4LOD or ODRS[3].

References

1. Rodríguez-Doncel, V., Gómez-Pérez, A., and Mihindukulasooriya, N.: Rights declaration in Linked Data. in Proc. of the 3rd Int. Workshop on Consuming Linked Data. (2013)
2. Costabello, L., Villata, S., Rocha, O. R. and Gandon, F. Access Control for HTTP Operations on Linked Data. In The Semantic Web: Semantics and Big Data (pp. 185-199). Springer Berlin Heidelberg (2013)
3. Rodríguez-Doncel, V., Suárez-Figueroa, M. C., Gómez-Pérez, A., and Poveda, M.: Licensing Patterns for Linked Data. in Proc. of the 4th Int. Workshop on Ontology Patterns (to appear) (2013)
4. Noy, N., Rector, A., Hayes, P., and Welty, C.: Defining n-ary relations on the semantic web. W3C Working Group Note (2006)

[2] http://oeg-dev.dia.fi.upm.es/licensius/static/lldr.html
[3] http://data.opendataday.it/LiMo, http://ns.inria.fr/l4lod/v2/l4lod_v2.htm and http://schema.theodi.org/odrs/ respectively

Abstracting *Transport* to an Ontology Design Pattern for the Geosciences

Brandon Whitehead[1], Benjamin Adams[1], Mark Schildhauer[2], Charles Vardeman[3], Werner Kuhn[4], Adam Shepard[5], and Krishna Sinha[6]

[1] Centre for eResearch, University of Auckland
[2] National Center for Ecological Analysis and Synthesis, UCSB
[3] Center for Research Computing, University of Notre Dame
[4] University of Münster
[5] Woods Hole Oceanographic Institution
[6] Virginia Polytechnic Institute and State University

Abstract. A core concept in geoscience/physical science research is the concept of transport. We present an ontology design pattern for the notion of *transport* in the geosciences using natural language coupled with a concept map. The top level concepts of the Transport Pattern are *transport entity*, *transport mechanism*, and *transport event*. These concepts are described in detail, and a brief example is provided to illustrate the usefulness of the pattern.

1 Introduction & Related Work

The term *transport* is used to describe a variety of phenomena from different contexts, and occurring across varying spatial scales. In the field of transportation, transport primarily refers to the movement of people or goods within an infrastructure such as a road or airline network. In physical and chemical systems, transport refers to the movement of molecules or other physical matter, energy, or momentum, from one system to another. In computer networking, packets of digital information are transported. We also talk about humans transporting thoughts, ideas, and opinions through communication. In all these cases, there is movement of some entity via a transport mechanism from one place to another.

The *Semantic Transport* Ontology Design Pattern (ODP) is designed as a basic, extensible foundation for modelling transport concepts and relations in an ontology [4, 7]. In the geosciences, transport is a prevalent concept. Within the popular Semantic Web for Earth and Environmental Terminology (SWEET) ontology[7] [8], for example, there are at least 22 different concepts that reference "transport" in some form. By using it as a template for describing the relations between the mechanisms and entities involved in transport events, the *Semantic Transport* ODP can enrich descriptions of scientific data sets to aid in their interoperability, and re-use; as well as for data mining.

[7] http://sweet.jpl.nasa.gov/ontology/

The *Semantic Transport* ODP is designed to be compatible with other ODPs generated during the GeoVoCamp[8] series of workshops. Previous workshops focused on a range of geo-spatial topics from cartographic map scaling [2] to semantic trajectories [5]. Of particular note, the *Semantic Transport* ODP can operate in tandem with the the *Semantic Trajectory* pattern [5] as the former describes the entity and energy of transport, and the latter describes the path along which the transport occurred.

The *Semantic Transport* pattern is also conceptually related to the proposed *Move* ontology design pattern[9] that is derived from the CIDOC model [3]. The *Semantic Transport* pattern, however, decouples the source energy from the entity being displaced while capturing their interdependence.

2 Transport Pattern

In this section we present the core elements of the *Semantic Transport* pattern, and describe how it can be extended to cover two different types of transport: active and passive. We focus on applying the *Semantic Transport* ODP in the context of physical systems, though it may also be useful for other domains, e.g. describing cultural transmission.

