Effective NL Paraphrasing of Ontologies on the Semantic Web

Daniel Hewlett[1], Aditya Kalyanpur[2], Vladimir Kolovski[2],
Christian Halaschek-Wiener[2]
[1]hewlett@umd.edu [2]{aditya, kolovski, halasche} @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 on the Semantic Web. Our goal is to ensure both fluency (readability) and accuracy of the output, in terms of preserving the meaning conveyed by its description logic formalism. The approach described is a generic domain-independent one, and is completely automated. We describe details about the algorithm and follow it up with a subjective evaluation (pilot study) of our approach using real world ontologies comparing it with current tools that provide similar functionality.

1 Motivation and Goals

With the advent of OWL, and its subset OWL-DL, semantic web content is backed by a precisely-defined Description Logic (DL). This property means that the meaning of semantic web content will always be clear and potentially useful to an intelligent agent, or reasoner-equipped software application. However, concept definitions (OWL Classes) are specified in the language of logic, requiring humans to understand this logical language in order to decipher the meaning of concepts. For end users of semantic web enabled applications, this may pose a usability problem in many important circumstances, effectively creating a barrier for entry into the semantic web.

To remove this barrier, we have designed and implemented a procedure for generating near-Natural Language (NL) paraphrases (in English) of OWL concept definitions that preserve the semantics of the DL description. These paraphrases can be presented to the user either in addition to or instead of the logical class definitions. By presenting the class names and English definitions, designers can keep users in an environment of entirely natural language-based interaction, while not losing the semantic rigor and precision that OWL provides.

For a procedure such as ours to be widely useful, it has to be not only robust but also domain-independent, able to work with a large number of the concepts and ontologies available. A domain-independent solution is desirable because it can immediately make use of the numerous OWL

ontologies that already exist, modeling everything from clinical and environmental information (e.g., NCI and JPL) to personal interests and relationships (e.g., FOAF). A domain-specific procedure, however, will need to be re-tuned to each domain or ontology, greatly increasing the amount of work on the part of ontology designers. Also, distributing and integrating the extra domain-specific information required by such programs would add a layer of data not included in standard OWL.
Our approach fulfills all of these criteria. First, because we use only the names of properties and classes, which are already present in the OWL ontology, we do not require extensions to the ontology or additional information sources. This makes our approach valid for any ontology where the classes and properties are named appropriately. Also, the most sophisticated NL processing tool our approach utilizes is a part-of-speech (POS) tagger, which is a fast and simple application. Slightly better results could possibly be generated using a richer set of NL abilities, such as conjugation of verb forms, grammaticality judgments, parsing, etc., but such an application would be much less efficient.

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 [5]. We take inspiration from this 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 very complex for novice end users (for an small example of this, see Figure 1).
In [2], 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 [3], 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.
[6] 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

Our approach takes advantage of the standard naming conventions associated with OWL classes and properties, and uses the semantics of English constructions to convey clearly the semantic constraints imposed by OWL concept definitions. The first step in the design is to generate a tree corresponding to the relations between the class and other entities. While we decided to use the visitor design pattern for our implementation, we chose to build this parse tree rather than directly render NL information inside each visitor node (method). Creating this tree gives us additional flexibility in our approach since we are free to alter it (post-process) in any way deemed necessary (for an example, see Figure 1).

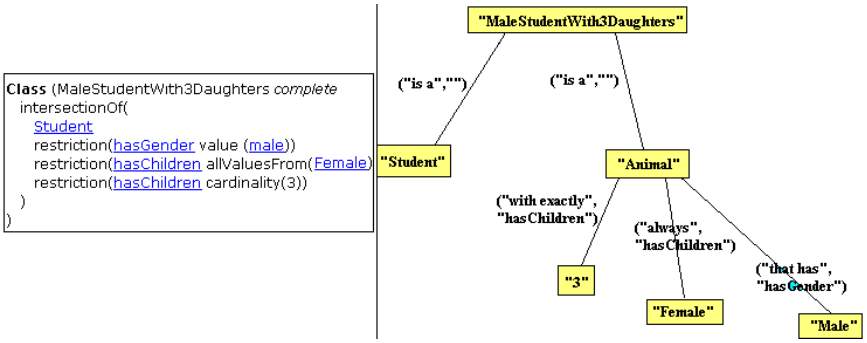

Fig. 1. NL Parse Tree for OWL Class `MaleStudentWith3Daughters`

3.1 Properties

Properties relate one OWL individual to another. A survey of property names from several major ontologies reveals that, while properties could theoretically be named with any arbitrary words, their names can almost always be parsed into one of a small number of simple phrase structures. The table below lists these phrase structure categories, together with

representative property names, and the expanded NL forms generated from each propery name.

