Effective NL Paraphrasing of Ontologies on the Semantic Web

Aditya Kalyanpur, Christian Halaschek-Wiener,
Vladimir Kolovski, James Hendler
{aditya, halasche, kolovski, hendler} @cs.umd.edu

Dept. of Computer Science,
University of Maryland,
College Park MD 20742

Abstract. In this paper, we present an algorithm that provides natural language (NL) *paraphrases* for OWL Ontologies (specified in RDF/XML) on the Semantic Web. Our goal is to ensure both, fluency (readability) of the output, and accuracy, in terms of meaning conveyed by its description logic formalism. The algorithm uses the Visitor Design Pattern to build a NL parse tree from the ontology, which interleaves textual phrases with ontological terms and relationships. Various heuristics – syntactic (using a POS tagger) and semantic (using a reasoner) are then used to generate appropriate NL sentences. We provide implementation details of the algorithm in the context of a semantic web ontology engineering toolkit, SWOOP, currently being developed in our research lab. Lastly, we test our approach using real world ontologies and compare it with current tools that provide a similar functionality

1 Introduction

The Semantic Web [2] is an extension of the current World Wide Web. The hypertext pages that present information to humans remain, but a new layer of machine understandable data is added to allow computers to participate on the Web in new ways. Using standardized ontology representation languages such as OWL [3], semantic web data can precisely describe the knowledge content underlying HTML pages, specify the implicit information contained in media like images and videos, or be a publicly accessible and usable representation of an otherwise inaccessible database.
OWL is built on top of the Resource Description Framework (RDF [6]), which is itself built upon the XML syntax. RDF (including RDFS - the RDF schema language) and OWL provide provide the capability of creating Classes, Properties, and Instances. Classes (or concepts) are general categories that can be arranged in hierarchies. Each class defines a group of individuals that belong together because they share some properties. Instances (or individuals) are specific objects, and classes are used to define what type an object has. Properties (or relationships) are attributes of instances and are used to either specify data values or link

to other instances. **Figure 1** depicts a sample OWL class (describing the concept Medoc taken from the Wine Ontology [Wine]) serialized in RDF/XML.

```
<owl:Class rdf:ID="Medoc">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Bordeaux">
        </owl:Class>
        <owl:Restriction>
          <owl:onProperty rdf:resource="#locatedIn" />
          <owl:hasValue rdf:resource="#MedocRegion" />
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasColor" />
      <owl:hasValue rdf:resource="#Red" />
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasSugar" />
      <owl:hasValue rdf:resource="#Dry" />
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Fig. 1. OWL Class 'Medoc' in the Wine Ontology serialized in RDF/XML

The above example clearly demonstrates that the verbose syntactic structure of RDF/XML is extremely difficult to read for the novice user. Moreover, complex concepts and relationships in an ontology (such as the one shown above) are constructed using a combination of description logic (DL) based operators that exacerbate the problem. Thus, readability can be greatly enhanced if a NL representation of the DL-based term definitions is provided. The class presented in **Figure 1** could be translated to the following NL representation: *'Medoc is a sweet, red color wine located in the Medoc region.'*

Additionally, a need to provide an easily readable explanation of terms in the ontology arises from the fact that the intended purpose of ontologies is for information sharing, which could involve external parties that have little or no background knowledge of the ontology domain. In such cases, it becomes the responsibility of the domain experts creating the ontology to provide textual documentation for the terms within. However, in the absence of such information, understanding the explicit meaning behind

the terms can be difficult. Again here, a NL representation of the ontology can be immensely useful.
Given this motivation, we present our work on developing such a system, which generates NL representations of OWL class definitions. Properties and individuals in some cases have more complex NL representations and are not handled in the present version of our system. Our goal is two-fold:

1. Ensuring fluency (readability) of the final output sentence(s)
2. Maintaining logical correctness of the concept definition in the output, in terms of meaning conveyed by the concept's DL formalism

2 Related Work: Current State of the Art

As discussed earlier, our aim is to devise an algorithm for generating NL explanations of a conceptual term defined in OWL. We intend to build upon and extend the results of previous efforts in this area, which are briefly discussed here.
