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

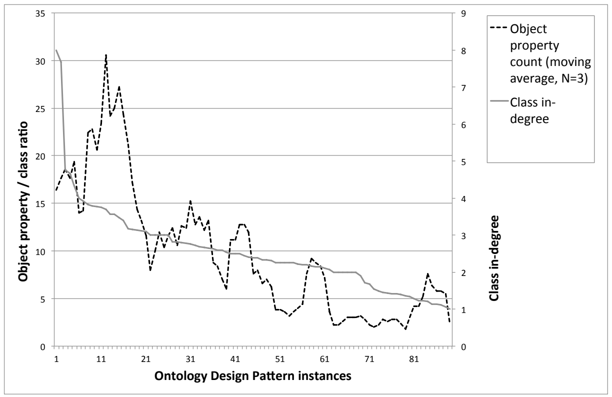

Fig. 1. Class in-degree and object property per class distributions

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

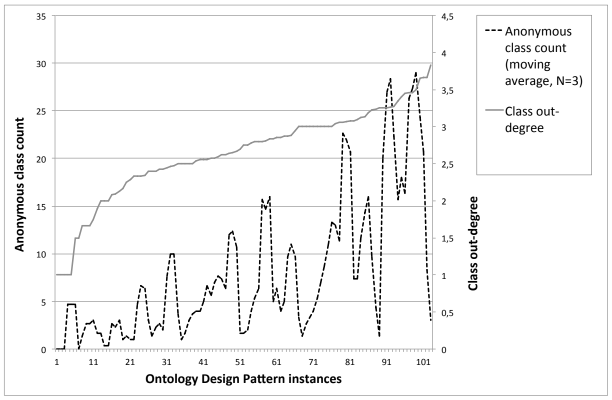

Fig. 2. Class out-degree and Anonymous class count distributions

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