Talking OWLs: Towards an Ontology Verbalizer

Graham Wilcock

University of Helsinki
00014 Helsinki, Finland
graham.wilcock@helsinki.fi

Abstract. The paper describes on-going work on an ontology verbalizer which can provide spoken summaries and explanations of the information specified in ontologies. The approach combines semantic web techniques with natural language generation and text-to-speech.

1 Previous work on generation from ontologies

Referring to ontologies formalized in Ontolingua, Aguado et al. say:

Our experience shows that domain experts and human final users do not understand formal ontologies codified in such languages even if such languages have a browser and a graphic user interface to display the ontology content. [1]

They describe a system that translates the ontology into natural language to help users understand it. To map from domain concepts to linguistic representations they use the Generalized Upper Model, based on the Penman Upper Model [2], as a “linguistic ontology”. Surface realization is done with KPML.

Frohlich and Riet [4] describe domain independent tools for generation based on “using different ontologies to represent the domain knowledge for different tasks of the generation process.” Like [1], they have a domain-specific layer at the top and a domain-independent layer based on the Penman Upper Model at the bottom, with KPML for surface realization.

These earlier projects used languages and tools such as Ontolingua, Penman Upper Model, LISP, LOOM and KPML. Although we can now use Java, XML, RDF and OWL, we still need to help users to understand the ontologies.

2 Generating summaries from RDF

In XML-based natural language generation [9], [10], [11], a pipeline of XSLT transformations implements the sequence of processing stages in an orthodox pipeline architecture for natural language generation. At the start of the pipeline, XSLT template-based generation creates an XML text plan tree whose leaves are domain concept messages. The text plan tree is transformed by the microplanning stages into an XML text specification tree whose leaves are linguistic phrase
specifications. The XSLT processors are embedded in Java, using SAX events to pass XML content efficiently down the pipeline.

XML-based generation has been used in a spoken dialogue system [6]. For spoken output, the final realization stage produces JSML (Java Speech Markup Language) [7] which is XML-based. The JSML is passed to FreeTTS [8], a speech synthesizer implemented entirely in Java.

XML-based generation can naturally be used for generation from RDF. A prototype implementation uses Jena [5] to feed content from RDF into the XSLT pipeline. Jena includes an RDF parser (ARP), an RDF query language (RDQL), and support for persistent storage of RDF models in relational databases.

```
<rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
    xmlns:vcard='http://www.w3.org/2001/vcard-rdf/3.0#'>
  <rdf:Description rdf:about='http://somewhere/JohnSmith/'>
    <vcard:FN>John Smith</vcard:FN>
    <vcard:N rdf:nodeID='A0'/>
    <vcard:EMAIL rdf:nodeID='A1'/>
  </rdf:Description>
  <rdf:Description rdf:nodeID='A1'>
    <rdf:value>John@somewhere.com</rdf:value>
    <rdf:type rdf:resource='http://www.w3.org/2001/vcard-rdf/3.0#internet'/>
  </rdf:Description>
  <rdf:Description rdf:nodeID='A0'>
    <vcard:Family>Smith</vcard:Family>
    <vcard:Given>John</vcard:Given>
  </rdf:Description>
</rdf:RDF>
```

Fig. 1. Example RDF description from the Jena tutorial

The simple RDF description in Figure 1 is taken from the Jena tutorial [5]. It describes a specific person (John Smith), not the general class of persons, and it uses the RDF encoding for vCard (visiting card) personal information [13]. If the natural language generator were limited to the information given explicitly in the RDF representation, it might produce something like Example 1.

Example 1.
This is a description of 'http://somewhere/JohnSmith/'. The description includes 3 items: 'vcard:FN', 'vcard:N' and 'vcard:EMAIL'.

The value of 'vcard:FN' is 'John Smith'.

The description of 'vcard:N' includes 2 items: 'vcard:Family' and 'vcard:Given'.

The value of 'vcard:Family' is 'Smith'. The value of 'vcard:Given' is 'John'.

The description of 'vcard:EMAIL' includes a value and a type. The value is 'John@somewhere.com'. The type is 'http://www.w3.org/2001/vcard-rdf/3.0#internet'.
However, the generator can exploit the use of vCard by providing predefined XSLT text plan templates for vCard, following the domain-specific approach of shallow generation [3], [10]. The values from the RDF representation are copied into the slots in the text plan template. By using knowledge about vCard, the generator can create a much better text plan equivalent to Example 2.

Example 2.
This is a description of John Smith identified by 'http://somewhere/JohnSmith/’. John Smith’s given name is ‘John’. John Smith’s family name is ‘Smith’. John Smith’s email address is ‘John@somewhere.com’. John Smith’s email address is type ‘internet’.

