# Ontology Design Pattern Detection - Initial Method and Usage Scenarios

Muhammad Tahir Khan
*School of Engineering, Jönköping University*
*P.O. Box 1026, SE-551 11 Jönköping, Sweden*
*Email: khmu09mt@student.hj.se*

Eva Blomqvist
*StLab, ISTC-CNR*
*via Nomentana 56, 00161 Roma, Italy*
*Email: eva.blomqvist@istc.cnr.it*

***Abstract*—Ontology Design Patterns (ODPs) are emerging as an important support for ontology engineering. In this paper, we show how a method for detecting Content ODPs in existing ontologies can be used as a means to characterize online ontologies, e.g., for finding, browsing and analyzing them, as well as a means of analyzing an ontology being built, by detecting partial instantiations of a Content ODP in that ontology. The main contribution of this paper is the simple but effective method for pattern detection, together with its initial evaluation, as well as the study made on online ontologies providing an overview of Content ODP usage in real-world ontologies as well as a proof-of concept of the proposed method.**

***Keywords*-Ontology Design Patterns; Ontology Engineering;**

## I. Introduction

Ontology Design Patterns (ODPs) are emerging as an important support for ontology engineering. On the semantic web, ontologies are no longer only constructed by developers having a background in logical languages and knowledge modeling. On the contrary, ontologies are commonly drafted by software engineers or found online, combined, and reused. Some small and frequently reused ontologies exist, e.g., the foaf ontology[1], however selecting and reusing larger and more complex ontologies is still a challenging task. Finding reusable ontologies is facilitated by ontology search engines, however, to understand and assess the ontologies is still up to the user, as well as formulating the keyword query to retrieve an accurate search result.

ODPs provide encoded best practices that can facilitate the construction of high-quality ontologies, despite lack of experience and deep knowledge of the logical languages (e.g., as experimentally shown in [2]). However, certain types of ODPs, e.g, Content ODPs, also come with a ‘reference implementation’, i.e., a small reusable component (usually represented in OWL [3]). A collection of such Content ODPs can be found in the *ODP Portal*[4], a wiki portal supporting the collection and management of ODPs.

In this paper, we show how a method for detecting such Content ODPs in existing ontologies can be used as a means to characterize online ontologies, e.g., for finding, browsing and analyzing them, as well as a means of analyzing an ontology being built, by detecting partial instantiations of a Content ODP in that ontology. The main contribution of this paper is the simple but effective method for pattern detection, together with its initial evaluation, as well as the study made on online ontologies providing both an overview of Content ODP usage in real-world ontologies as well as a proof-of-concept of the proposed method. In the following section, we describe Content ODPs in more detail. Section III describes related work, as well as two usage scenarios motivating our approach. In Section IV we present the pattern detection method, and its experimental validation is presented in Section V. Finally, in Section VI we conclude the paper and outline future work opportunities.

## II. Content Ontology Design Patterns

There exist different types of ODPs having different characteristics, e.g., focusing on logical language constructs, architecture issues, naming, or efficient provision of reasoning services; for details on ODP types see [5], [6]. However, in this paper we focus on Content ODPs. Content ODPs are small ontologies with explicit documentation of design rationales, which can be used as building blocks in ontology design [5], [6]. As an example, we describe a Content ODP that is called *Agent Role*. It represents the relation between agents, e.g., persons, and the roles they play, e.g., professional roles such as researcher and teacher, as well as personal ones such as father and friend. Figure 1 shows an illustration of the OWL building-block representing this Content ODP. Content ODPs are collected and presented in different catalogues, such as the *ODP Portal*. In addition to their diagrammatic representation Content ODPs are described using a number of catalogue entry fields (c.f. software pattern templates), such as *name*, *intent*, *covered requirements*, *consequences*, and *building block* (linking to an OWL realization of the pattern). Reusing Content ODPs is a special case of ontology reuse, when the elements of the Content ODP are specialized, e.g., subclasses and subproperties that use domain-specific terminology are added, and more specific axioms are included.