1. (has) NP
   - Examples: email, hasColor
   - Expansions: X has a color Y
   - Alternate (if Y is an AdjP): X has Y color
2. V
   - Example: knows
   - Expansion: X knows Y
3. (is) NP P
   - Examples: brotherOf, isBrotherOf
   - Expansion: X is a brother of Y
4. (is) VP P
   - Examples: producedBy, isMadeFrom
   - Expansions: X is produced by Y, X is made from Y
5. VP NP
   - Example: producesWine
   - Expansion: X produces a wine Y
   - Alternate (if Y is an AdjP): X produces a Y wine
6. is NP
   - Example: isMetal
   - Expansion: X is a metal
   - Alternate (boolean value is false): X is not a metal
7. (is) AdjP
   - Example: isHardWorking
   - Expansion: X is hard working
   - Alternate (boolean value is false): X is not hard working

Using a part-of-speech (POS) tagger, the program can automatically detect which of the above categories the property belongs to, and generate the corresponding grammatical NL paraphrase.
Unfortunately, ambiguities may arise when the set of tags is returned, because lexical items in English do not always have a unique categorization. For instance, 'stop' can be both a verb and a noun, and there may not be enough contextual information in the property name to determine its part of speech. While not all ambiguities can be resolved, there are some general heuristics that can be used. The most common ambiguous tag, for example, arose from words such as 'likes', which is both the 3rd person present tense of the verb 'to like' ('John likes Mary') and also the plural of the noun 'like' (' all of his likes and dislikes'). In these cases, we resolve the ambiguity by giving priority to the verbal form, since none of the ontologies studied ever used a plural noun as a property name.

3.2 Classes

An OWL class represents a set or collection of individuals. OWL classes may have restrictions on the relations their members may participate in, and what the targets of these relations may be. Also, OWL classes stand in relations to one another, such as subsumption, equivalency, and disjointness. To describe an OWL class, our approach presents the necessary

and sufficient conditions to be a member of the class. For example, for the class $Student \sqsubseteq \exists enrolledIn.Course$, an individual must stand in the $enrolledIn$ relation to some member of the class $Course$. We paraphrase this condition as follows:

`A Student is enrolled in a Course.`

When a domain is specified for a property, we can use this to provide extra information in the paraphrase. The following is the same example as above, but with the domain of enrolledIn specified as $Person$. Since members of the class $Student$ must now also belong to the class $Person$, the following is a valid NL paraphrase:

`A Student is a Person that is enrolled in a Course.`

This method works exactly the same for the hasValue construct. There is no ambiguity, however, because the objects of the hasValue restriction are always individuals, so they will not be prefixed with 'a'.

To capture the semantics of an `owl:allValues` restriction, we use a conditional construction. AllValues does not imply the existence of any actual relationships, but rather puts conditions on any relationships that might exist. Here we use POS recognition to generate a grammatically acceptable conditional sentence. If the property name does not contain an NP, the conditional sentence is generated as follows:

`If a Giraffe eats something, then that thing is a Leaf.`

In this sentence, the indefinite NP's 'something' and 'thing' are used to represent the target of the relation 'eats', since no information is given in the property name about the target. However, many properties contain an NP target in their names, which can be used to replace the indefinite NP's:

`If a BlandFishCourse has a drink, then that drink has Delicate flavor.`

Note that the approach is relatively straightforward for constructs such as `owl:intersectionOf` (using *and*), `owl:unionOf` or `owl:oneOf` (using *either...or*), and for cardinality restrictions, using phrases such as *at least* m ($\geq m$), *at most* n ($\leq n$) and *between* m *and* n ($\geq m, \leq n$)

3.3 Procedure

After generating the parse tree, there are several steps in generating the NL output. The first is a pre-processing step where the tree is modified to eliminate nodes containing Thing. This is necessary because the original concept definitions contain many implicit references to Thing, such as in nested restrictions. In most cases, these can be replaced either by merging with a sister node containing a named class, or by promoting a class that the Thing node is related to by an 'is-a' relation. Both of these operations result in a tree that is logically equivalent to the original tree, but contains less redundancy, and fewer nodes in general, resulting in a simpler NL output.