Of course, natural language generation can produce something more natural than this. The referring expressions stage of the generator can convert the text plan into a text specification equivalent to Example 3.

Example 3.
This is a description of John Smith identified by 'http://somewhere/JohnSmith/’. His given name is ‘John’. His family name is ‘Smith’. His email address is ‘John@somewhere.com’. It is ‘internet’ type.

Further, by performing sentence aggregation, the microplanning stages of the generator can produce a text specification equivalent to Example 4.

Example 4.
This is a description of John Smith identified by ‘http://somewhere/JohnSmith/’. His given name is ‘John’ and his family name is ‘Smith’. His email address, which is ‘internet’ type, is ‘John@somewhere.com’.

3 Generating explanations from DAML+OIL

The approach described in Section 2 is a form of shallow generation. One of the ideas in shallow generation [3] is to build domain-specific and task-specific generators, and not to attempt general solutions.

Naturally, shallow generation is compatible with a domain-specific ontology, but at first sight it seems incompatible with more general ontologies. However, Aguado et al. [1] claim that their rhetorical schemas represent standard patterns of scientific discourse, and they identified a number of stereotypical paragraph templates including definitions, comparisons, examples and classifications. If a small number of explanation schemas are sufficient to generate explanations from ontologies, then shallow generation can be used. This is an important point, to be investigated further.

The approach used for RDF can again be extended to process DAML+OIL. Jena [5] provides Java methods to read a DAML+OIL ontology and load it as a Jena model. There are also Jena methods to list all the ontology classes and to list all the properties. This provides a starting point for verbalising the ontology.
contents, but raw lists of classes and properties are very difficult to understand. In order to generate something which is an explanation of the ontology, the classes and properties need to be organised into meaningful groups.

This is on-going work. The RDF examples, the DAML+OIL processing, and the use of ontologies in spoken dialogue systems are discussed further in [12]. The current prototype uses RDF and DAML+OIL with Jena 1. Future work will use RDF and OWL\(^1\) with Jena 2.

References


\(^1\) I thank Lauri Carlson for the phrase “Talking OWLs” in the title.
Unlike simpler owl verbalizers, which typically express a single axiom at a time in controlled, often not entirely uent natural language primarily for the benet of domain experts, we aim to generate uent and coher-ent multi-sentence texts for end-users. With a system like Naturalowl, one can publish information in owl on the Web, along with automatically produced corresponding texts in multiple languages, making the information accessible not only to computer programs and domain experts, but also end-users. We discuss the processing stages of Naturalowl, the optional domain-dependent linguist... By reusing ontology verbalizers, existing ontology visualization systems can be easily extended with a verbalization service. Figure 2 illustrates how the proposed approach might work in practice: 1. Visualizer is the existing visualization component that transforms an OWL ontology into its graphical representation. 2. The system is extended by a User Selection mechanism that allows users to select the graphical element that they want to verbalize. 6. Gruzitis, N., Barzdins, G.: Towards a more natural multilingual controlled lan-guage interface to OWL. In: Proceedings of the 9th International Conference on Computational Semantics. pp. 1006â€“1013 (2011). 7. Horridge, M., Patel-Schneider, P.F.: OWL 2 Web Ontology Language Manchester Syntax. I have an Ontology written in OWL. Does anyone know I can load it into python? any packages or even manually? rdflib which is mentioned in other questions is not suitable for me because it mainly concerns with RDF and "Seth" which is a very nice library doesn't work, because it requires "Pellet" library for which the website seems to be down and it(seth) also only works with. OWL verbalizer demo. Introduction. This demo shows how an ontology that is written in OWL 2 XML is verbalized in Attempto Controlled English (ACE). This conversion is designed to be reversible, i.e. one can convert the ACE representation back into OWL so that no loss in meaning occurs. The source code, issue list, documentation etc. of the OWL verbalizer are available at https://github.com/Kaljurand/owl-verbalizer. Demo. Enter the content of an OWL ontology in the OWL 2 XML format. (You can use OWL Syntax Converter to convert from RDF/XML, Turtle, etc. into OWL 2 XML.) Towards OWL-based Knowledge Representation in Petrology. This paper presents our work on development of OWL-driven systems for fo 06/08/2011 â€“ by Alex Shkotin, et al. â€™ 0 â€™ share.Â By reusing ontology verbalizers, existing ontology visualization systems can be easily extended with a verbalization service. Figure 2 illustrates the proposed approach: Visualizer is the existing visualization component that transforms an OWL ontology into its graphical representation. The system is extended by a User Selection mechanism that allows users to select the graphical element that they want to verbalize. Collector gathers a subset of the ontology axioms that correspond to the selected graphical element.