## III. Related Work and Motivation

The detection and analysis of *naming patterns* in ontologies was proposed in [7], where labels and other lexical entries are analyzed, e.g., for supporting refactoring. Although related in its aim, the approach uses lexical patterns to analyze the logical structure, while we use Content ODPs

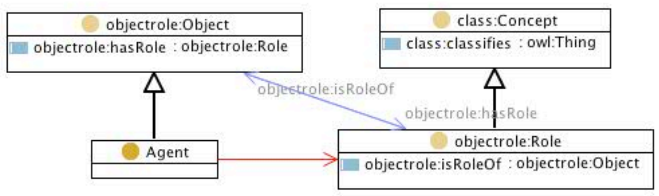
objectrole:Object
objectrole:hasRole : objectrole:Role
class:Concept
class:classifies : owl:Thing
objectrole:isRoleOf
objectrole:hasRole
Agent
objectrole:Role
objectrole:isRoleOf : objectrole:Object

Figure 1. The graphical representation, in UML, of the **Agent Role** ODP (the unlabeled arrow between Agent and Role representing disjointness).

as input. The approach in [8] for detecting *logical patterns* in ontologies using SPARQL queries is not applicable to our scenario since we are not only dealing with the logical constructs of the ontology, but the actual content, i.e., it is impossible to pose appropriate queries until the correct terminology has been established. In [9] a universal pattern language is introduced to monitor and detect constraint violations during ontology modeling, however, the approach is not focused on pattern detection in existing ontologies but rather detection of violations of the patterns being reused.

Our previous work includes OntoCase [5], a method for automatic ontology enrichment based on Content ODPs, mainly focused on enhancing ontologies generated from textual resources. The current method is a further development of the methods used in OntoCase. However, while OntoCase focused on finding matches that were 'useful enough' for interpreting the unrelated elements[1] of the input ontology, our focus in this paper is on simply finding instantiations of Content ODPs. OntoCase also applies an elaborate ranking scheme, which would be too computationally expensive to apply on a web scale, hence, its current implementation with focus on pattern ranking makes a direct comparison between OntoCase and the method proposed in this paper unrealistic.

Another approach that attempts to enrich ontologies by finding partial matches of Content ODPs and subsequently 'completing' the ontology by (automatically) adding the remaining part of the ODP as axioms in the ontology, is found in [11]. This approach does not assess if the missing parts are actually relevant and appropriate for that ontology, and where [11] chooses to add anything possible without evaluation, we leave such decisions up to the user by simply providing the pattern as a reference. Additionally, [11] uses a logical approach for the matching that involves both logical reasoning and manual pre-processing of the ODPs, hence it is not obvious that it will be feasible on a web scale.

## A. Application Scenarios

Approaches exist for finding reusable ontologies, e.g., in the form of semantic web search engines such as Watson [12], Sindice [13], and SWOOGLE [14], and ontology repositories such as [15]. Recent improvements of search engines allow for simple visualization and assessment of the ontologies, e.g., by providing basic information on the size, language and complexity, as well as displaying key concepts [16]. Nevertheless, it is a difficult task to assess the usefulness of the ontology for a particular case. Moreover, merely to pose an appropriate keyword query to the system is challenging, since the index is based on the particular terminology of the ontology. Attempts have been made to introduce the 'query-by-example' paradigm into ontology search engines [17], however, in this case based on a user-developed model of the query and not considering specialization/generalization in the matching. In this paper, we propose to apply Content ODP detection for using ODPs as queries to find online ontologies that (partly) realize or specialize that Content ODP. This will address the need of users already knowing what kind of modelling issue they are interested in, wanting to find online ontologies containing particular solutions for that modelling problem. One could imagine ontology repositories that are browsable by means of the different ODPs the ontologies contain. In addition, it provides an interesting possibility to study the type of modelling solutions applied in online ontologies, as well as to study the 'support' that Content ODPs have in online ontologies (see Section V-C1).

A second scenario is concerned with novice users building an ontology 'from scratch'. Studies such as [2] show that ODP selection is a problem that hampers the designers in fully exploiting the benefits of ODPs, hence, additional tool support should be provided for Content ODP selection. Recently some support has appeared, in the form of the XD Tools [18] for the NeOn Toolkit [19]. XD Tools currently provide ODP registry browsing and search facilities for retrieving Content ODPs based on a keyword query. The semantic vector search service extends standard keyword indexing, but does not take into account the fact that ODPs are usually more abstract than the terminology in the ontology. In this scenario we have a draft ontology and a set of Content ODPs, and wish to find some patterns that are already partially realized within the ontology. Hence, we try to detect occurrences of each pattern in the draft ontology, and if such occurrences are found we propose the pattern to the user, who can study the pattern and evaluate his or her solution against the best-practices that the pattern describe. Although other approaches have been proposed, to use patterns to enhance draft ontologies, they have either been purely automatic (e.g., the OntoCase approach [5]) or using a heavy logical approach also requiring the manual pre-processing of the ODPs, as in [11]. We propose the use of a light-weight method that will easily detect simple modelling attempts of a novice user, who receives a list of possible patterns of interest to be used as a means for evaluating or enhancing the ontology by specializing the Content ODPs found (see Section V-C2).