Next, a recursive visitor begins at the root node (the concept to be defined), processes it, and then visits each of its successors. The processing of a node begins by removing all immediate all-values relations. These relations (restrictions in the logic) are handled separately because they

require a special sentence structure, meaning they always require a separate sentence or bullet. Next, the links out of the node are examined to see if any of them can be combined and handled as a single larger link. One example of this is min-cardinality and max-cardinality, which are combined using the 'between' keyword. After this step, the remaining relations can (generally) be handled within the same sentence, so the visitor can traverse them without extra processing. It will print the name of the current node, and then follow all the relations from the current node to its successors. The property names are expanded when the tree is constructed, so they can be treated opaquely for the remainder of the procedure.

3.4 NL Fluency and Readability

In general, we aimed to generate full sentences of English whenever possible. However, after evaluating our initial prototype designs it was realized that rendering complex concepts entirely in NL results in very complex, difficult to understand sentences. This complexity arises from the combination of multiple types of restrictions. For example, the sentence below (part of the definition of 'Anjou' in the wine ontology) is a grammatical and understandable sentence of English:

```
An Anjou is a Wine that has Delicate flavor, Off Dry sugar, Rose
color and Light body.
```

We found that, in some cases, using a bulleted, nested list format for such complex sets of conditions was much clearer. For example, part of the definition of 'Beaujolais' from the wine ontology is given below:

```
A Beaujolais is a Wine that:
 -- is made from at most 1 grape, which is Gamay Grape
 -- has Delicate flavor
 -- has Dry sugar
 -- has Red color
 -- has Light body
```

More complicated generation methods exist that would likely improve the quality of text output, however many of these require knowledge of the ontology domain (or some other knowledge, such as commonsense world knowledge) that we deliberately did not build into the system, to make it completely generic and efficient. The way we have constructed our bulleting system, for example, does not require the use of pronouns, so the capability to determine which pronouns to use (he/she/it/they/etc.) is not needed. This capability could be designed, but it would almost certainly rely on some domain knowledge.

4 User Evaluation

We performed a pilot user survey to evaluate the output of our program. The goal was to gauge how subjects considered the output of our program relative to that of Protege[1] (3.1 with OWL plugin build 284)

[1] http://protege.stanford.edu

and OntoExpl[2]. We chose the latter two because they have generic NL generation systems for OWL. Ten subjects were chosen in all, who were individuals that did not have a background in the semantic web, nor a familiarity with the description logic underlying OWL. The procedure was as follows:

We gathered a small set (5) of classes from ontologies publicly available on the web: Wine[3], mad cow[4], and Pizza[5]. We chose a set of classes that both represent commonplace concepts and cover most of the constructors available in OWL-DL. Since none of the concepts used required expert knowledge, users would be able to recognize whether the definitions seemed correct. We presented each subject with three NL definitions of the same concept, and asked them to choose the NL description they preferred based on correctness of the definition, readability, and clarity.

It was necessary to present the user with only one set of descriptions (i.e., a single trial per user), because each of the programs has a distinct and recognizable style of output, which would allow the user to potentially recognize their favorite algorithm from the previous trial, and simply choose it again, defeating the purpose of multiple trials. All definitions were copied into a standard text format, and presented separately from any of the tools (which were never seen by subjects) to prevent any such recognition.

In this small pilot study, all the users chose the output of our program over that of the other two. In informal interviews, users were impressed by the combination of the fluency (readability) of our output and the clarity with which the logical meaning behind the class definition was presented. Though this study is not extensive enough to be conclusive, it indicates that the approach we have pursued can be effective and appealing to users.