An excellent example of the instructional use of NL paraphrases for understanding OWL Concepts is described in *'OWL Pizzas: Practical Experience of Teaching OWL DL: Common Errors and Common Patterns'* [8]. Indeed, we take inspiration from such work and attempt to automatically generate NL paraphrases such as the ones illustrated in their paper. Also note that at the time of writing this paper, the authors know of no implementation that has achieved this. The *Class Description Display* plugin (http://www.co-ode.org/downloads/cdc/) mentioned in their paper works with the Protege OWL plugin and provides simple quasi-NL descriptions that resemble OWL Abstract Syntax (http://www.w3.org/TR/owl-semantics/). We note that, in general, the OWL AS while a step above RDF/XML in terms of readability is still difficult to understand as demonstrated by examples in this paper.
In [1], a technique for mapping elementary semantic expressions to corresponding NL representations is presented. In their approach, the authors apply multiple sequence alignment techniques to a semantic expression along with corresponding alternative verbalizations. This then produces a more expressive and accurate single dictionary entry. Our approach differs in that we do not assume the verbalizations. We are actually generating the verbalizations algorithmically from the semantic expression itself.
In [4], a subset of English is introduced called Attempto Controlled English. ACE is translated unambiguously into first-order logic and thus can be used as a formal notation. Even though ACE seems to be a NL, it is actually a formal language with the semantics of First Order Logic (FOL). In comparison, our tool converts OWL classes, which are based on a decidable subset of FOL called Description Logics, into a NL description.
[9] describes an XML-based NL generation for RDF and DAML+OIL, which are two representation languages that are less expressive than OWL. In this work, a pipeline of XSLT transformations implements the sequence of processing stages in the orthodox pipeline architecture for

NL generation. The generator uses predefined XSLT text plan templates for specific ontologies, following a domain-specific approach of shallow generation. However, it remains to be seen whether this approach works efficiently for more complex OWL ontologies.

3 Design and Implementation

For clarity's sake, we describe our NL algorithm in the context of an existing semantic web ontology engineering toolkit, SWOOP. The design and implementation details are described in the following subsections.

3.1 Testbed Application: SWOOP

We decided to build our tool to generate NL representation of OWL concepts over an existing semantic web application. For this purpose, we chose SWOOP, a web ontology browser and editor designed within our research group, MINDSWAP. SWOOP has a modular and extensible architecture, which (among other things) supports visualization or renderer plugins seamlessly.
Figure 2 illustrates the architecture of SWOOP, along with the additional plugin implemented within the context of this work. As shown in the figure, various kinds of renderer plugins can be integrated into SWOOP, each of which work at a different granularity level. In our case, we designed an Entity Renderer plugin since our task involved rendering NL information at the level of a single OWL class or entity. The following sections will address the technical issues regarding the Natural Language Entity Renderer. Additional details regarding SWOOP can be found at its project homepage [1]

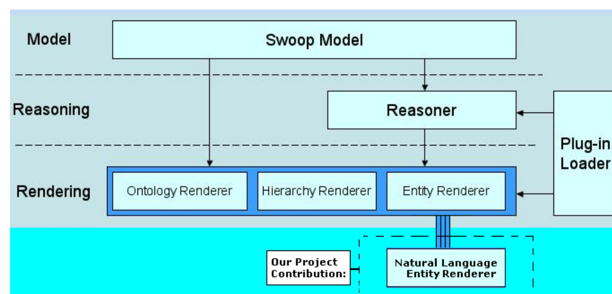

Fig. 2. Plugging our Natural Language Entity Renderer in SWOOP

[1] SWOOP Home: http://www.mindswap.org/2004/SWOOP

3.2 Visitor Design Pattern

The visitor pattern is a well-known software design pattern that involves the separation of the algorithm from the associated object structure [2]. The basic idea is that you have a set of element classes that form an object structure. Each of these element classes has an accept method that takes a Visitor object as an argument. The Visitor is an interface that has a different visit method for each element class. The accept method of an element class calls back the visit method for its class. Separate concrete Visitor classes can then be written that perform some particular operations

For the purpose of this work we choose to implement a visitor pattern as it seems an ideal choice to render complex, nested OWL class expressions. Thus, the different logical constructs of OWL can be represented by individual visitor nodes in the Visitor class, each of which accepts a visitor object created and passed from a separate Renderer class. Depending on the OWL expression to be rendered, the visitor 'visits' the corresponding node, does the necessary processing (rendering or otherwise), and is then passed from one node to the next, traversing the entire class expression (illustrated in **Figure 3**).