[1]In this paper the term *ontology element* (formally defined in [10]) denotes formal expressions used to represent any entity, e.g. named classes and properties are elements, as well as class definitions and restrictions.

## IV. Content ODP Detection

The method for Content ODP detection is based on OntoCase [5]. The basic principle of OntoCase pattern detection is to identify specializations of Content ODPs based on matching the terminology used for properties and classes, as well as matching the property structure through domain and range restrictions.The graph based pattern matching of OntoCase is currently not considered in this paper because of its complexity, e.g., processing time. The approach presented in this paper is thereby both an extension and a simplification of the OntoCase approach. The main extension is that the current approach not only matches domain and range restrictions but all axioms of the ontology, while a simplification is that the current approach uses less computationally expensive algorithms, e.g., the OntoCase ranking has been removed, instead relying on simple matching percentages.

The proposed approach uses three main methods for detection; (1) import detection, (2) direct matching, and (2) indirect matching. Import detection is the trivial detection of an explicit import of a pattern URI in the chain of the ontology's import closure. We are aware of the fact that imported patterns might not be used in the ontology, but taking this into account is still future work. The direct matching aims to detect clones of the pattern, including partial clones, existing in the ontology, while the indirect matching aims to detect specializations of a pattern in the ontology, i.e., where the pattern classes and properties have been exchanged for more (domain-) specific ones. The procedure is described using pseudocode in Figure 2.

**Require:** A pattern $p$, an ontology $o$, and the thresholds of class/property/axiom matches $t_1, t_2, t_3$.
**Ensure:** The matching percentages $P_c, P_p, P_a$ or $null$ if match was below threshold.

```
1: if import closure of $o$ contains an owl:import of $p$ then
2:     return $P_c = 100, P_p = 100, P_a = 100$
3: else
4:     $oClassNames = getClassNames(o)$
5:     $\ldots$
6:     $extend(oClassNames, oPropertyNames)$
7:     $extend(pClassNames, pPropertyNames)$
8:     $classMatches, propMatches, axiomMatches = \emptyset$
9:     for each set in $pClassNames = pElement$ do
10:        $pairs = stringMatch(pElement, oClassNames)$
11:        if $pairs \neq \emptyset$ then
12:            $classMatches = classMatches + pairs$
13:    $P_c = percentage(classMatches, p)$
14:    for each set in $pPropertyNames = pElement$ do
15:        $pairs = stringMatch(pElement, oPropertyNames)$
16:        if $pairs \neq \emptyset$ then
17:            $propMatches = propMatches + pairs$
18:    $P_p = percentage(propMatches, p)$
19:    for each axiom of $p = a$ do
20:        $pairs = tripleMatch(a,$ axioms of $o)$
21:        if $pairs \neq \emptyset$ then
22:            $axiomMatches = axiomMatches + pairs$
23:    $P_a = percentage(axiomMatches, p)$
24:    if $(P_c > t_c)\&(P_p > t_p)\&(P_a > t_a)$ then
25:        return $P_c, P_p, P_a$
26:    else
27:        return $null$
```

Figure 2. Detection procedure.