5 Open Issues

1. A key issue for our NL generation is the trade-off between readability and accuracy. To maintain a high level of accuracy, one could generate NL sentences of entity definitions following the model-theoretic semantics of OWL, such as representing $C \sqsubseteq \exists R.D$ as 'For each instance x of class C, there exists a relation R(x,y) such that y is an instance of class D'. However in such cases, overall readability of the sentence is greatly hampered. The choice of preserving readability vs. accuracy depends largely on end-user background/expertise and the application context.
2. Currently, the only support in OWL for providing NL paraphrases of entity definitions is via the annotation properties such as `rdfs:label` and `rdfs:comment`. Certain NL-specific tags could be added to OWL

[2] http://www.cs.concordia.ca/ying_lu/
[3] http://www.w3.org/2001/sw/WebOnt/guide-src/wine
[4] http://www.cs.man.ac.uk/~horrocks/OWL/Ontologies/mad_cows.owl
[5] http://www.co-ode.org/ontologies/pizza/pizza_20041007.owl

(or special-purpose annotation properties be used) in order to construct NL paraphrases from definitions more easily without introducing too much overhead for the ontology author. For example, if authors could specify which POS the entity ID belongs to (noun, verb, adjective) or an ordering format for the NL clause (in terms of POS or otherwise), auto-generation of phrases could be done more easily.

3. The presence of hyperlinks in the NL paraphrases can be immensely useful for navigating inter-related class definitions. However, given that entity names (IDs) get split across the NL sentence, positioning hyperlinks in the sentence is not easy. In such cases, tool-tips can be used to provide correlation between hyperlinks and the original entity references.
4. Another issue worth exploring is leveraging the power of DL reasoners in NL generation. For example, to make the sentence clearer, we are using the domain of the property as the subject in our generated phrases. Using techniques like this we managed to greatly improve the readability and to present much clearer meaning in our NL description. Additionally, an OWL-DL reasoner might infer new information about the class in question which will make the NL description even more informative.
5. Our approach takes advantage of the specific structure of constructions in English, such as the implication for all-values, as well as word orders specific to English. Thus, new systems would have to be built to support other languages. Perhaps more serious, our system relies on conventions for naming classes and properties that may not be found in all languages (for example, not all languages use a verb like 'has' to indicate possession or attribute specification). Thus, any parameterization of this system will need to be very sophisticated to accommodate the scope of natural languages.

6 Applications

1. *Semantic Annotation*: Recently, numerous tools for semantically annotating text, images, video etc have been developed [1]. Most of these tools use ontologies for driving the annotation process, allowing users to link their data with entities in the ontology. In order to support accurate and speedy annotation, NL description of the classes can be provided in order to explain the meaning of the concept and to point out its correct usage.
2. *Web-Service Advertising*: OWL-S based semantic web-services advertise themselves as instances of the service-profile. Rendering NL paraphrases of these service-profile instances can make web-service descriptions more accessible to end-users.
3. *Web-Policy (/Rules) Description:* In [4], the authors showed that Web-Service policies can be represented in OWL (using syntactic-sugar rules). However translating the WS-Policy operators (wsp:All, wsp:ExactlyOne) in OWL produced some non-trivial, complex class expressions. Policy developers new to OWL might find it difficult to

specify constraints and capabilities of their web services when working with these class expressions. NL paraphrases of the policies will make their meaning more accessible, thereby reducing the possiblity of error, without losing the intended semantics.

7 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. We have conducted a promising preliminary user evaluation, and plan to conduct formal user studies to fully evaluate the contribution of our work.

References

1. Photostuff image annotation tool. http://www.mindswap.org/2003/PhotoStuff/.
2. I. Androutsopoulos, S. Kallonis, and V. Karkaletsis. Exploiting OWL ontologies in the multilingual generation of object descriptions *Proceedings of the 10th European Workshop on Natural Language Generation (ENLG 2005)*, 2005.
3. R. Barzilay and L. Lee. Bootstrapping lexical choice via multiple-sequence alignment. *Proceedings of EMNLP*, 2002.
4. K. Bontcheva and Y. Wilks. Automatic report generation from ontologies: The MIAKT approach. *NLDB 2004*.
5. N. Fuchs, U. Schwertel, and S. Torge. Controlled natural language can replace first-order logic. *14th IEEE International Conference on Automated Software Engineering, Cocoa Beach, Florida*, Oct.
6. V. Kolovski, B. Parsia, Y. Katz, and J. Hendler. Representing web service policies in owl-dl. In *ISWC*, 2005.
7. 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.
8. G. Wilcock. Talking owls: Towards an ontology verbalizer. *Proceedings of the 2nd International Semantic Web Conference (ISWC)*, Oct. 2003.