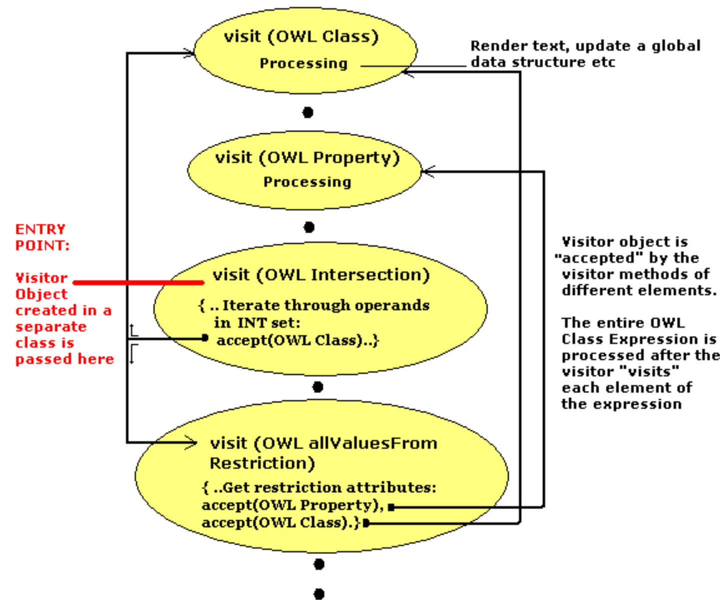

Fig. 3. Visitor Pattern used to process OWL Class Expressions

[2] Visitor Pattern discussion: http://home.earthlink.net/~huston2/dp/visitor.html

3.3 NL Parse Tree

We decided to use the visitor design pattern for our implementation, but instead of directly rendering NL information inside each visitor node (method), we chose to build a NL parse tree of the OWL class expression. Creating this tree gives us additional flexibility in our approach since we are free to alter it (post-process) in any way deemed necessary.
Take the example of the OWL Class `MaleStudentWith3Daughters` whose definition is shown in the table below in OWL Abstract Syntax [AS]:

```
Class (koala:MaleStudentWith3Daughters complete
intersectionOf(
restriction(koala:hasGender value (koala:male))
restriction(koala:hasChildren allValuesFrom(koala:Female))
restriction(koala:hasChildren cardinality(3))
koala:Student
)
))
```

In this case, the NL parse tree generated by SWOOP is shown in **Figure 4**:

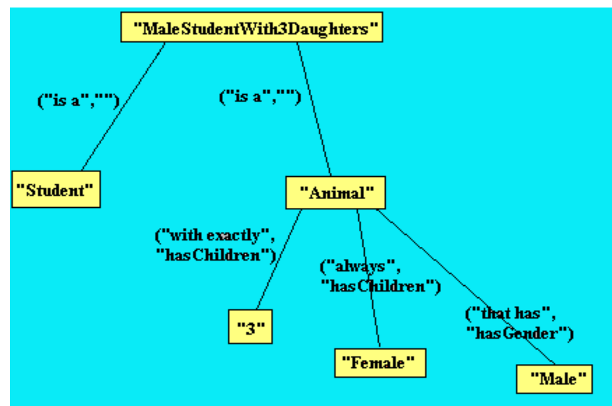

Fig. 4. NL Parse Tree for OWL Class `MaleStudentWith3Daughters`

Each node in the tree represents an OWL entity: class, individual or data-value and contains just the keyword(s) for the specific entity name or value. In addition, the node contains a (hash) map that provides all its link-target pairs in the tree.
Each link in the graph represents a relationship between two OWL entity nodes and is specified by two parameters:

1. *Link Type* i.e. type of OWL restriction (relation) between the two entity nodes such as: *subClassOf, someValuesFrom, allValuesFrom, hasValue, cardinality* etc which are all predefined OWL constructs. Note that the keywords for these link types

can be fine tuned depending on the desired output. The default keywords are shown in Table 1.

2. *Link Name* i.e. name of the connecting OWL Property if any.