If no import was found the detection procedure starts by retrieving all the terms to be used, e.g., local names (i.e., excluding the namespaces) and labels of classes and properties. The *extend()*-function uses heuristics to extend the terms into term sets, each representing one original element, e.g., class or property, of the ontology or the pattern respectively. The intuition is that we need to allow for certain variations in the matches, i.e., for the direct matching this would include heuristics for capitalization and morphological variations, while for the indirect matching this includes also term specialization, for instance, using background knowledge such as WordNet [20]. The *stringMatch()*-function performs exact string matching on the elements of the pairs $A \times B$, where $A$ is one of the extended term sets from the pattern and $B$ is the set of all such extended term sets of the ontology. Depending on the previous application of heuristics, this may return several matching pattern element-ontology element pairs. The axioms are matched based on matching the subject-predicate-object structure of their contained triples, i.e., the *tripleMatch()*-function above. It converts all the axiom constituents to sets of strings, similar to above but taking into account the uniqueness of the OWL constructs, e.g., 'reserved words' such as `disjointWith` does not need to be extended with synonyms and lexical variations, then matches the string sets for subject, predicate, and object respectively. The *percentage()*-function calculates the fraction of classes/properties/axioms that are in the matched set, with respect to the overall number of the pattern or ontology. If above the threshold values, the match is confirmed. If needed, the details of each match can also be recorded, although not shown in the procedure above.

## V. Experimental Validation

To allow for a proof-of concept validation of the approach it has been implemented as a stand-alone Java application, and applied to two different datasets.

### A. Implementation

The method described in Section IV has been implemented using Java. The implementation exploits the OWL API (3.0) [21] for handling the ontologies, the Watson API [22] for retrieving online ontologies, and JAWS [23] for interfacing WordNet and supporting the indirect matching. In this implementation the extension heuristics for the direct matching are restricted to (i) ignoring capitalization, and (ii) recognizing the most common ways of replacing spaces in element names, e.g., using the camel convention or _ instead of spaces. For the indirect matching these heuristics are extended by using JAWS. Through a simple lookup mechanism the corresponding synset of every element name or label in the pattern is retrieved from WordNet, and all specializations (hyponyms) of that synset are additionally added to the extended term set. No disambiguation of the terms are currently performed, i.e., all possible chains of synsets are used, but since only specialization and not generalization is considered this is a manageable set. In the current implementation only direct matching is performed on the axiom triples, i.e., no specialization of these are allowed.

### *B. Data Collection*

Two sets of ontologies were collected, each corresponding to one of the usage scenarios described previously.

*1) Online Ontologies:* The online ontologies were retrieved through the keyword search feature of the Watson API [12]. The set of keywords entered are matched to the local names, labels, comments, or literals of elements occurring in semantic documents, i.e., ontologies. Based on query logs of the Watson search engine we collected a list of the 70 most used search keywords. From this list we further selected the 50 keywords that returned the highest number of ontologies, in order to get a sample that represents typical search results. A list of matching ontologies was retrieved, for each of the keywords. The results were filtered based on language (i.e., only allowing RDF/OWL). Next, all broken links were filtered out, e.g., where the ontology was no longer accessible at that URI. The resulting set consisted of 845 ontologies, which were saved locally for repeatability reasons[24] (no additional sampling was performed).

*2) Ontology Drafts:* The ontology drafts result from student assignments[2] to design an ontology within the theater domain, based on a fixed set of requirements, i.e., competency questions (CQs) [26]. They had not previously been introduced to ODPs (assured by self-assessment) but had some training (one full day) on OWL and ontology engineering. The task was designed so that it would be possible to solve some of the design problems using Content ODPs, in order to expose the students to those problems before introducing Content ODPs later in the course. The students were given 3 hours to solve the exercise, and they all used the same tool. The resulting set consists of 15 ontologies (of between 9-20 classes, and 12-30 properties).

### *C. Accuracy of Implicit Content ODP Detection*

To evaluate the Content ODP detection implementation we have applied it to the two sets of ontologies, together with a set of 76 content ODPs (the complete set of Content ODPs at that time available from the ODP Portal).

*1) Online Ontologies:* The accuracy evaluation of the indirect ODP detection within online ontologies was for practical reasons performed on a small sample of ontologies, randomly selected from the data set. The sample contained 40 ontologies, where 33 of them had at least one match to any of the 76 Content ODPs (threshold of class matches set to 50%), summing up to a total of 200 pattern detections. The ontologies were then reviewed by a human evaluator, to assess the matches. The human evaluator classified each proposed detection (i.e., each pattern-ontology pair) either as (a) 'I agree that there is a match, the suggestion is correct', (b) 'I do not agree that there is a match, the suggestion is incorrect', or (c) 'I cannot decide based on the available information'. The evaluator classified 62% of the suggestions into category (a), i.e., correct matches, 31% into (b), i.e., incorrect, and 7% into (c). Counting (a) as the correctly suggested patterns, **62%** corresponds to the level of **precision** of the detection approach.