2.1 Core elements

The *Semantic Transport* pattern consists of three core concepts: Event, Entity, and Mechanism (see Manchester OWL syntax following this paragraph). The *TransportEvent* acts as the top level concept for the pattern. A *TransportEvent* describes a specific *transport* phenomenon, as movement of some mass or energy (measurable entity) from one location to another, based on a common and persistent frame of reference. Induction of the mass or energy movement can arise from the transported entity itself, or from external sources. The *TransportEvent* thus has two main parts, *TransportEntity* and *TransportMechanism*. The *TransportEntity* concept represents the identity of the circumscribed portion of energy or mass that is moved. The *TransportMechanism* concept captures the nature of the source that acts upon the *TransportEntity*, and thus induces a *TransportEvent*.

```
Class: TransportEvent
TransportEvent SubClassOf owl:Thing
TransportEntity SubClassOf partOf some TransportEvent
TransportMechanim SubClassOf partOf some TransportEvent

Class: TransportMechanim
TransportMechanim SubClassOf owl:Thing
TransportMechanim SubClassOf partOf some TransportEvent
```

[8] http://vocamp.org/wiki/GeoVoCampSB2013
[9] http://ontologydesignpatterns.org/wiki/Submissions:Move

```
Class: TransportEntity
TransportEntity SubClassOf owl:Thing
TransportEntity SubClassOf partOf some TransportEvent
```

The *TransportEvent* has one top level property, the *referenceFrame* (see Manchester OWL syntax following this paragraph). The *referenceFrame* provides context to the pattern by specifying spatial and temporal qualities of any associated observations via the specification of time and location information associated with the *TransportEvent*. As time and location can be fixed or relative, abstracting the property types serves to facilitate semantic interoperability between disparate data entities.

```
ObjectProperty: referenceFrame
referenceFrame Domain TransportEvent
referenceFrame Range TransportEvent
```

It is worth noting that even if the transported entities were to return somehow to their exact place of departure, they still participated in a *TransportEvent*, even if their initial and final locations result in no net change in location.

2.2 Extending the Transport Pattern

Figure 1 illustrates the *Semantic Transport* pattern along with a few logical extensions to illustrate how the pattern might be used. The pattern constructs are depicted in figure 1 using black text in translucent shapes. Further, all classes in the figure are represented using an oval, and each square box delineates a property. In most cases there will be interest in a more specific description of the event. These additional aspects are accommodated by extending the pattern through subclassing of the existing *TransportMechanism* and *TransportEntity* concepts, while adding properties to the *referenceFrame* (Figure 1). These additional pragmatic components are included in figure 1. Class symbols with red text are examples of the nomenclature that may be used as a *TransportMechanism*. Sub-properties with blue text illustrate examples of modules that may be described as part of the constructs comprising the *referenceFrame*.

The *TransportMechanism* can be usefully subclassed into specialised topics, such as in the case of involving disjoint classes such as *PassiveTransport* and *ActiveTransport*. Classic examples of *PassiveTransport* include, e.g. diffusion or osmosis, while *ActiveTransport* would include, e.g. ATP pumps or air travel [1].

The *referenceFrame* is a non-trivial property. Geoscience phenomena often exhibit unique statistical signatures as mechanical, chemical, and biological processes work in tandem, or asynchronously, through time as they tend toward equilibrium [6]. This property is significant in that it serves to preserve the spatiotemporal information necessary to maintain a consistent granularity of context throughout the pattern.