Table 1. Description of Links in the NL Parse Tree

Link Type	OWL Restriction	Link Value (can be fine-tuned)
Link_Equivalent	`owl:equivalentTo`	'is a'
Link_Subclass	`rdfs:subClassOf`	'is a'
Link_Complement	`owl:complementOf`	'is not a'
Link_Intersection	`owl:intersectionOf`	'and'
Link_Union	`owl:unionOf`	'either..or'
Link_OneOf	`owl:oneOf`	'either..or'
Link_All	`owl:allValuesFrom`	'that (if..) always'
Link_Some	`owl:someValuesFrom`	'which (among other things)..some'
Link_Value	`owl:hasValue`	'that has/is'
Link_Max	`owl:maxCardinality`	'with at most'
Link_Min	`owl:minCardinality`	'with at least'
Link_Card	`owl:cardinality`	'with exactly'
Link_Between	`max&minCard. co-occur`	'with between'

After generating the parse tree, our task of displaying a NL output of the OWL class expression reduces to walking this tree and printing out the values of nodes and links in an orderly fashion. Additionally, to improve results (i.e. make the NL output more readable), we use a combination of filters and rules to sort the links, aggregate relevant information and combine data about different restrictions on the same property. We summarize these in Table 2: In the case of the Class `MaleStudentWith3Daughters`, the following operations are performed:

- **Rule 1:** The domain of the property `hasChildren` i.e the class `Animal` is used as the subject of the verb phrase (VP) 'always hasChildren Female' etc (Note: domain can be inferred using a reasoner)
- **Rule 2:** The link-target pair *is-a-Student* is given higher priority than the pair *is-a-Animal* (and the subsequent subtree below 'Animal') so that the final renderer output has the most specific definition ('Student') before printing constraints on the generic definition ('Animal that has..')
- **Rule 3:** The nested `Animal` class has two links (restrictions) on the same property `hasChildren` so we aggregate this information in the NL output (using ',' as a separator) instead of repeating the property name each time ('..hasChildren...hasChildren..')
- **Rule 4:** We combine the property restriction 'exactly 3 hasChildren' with the restriction 'always hasChildren Female' to form

Table 2. Syntactic and Semantic Post-Processing Rules

Rule Name	**Description**
1. Use property attributes	The *domain* of the property is used as the subject of the predicate (or VP) to give the sentence a clearer meaning. Property *range* is used in specific cases, see examples in section 4.
2. Sort is-a links	More specific *is-a* relationships are given higher priority than restrictions on generic *is-a* relationships to enhance readability
3. Combine property object values	Links having the same property name are aggregated to prevent duplication of property name in output
4. Combine property restriction types	Different restriction types on the same property are combined using appropriate phrases to enhance readability
5. Separate clauses	Complex sentences with multiple nested clauses are broken into appropriate simpler sentences

the expression 'exactly 3 hasChildren, each of which has Female'. Here, the phrase 'each of which' has replaced the default keyword 'always' to enhance readability

- **Rule 5:** Since the class definition contains an additional *hasValue* restriction on property `hasGender` (set to `Male`), combining this restriction clause with the earlier sentence fragment (restriction from Rule 4) hinders readability. Hence we split the output NL paraphrase into two sentences using the keyword 'Also..' (a special-purpose flag in the program is used to keep check of sentence clauses/complexity).

Hence, the NL output generated from the parse tree after application of the above rule set is:
'`MaleStudentWith3Daughters` *is a Student and is an Animal with exactly 3 has children each of which has Female. Also, it has gender Male.'* [3]

3.4 Improving Fluency using a POS Tagger

Typically, a part-of-speech (POS) tagger accepts a sentence string as an input and returns a POS tag for each word in the sentence. Using a POS tagger can greatly improve the fluency of the NL paraphrases by reordering words/phrases depending on their context-aware tags. In our SWOOP plugin, we use the online statistical POS tagger (TnT - http://www.coli.uni-sb.de/~thorsten/tnt/) which is based on the Penn Treebank Tagset.