While 62% precision may seem low, it is comparable to other complex search mechanisms operating on online data that are currently widely appreciated. For instance, consider online search engines, where the precision on complex queries has been assessed in [27]. The average precision of the search engines in this study ranged from 51.25% for Google, down to 32.5% for Ask.com, for complex queries. Although this study has a completely different aim, it shows that in fields such as online information retrieval, a precision as low as 51.25% is considered acceptable.

*2) Ontology Drafts:* The task given to the students had a clear set of requirements (CQs) and was constructed with a set of Content ODPs in mind, i.e., 6 of the Content ODPs available in the portal, hence, also the recall of the approach could be assessed. To obtain the 'gold standard' on which to perform the recall calculation, we additionally analyzed the intents and requirements of all the 76 Content ODPs and recognized 13 additional Content ODPs where the requirements match the CQs. Table I shows the resulting set of 19 Content ODPs [3]. Most of these were compositions or generalizations of the smaller set, while a few also represented alternative modelling choices applicable in this context, e.g., to view a music album as a collection of tracks, or tracks as being proper parts of the album.

Table I
CONTENT ODPS APPROPRIATE FOR USE IN SOLVING THE TASK.

| | **Content ODP Name** | | **Content ODP Name** |
|---|---|---|---|
| 1. | Agent Role | 11. | Person |
| 2. | Collection/Collection Entity | 12. | Place/Location |
| 3. | Componency | 13. | Region |
| 4. | Co-participation | 14. | Situation |
| 5. | Information Realization | 15. | Time-indexed Participation |
| 6. | N-ary Participation | 16. | Time-indexed Part Of |
| 7. | Object Role | 17. | Time-indexed Person Role |
| 8. | Participant Role | 18. | Time-indexed Situation |
| 9. | Participation | 19. | Time Interval |
| 10. | Part Of | | |

The results of the indirect matching can be seen in Table II. The table shows the average precision and recall over the set of 15 ontologies (threshold for class matches again at

[2]Assignment details at [25]

[3]Collection and Collection Entity are represented by the same OWL-building block although having separate pages in the ODP portal, just as Place and Location. Since our method works on the OWL building blocks, they are here treated as the same pattern.

50%). The reader should however note that while precision is highly relevant, recall is not an entirely relevant measure from a user perspective since some patterns are overlapping or simply specializations of others. This means that even with a recall less than 100% the set of proposed ODPs could cover the complete task.

Table II
AVERAGE PRECISION AND RECALL OVER THE 15 ONTOLOGIES.

| Method | Avg. Precision | Avg. Recall |
|---|---|---|
| Direct matching only | 14.4% | 1.4% |
| Direct+Indirect matching | 66.4% | 38.9% |

As a comparison we note that using only direct matching, on average 1 pattern was proposed for each ontology, while using the indirect matching the system proposed on average 11 patterns for each ontology, which is considered a reasonable number to be assessed by the ontology engineer (compared to the catalogue of 76, hence, 86% of the patterns were filtered out). An interesting problem to consider is how the method would perform on larger ontologies, however, it is worth noting that many patterns are applicable in several places within an ontology (e.g., if 'partOf' is applicable, it will only be proposed once although applicable throughout the ontology). This indicates that the number of proposed patterns will not explode when the ontology size increases.

Comparing to the XD Tools search functionality, when entering the ontology requirements, i.e., the CQs that were the basis of the draft ontologies, into the search interface (standard keyword indexing) of XD Tools only reaches a precision of 36% and recall of 21% for the first 11 results (i.e., the average number of patterns suggested by our pattern detection method). When considering the first 20 results we also note that the precision drops to 20% while the recall remains on 21%, indicating that we do not get any more useful results even if we check the next ten results, i.e., some patterns are very hard to retrieve with standard keyword-search.

### *D. Content ODPs Detected in Online Ontologies*

The study of online ontologies additionally aimed to assess Content ODP usage. In Table III we present the count of ontologies where the 20 most frequent patterns were detected using our method as described above (class match threshold again set to 50%). The dataset consisted of 682 ontologies previously collected[4].