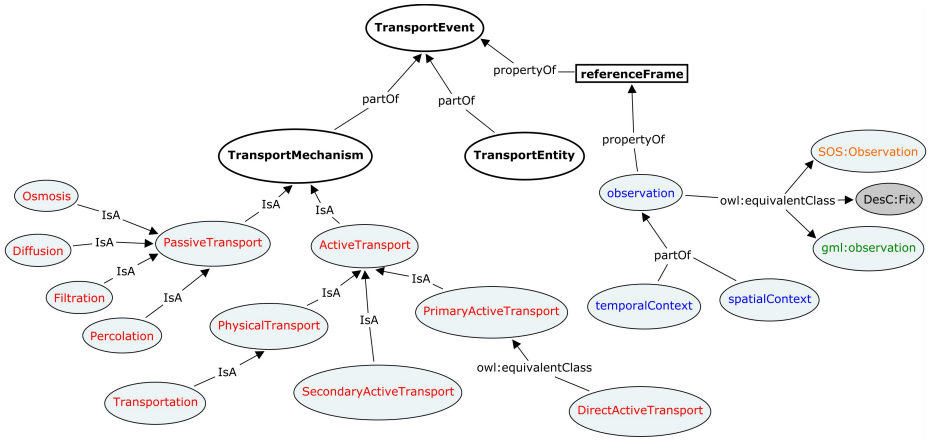


Fig. 1. Transport Pattern extended. Each class in the figure is represented by an oval, and each square box delineates a property. The *Semantic Transport* pattern constructs are depicted using black text in translucent shapes. Classes using red text are logical subclasses, while sub-properties are styled using blue text. Three equivalent class relations are represented on the right of the figure, each one illustrating how the *referenceFrame* can connect to other established schemas.

Further, an *observation* (as it relates to the *referenceFrame* in figure 1) is considered semantically equivalent to other well established geoscience explications, namely the observation entities associated with the Sensor Observation Service[10] (SOS) and the Geography Markup Language[11] (GML), as well as the concept of a "fix" in the *Semantic Trajectory* pattern [5].

Of course, further subclassing will likely be necessary for this pattern to connect, and be useful, to much of the disparate data available to geoscientists. The current framework is complete and extendable. By creating logical semantic equivalences to constructs already used throughout the domain, the *Semantic Transport* pattern can be a powerful module when mining and filtering large data stores.

3 Summary and Future Work

In this paper we presented an ontology design pattern to describe transport phenomena. The core *Semantic Transport* ODP is deliberately simplified to essential elements, to be applicable to a wide range of use cases in the physical sciences. We described how the Transport pattern can be extended for Active and Passive transport and illustrate briefly how it might be used to interoperate over large disparate geoscience data.

[10] http://www.opengeospatial.org/standards/sos
[11] http://www.opengeospatial.org/standards/gml

Next steps will involve how the ODP can be filled out to describe data for a variety of use cases, as well as application and usability testing. An important future extension to the pattern for application to the physical sciences will include explicating the relationship between concepts such as *system* and *energy input* (in terms of entropy of the system).

References

1. Berg, H.C.: Random walks in biology. Princeton University Press (1993)
2. Carral, D., Scheider, S., Janowicz, K., Vardeman, C., Krisnadhi, A., Hitzler, P.: An ontology design pattern for cartographic map scaling. In: Cimiano, P., Corcho, O., Presutti, V., Hollink, L., Rudolph, S. (eds.) The Semantic Web: Semantics and Big Data, Lecture Notes in Computer Science, vol. 7882, pp. 76–93. Springer Berlin Heidelberg (Jan 2013)
3. Doerr, M.: The cidoc conceptual reference module: an ontological approach to semantic interoperability of metadata. AI magazine 24(3), 75 (2003)
4. Gangemi, A.: Ontology design patterns for semantic web content. In: The Semantic Web - ISWC 2005. Lecture Notes in Computer Science, vol. 3729. Springer (2005)
5. Hu, Y., Janowicz, K., Carral, D., Scheider, S., Kuhn, W., Berg-Cross, G., Hitzler, P., Dean, M.: A geo-ontology design pattern for semantic trajectories. In: COSIT 2013. Springer (2013)
6. Kastens, K., Manduca, C., Cervato, C., Frodeman, R., Goodwin, C., Liben, L., Mogk, D., Spangler, T., Stillings, N., Titus, S.: How geoscientists think and learn. Eos. Trans. AGU 90(31), 265–266 (2009)
7. Presutti, V., Gangemi, A.: Content ontology design patterns as practical building blocks for web ontologies. In: Li, Q., Spaccapietra, S., Yu, E.S.K., Olivé, A. (eds.) ER. Lecture Notes in Computer Science, vol. 5231, pp. 128–141. Springer (2008)
8. Raskin, R.G., Pan, M.J.: Knowledge representation in the semantic web for earth and environmental terminology (SWEET). Computers & Geosciences 31(9), 1119–1125 (2005)