[3] Note: We also use simple syntactic rules such as removal of duplicate words/phrases, usage of correct form of a determiner etc. that have not been mentioned explicitly

Taking the earlier example of `MaleStudentWith3Daughters`, obtaining the POS tags for words in the sentence, we make the following changes to the earlier output:

1. *Re-phrase cardinality clauses*: by positioning the phrase (exactly/at least/at most) just before the noun phrase. In this case, 'with exactly 3 has children' becomes 'which has exactly 3 children'
2. *Re-order predicate-value pairs*: by splitting the predicate into component words and using its tags to determine positioning. In this case, 'has gender Male' becomes 'has Male gender'

Hence, the output NL paraphrase now becomes: *is a Student and is an Animal which has exactly 3 children each of which has Female. Also, it has Male gender.'*
Note: Generic rule not applicable for the current case– *Prefix 'has/is' appropriately*, if the predicate starts with a VBZ (verb 3-person singular present tense), neither 'has/is' is prefixed to it, otherwise 'is' is prefixed for a VBN/VBG (verb past participle/gerund) and 'has' is prefixed for the remaining cases by default. Also note that rules such as this, which insert words/phrases based on POS tags of ontological terms can be easily added to the system to improve readability of the final output.

4 Comparison

In order to compare our methodology, we tested our NL generation algorithm on five disparate real-world ontologies that met two essential criteria: OWL Classes defined in them made use of complex DL operators (rather than forming a simple taxonomy), and the ontologies were well-written following the standard class/property naming conventions i.e. meaningful names for terms with appropriate capitalization at word boundaries. The ontologies include Wine/Food, Koala, W3C-Photo, Pizza and MadCow. A few select class outputs that characterize the algorithm features is shown in Table 4
As illustrated in the output table, our algorithm provides promising results across the entire cross-section of ontologies tested. The NL paraphrases generated are fairly fluent (barring a few minor grammatical mistakes) and capture the DL meaning of the OWL Class definition consistently.
We also compare our performance against current state-of-the-art tools that provide similar functionality. Comparison is based on two metrics - output NL fluency i.e. sentence formation, grammar etc, and output DL correctness i.e. accurate representation of logical meaning in the output. As shown in Table 3, our SWOOP plugin produces the best overall result.

Table 3. Tool Comparison: NL descriptions of OWL Classes

Tool Name	**NL fluency**	**Emphasizing DL meaning**
Protege OWL CO-ODE plugin	Quasi-NL (fragmented phrases) with poor readability	Uses single keywords that correspond to OWL terms
OntoXpl [5]	Quasi-NL with good readability	Uses elaborate phrases
XML-based generation using XSLT [9]	NL with excellent readability	Does not explicate complex conceptual definitions
SWOOP plugin	NL with good readability	Uses long descriptive paraphrases

5 Open Issues

As demonstrated in the comparison earlier, our NL generation system for OWL Class expressions works reasonably well on our test case ontologies. However, there still remain some fundamental open issues related to the complexity of the problem:

1. If the OWL class expressions are deeply nested (depth > 3), the NL sentences become difficult to read. Consider the following class expression:
```
Class (bus+driver partial
subClassOf(person)
unionOf(
complementOf(
restriction(likes someValuesFrom(
intersectionOf(restriction(age someValuesFrom(oneOf(young))))
person))
)
)
restriction(reads allValuesFrom(broadsheet))
))
```
We generate: *'bus+driver is either a person that (if reads) always reads broadsheet or is not a person which (among other things) likes some person that has young age'*
Our solution, consistent with our approach, is moderately good but readability of the sentence is hampered given the complex nature of the class definition. In such cases, finding generic solutions is very hard and we need to investigate such cases further.
2. We rely heavily on the standard naming conventions for class and property names as noted in our comparison. Obviously, our solution gives inaccurate results if such conventions are not followed.