When analyzing these results we note that certain patterns are favored by the background knowledge used for the matching, e.g., the Constituency pattern contains only one concept, named 'entity', which appears at the top level of WordNet. Other patterns are instead never matched, due to the simple property matching applied, e.g., the Part Of pattern, which contains no classes at all but only two properties. This leads us to conclude that the numbers presented are probably not reliable as an absolute count, rather the result can be seen as an indication that the solutions proposed by patterns are in fact used in online ontologies. More precise matching needs to be applied in order to derive accurate statistics. However, through our experience in working with patterns we can confirm that the patterns in Table III are in fact a selection of the ones we, as ontology engineers, have used most frequently, although some very frequent ones could not be detected due to limitations in the heuristics.

[4]Unfortunately, 162 of the original 845 ontologies were not processable through the OWL API, due to syntactic errors or other problems.

Table III
THE 20 MOST FREQUENTLY DETECTED CONTENT ODPS AND THE NUMBER OF ONTOLOGIES WHERE THEY WERE DETECTED.

| Content ODP Name | # | Content ODP Name | # |
|---|---|---|---|
| Constituency | 666 | Topic | 68 |
| Participation | 204 | Classification | 67 |
| Componency | 148 | Description | 67 |
| Co-participation | 101 | Parameter | 67 |
| Types of Entities | 101 | Basic Plan Execution | 59 |
| Collection | 98 | Participant Role | 47 |
| Agent Role | 85 | Task Role | 44 |
| Region | 75 | Task Execution | 44 |
| Object Role | 73 | N-ary Participation | 37 |
| Communities | 70 | Situation | 34 |

## VI. CONCLUSIONS AND FUTURE WORK

We have presented a simple Content ODP detection method and described its proof-of-concept implementation and evaluation in two usage scenarios. Although the accuracy of the current implementation can certainly be improved, we believe that it is a valuable complement to traditional ontology search and retrieval, where keyword-based search in most cases returns few or no results when applied to the task of finding highly abstract Content ODPs. In addition, we have presented a small study on Content ODP support in real-world ontologie. The results show that many Content ODPs are widely used, although very few seem to be explicitly imported.

Future work includes to improve the heuristics used by the method, especially on the side of properties and axioms where the results at the moment are quite poor. Typical naming patterns for properties could be used in order to compare property names with different structure but a similar meaning. Additionally, the axiom matching could be extended to exploit the matching results already provided by the class and property matching, in order to achieve an indirect axiom matching method as well. OntoCase, as it is currently implemented, is not applicable to our usage scenarios, however as future work it would also be interesting to compare other possible relaxations of OntoCase, to improve the trade-off between scalability and accuracy. We are currently exploring the possibility to integrate the implementation as a selection service in the XD Tools plugin within the NeOn Toolkit, and in the future we will also consider the possibility of providing a detection service for online search, to bring the advantages of this approach into practice.

## Acknowledgments

Part of this work was performed with support from The Swedish Foundation for International Cooperation in Research and Higher Education, project DEON (grant #IG2008-2011).

## References

[1] [Online]. Available: http://xmlns.com/foaf/0.1/ Accessed: 08.10.2010

[2] E. Blomqvist, A. Gangemi, and V. Presutti, "Experiments on Pattern-Based Ontology Design," in *K-CAP 2009*. ACM, 2009, pp. 41–48.

[3] [Online]. Available: http://www.w3.org/2002/07/owl Accessed: 08.10.2010

[4] [Online]. Available: http://www.ontologydesignpatterns.org Accessed: 08.10.2010

[5] E. Blomqvist, "Semi-automatic Ontology Construction based on Patterns," Ph.D. dissertation, Linköping University, Department of Computer and Information Science at the Institute of Technology, 2009.

[6] A. Gangemi and V. Presutti, "Ontology Design Patterns," in *Handbook on Ontologies, 2nd Ed.*, ser. International Handbooks on Information Systems. Springer, 2009, pp. 221–243.

[7] O. Sváb-Zamazal and V. Svátek, "Analysing Ontological Structures through Name Pattern Tracking," in *Proceedings of EKAW 2008*, ser. Lecture Notes in Computer Science, A. Gangemi and J. Euzenat, Eds., vol. 5268. Springer, 2008.

[8] O. Sváb-Zamazal, F. Scharffe, and V. Svátek, "Preliminary Results of Logical Ontology Pattern Detection using SPARQL and Lexical Heuristics," in *Proceedings of WOP 2009, collocated with ISWC-2009*, vol. 516. Washington D.C., USA: CEUR Workshop Proceedings, October 2009.