3. OWL also has a provision for defining General Concept Inclusion (GCI) axioms in the ontology, of the form:
Expression1 (operator) Expression2 e.g. intersection(A, B) (subClassOf) union(C,D). Integrating the information given by the GCI's with the associated OWL classes in NL is a difficult research problem and needs to be studied separately

Table 4. Sample Outputs on Test-Case Ontologies

Ontology: Pizza – http://www.co-ode.org/ontologies/pizza/pizza_20041007.owl

`Class (MediumPizza partial`
`Pizza`
`restriction(hasTopping maxCardinality(5))`
`restriction(hasTopping minCardinality(3))`
`)`
`)`
Output: MediumPizza is a Pizza which has between 3 - 5 topping

Ontology: W3C-Photo – http://www.mindswap.org/2004/www04photo.owl

`Class (KeynoteSpeaker partial`
`Presenter`
`restriction(presenterOf someValuesFrom( KeynoteTalk))`
`)`
`)`
Output: KeynoteSpeaker is a Presenter which (among other things) is presenter of some Keynote Talk

Ontology: Wine – http://www.w3.org/2001/sw/WebOnt/guide-src/wine.owl

`Class (Pauillac partial`
`restriction(hasFlavor value (Strong))`
`restriction(madeFromGrape maxCardinality(1))`
`restriction(madeFromGrape value (CabernetSauvignonGrape))`
`restriction(hasBody value (Full))`
`)`
`)`
Output: Pauillac is a Wine which is made from atmost 1 grape which is Cabernet Sauvignon Grape
Also it is a Wine that has Full body and that has Strong flavor

Ontology: Koala – http://protege.stanford.edu/plugins/owl/owl-library/koala.owl

`Class (GraduateStudent partial`
`Student`
`restriction(hasDegree someValuesFrom(oneOf(BA BS))))`
`)`
`)`
Output: GraduateStudent is a Student and is a Person which either has B A or B S degree

Ontology: Food – http://www.w3.org/2001/sw/WebOnt/guide-src/food.owl

`Class (ShellfishCourse partial`
`restriction(hasDrink allValuesFrom(restriction(hasBody value (Full))))`
`restriction(hasDrink allValuesFrom( restriction(hasFlavor allValuesFrom(`
`oneOf(Strong Moderate))))))`
`)`
`)`
Output: ShellfishCourse is a Meal Course that (if has drink) always has drink Potable Liquid that has Full body and which either has Moderate or Strong flavor
Note: The phrase 'Potable Liquid' in the above output is obtained from the *range* of the property `hasDrink` replacing the default keyword 'Thing'.

6 Conclusion and Future Work

We have presented an algorithm that generates concise, accurate NL paraphrases for OWL Concepts based on a variety of NLP techniques and implemented it in an ontology engineering toolkit, SWOOP. As our comparisons show, our results are quite promising on a diverse cross-section of OWL ontologies. Furthermore, our approach clearly outperforms current state of the art tools offering similar functionality. There are noticeable limitations in our methodology such as the reliance on standard naming conventions etc, but the overall result meets our primary goal of language fluency and logical accuracy. We plan to conduct formal user studies to fully evaluate the contribution of our work.

As a next step, we intend using a lexicon such as Wordnet [7] to tag ontological terms to formulate NL sentences better. Additionally, we are keen on explaining the DL formalism in more detail as done in [8]. Finally, we would like to provide NL paraphrases for OWL properties and individuals. the latter being complicated due to abundant triples (maybe nested or reified) and linked-individual expressions.

References

1. R. Barzilay and L. Lee. Bootstrapping lexical choice via multiple-sequence alignment. *Proceedings of EMNLP*, 2002.
2. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, May 2001.
3. M. Dean and G. Schreiber. OWL Web Ontology Language Reference W3C Recommendation. http://www.w3.org/tr/owl-ref/. Feb. 2004.
4. S. U. Fuchs, N. E. and S. Torge. Controlled natural language can replace first-order logic. *14th IEEE International Conference on Automated Software Engineering, Cocoa Beach, Florida*, Oct.
5. V. Haarslev, Y. Lu, and N. Shiri. Ontoxpl: Exploration of owl ontologies. *In Description Logics (DL)*, 2004.
6. O. Lassila and R. Swick. Resource description framework (rdf) model and syntax specification. w3c recommendation, www consortium (cambridge, ma). Feb. 1999.
7. G. Miller. Wordnet: A lexical database for the english language. *Cognitive Science Laboratory, Princeton University.*
8. A. Rector, N. Drummond, M. Horridge, J. Rogers, H. Knublauch, R. Stevens, H. Wang, and C. Wroe. Owl pizzas: Common errors & common patterns from practical experience of teaching owl-dl. *In European Knowledge Acquisition Workshop (EKAW)*, 2004.
9. G. Wilcock. Talking owls: Towards an ontology verbalizer. *Proceedings of the 2nd International Semantic Web Conference (ISWC)*, Oct.