[9] O. Noppens and T. Liebig, "Ontology Patterns and Beyond - Towards a Universal Pattern Language," in *Proceedings of WOP2009 collocated with ISWC2009*, vol. 516. CEUR-WS.org, November 2009.

[10] [Online]. Available: http://ontologydesignpatterns.org/cpont/codo/codolight.owl Accessed: 08.10.2010

[11] N. Nikitina, S. Rudolph, and S. Blohm, "Refining Ontologies by Pattern-Based Completion," in *Proceedings of WOP 2009, collocated with ISWC-2009*, vol. 516. Washington D.C., USA: CEUR Workshop Proceedings, October 2009.

[12] M. d'Aquin, C. Baldassarre, L. Gridinoc, M. Sabou, S. Angeletou, and E. Motta., "Watson: Supporting next generation semantic web applications." in *Proceedings of the WWW/Internet conference, Vila real, Spain, 2007*, 2007, pp. 363–371.

[13] G. Tummarello, E. Oren, and R. Delbru, "Sindice.com: Weaving the Open Linked Data," in *Proceedings of ISWC/ASWC2007, Busan, South Korea*, ser. LNCS, vol. 4825. Berlin, Heidelberg: Springer Verlag, November 2007, pp. 547–560.

[14] L. Ding, T. Finin, A. Joshi, R. Pan, R. S. Cost, Y. Peng, P. Reddivari, V. C. Doshi, and J. Sachs, "Swoogle A Semantic Web Search and Metadata Engine," in *Proc. 13th ACM Conf. on Information and Knowledge Management*, Nov. 2004.

[15] N. F. Noy, N. H. Shah, B. Dai, M. Dorf, N. Griffith, C. Jonquet, M. J. Montegut, D. L. Rubin, C. Youn, and M. A. Musen, "BioPortal: A Web Repository for Biomedical Ontologies and Data Resources," in *Poster and Demo Proceedings of ISWC 2008*, 2008.

[16] S. Peroni, E. Motta, and M. d'Aquin, "Identifying Key Concepts in an Ontology, through the Integration of Cognitive Principles with Statistical and Topological Measures," in *Proceedings of ASWC 2008*, ser. Lecture Notes in Computer Science, J. Domingue and C. Anutariya, Eds., vol. 5367. Springer, 2008, pp. 242–256.

[17] C. Anutariya, R. Ungrangsi, and V. Wuwongse, "SQORE: A framework for semantic query based ontology retrieval." in *Advances in Databases: Concepts, Systems and Applications*, ser. LNCS, vol. 4443. Springer, 2007, pp. 924–929.

[18] [Online]. Available: http://stlab.istc.cnr.it/stlab/XDTools Accessed: 08.10.2010

[19] V. Presutti, E. Daga, A. Gangemi, and E. Blomqvist, "eXtreme Design with Content Ontology Design Patterns," in *Proc. of WOP 2009, collocated with ISWC-2009*, vol. 516. Washington D.C., USA: CEUR Workshop Proceedings, October 2009.

[20] C. Fellbaum, Ed., *WordNet - An Electronic Lexical Database*. MIT Press, 1998.

[21] [Online]. Available: http://owlapi.sourceforge.net/ Accessed: 08.10.2010

[22] [Online]. Available: http://watson.kmi.open.ac.uk/WS_and_API.html Accessed: 08.10.2010

[23] [Online]. Available: http://lyle.smu.edu/∼tspell/jaws/index.html Accessed: 08.10.2010

[24] [Online]. Available: http://ontologydesignpatterns.org/experiments/ODPDetectionCorp2010.zip Accessed: 08.10.2010

[25] [Online]. Available: http://ontologydesignpatterns.org/wiki/Training:PhD_Course_on_Computational_Ontologies_%40_University_of_Bologna/Hands-on_(Day2):_Theater_productions Accessed: 08.10.2010

[26] M. Gruninger and M. S. Fox, "The role of competency questions in enterprise engineering," in *Proc. of the IFIP WG5.7 Workshop on Benchmarking - Theory and Practice*, 1994.

[27] M. L. Robinson and J. Wusteman, "Putting Google Scholar to the test : A preliminary study," *Program*, vol. 41, no. 1, pp. 71–80, 2007.