Towards a Generation-Based Semantic Web Authoring Tool

Richard Power
Department of Computing
Open University
Milton Keynes, UK
r.power@open.ac.uk

Abstract

Widespread use of Semantic Web technologies requires interfaces through which knowledge can be viewed and edited without deep understanding of Description Logic and formalisms like OWL and RDF. Several groups are pursuing approaches based on Controlled Natural Languages (CNLs), so that editing can be performed by typing in sentences which are automatically interpreted as statements in OWL. We suggest here a variant of this approach which relies entirely on Natural Language Generation (NLG), and propose requirements for a system that can reliably generate transparent realisations of statements in Description Logic.

1 Introduction

We describe here a simple prototype of an editing tool that allows a user to create an ontology through an open-ended Natural Language interface. By 'open-ended' we mean that when introducing class or property names into the ontology, the user also decides how they should be expressed linguistically: thus the lexicon of the Natural Language interface is not predetermined. The purpose of such a tool is to support knowledge editing on the Semantic Web, which at present requires training in graphical user interfaces like Protégé (Rector et al., 2004), or direct coding in OWL and RDF. Linking OWL to Controlled Natural Language is currently the topic of an OWL1-1 task force, and several groups are already working in this area (Schwitter and Tilbrook, 2004; Thompson et al., 2005; Bernstein and Kaufmann, 2006; Pool, 2006; Dongilli, 2007); the novelty in our approach is that we rely entirely on Natural Language Generation (NLG), extending the WYSIWYM (or Conceptual Authoring) method (Power and Scott, 1998; Hallett et al., 2007) so that it supports knowledge editing for ontologies as well as for assertions about individuals.

The idea of linking formal and natural languages can be traced back to Frege (1879), who observed that mathematical proofs were made up of formulae *interspersed with passages in natural language*, and invented formal logic as a way of rendering these passages in a precise notation. With the arrival of Artificial Intelligence in the 1950s, formal logic became the foundation for knowledge representation and automatic reasoning — a trend leading to the recent concept of a 'semantic web' that would open up knowledge encoding and utilisation to a world-wide community (Berners-Lee et al., 2001). However, accessible knowledge management requires accessible presentation: hence the current interest in methods of 'sugaring' formal logic into natural language text (Ranta, 1994; Horacek, 1999), thus in a sense turning Frege upside-down.

1.1 Description Logic

The theoretical underpinning of OWL (and hence of the semantic web) is a discipline that evolved under various names in the 1980s and 1990s and is now called Description Logic (Baader et al., 2003). This refers not to a single logical language, but to a family of languages. All of these languages allow statements to be built from individuals, classes and properties, but they differ in the resources provided in order to construct classes and properties, thus allowing different balances to be drawn between the conflicting demands of expressiveness and tractability (i.e., decidability and efficiency of reasoning).

Figure 1 shows some common class constructors, using mathematical notation rather than OWL syntax (which is equivalent, but much lengthier). There are in fact three versions of OWL (Lite, DL and Full) which provide pro-

Proceedings of the 12th European Workshop on Natural Language Generation, pages 9–15,
Athens, Greece, 30 – 31 March 2009.

Description	Syntax
atomic class	A (etc.)
universal class	$\top$
negation	$\neg C$
intersection	$C \sqcap D$
union	$C \sqcup D$
value restriction	$\forall R.C$
exists restriction	$\exists R.C$
enumeration	$\{a\}$

Table 1: Class constructors

gressively more constructors, not only for classes but also for properties and axioms. Having chosen the desired logic, the ontology builder is free to introduce new atomic classes (and also properties and individuals), which can be given any name consistent with the RDF naming conventions (i.e., names must be Unique Resource Identifiers). Thus a new class might be named `http://myontology.net/parent` and a new property `http://myontology.net/hasChild`, although for brevity we will henceforth omit namespaces (i.e., $parent$, $hasChild$). Statements about classes can then be expressed by axioms, the most important of which are $C \sqsubseteq D$ (C is subsumed by D) and $C \equiv D$ (C is equivalent to D). For instance:

(1) $parent \equiv person \sqcap \exists hasChild.\top$
(2) $person \sqsubseteq \forall hasChild.person$

The meanings are probably obvious: (1) a parent is defined as a person with one or more children; (2) every person only has persons as children. Note that expressing these axioms in clear English is not trivial — for instance, in (2) we must take care not to imply that every person has children.

A collection of such axioms is called a TBox: intuitively, a TBox expresses concept definitions and generalisations. Description Logics also contain names for individual instances (e.g., $Abraham, Isaac$) and formulas expressing facts about individuals: thus $father(Abraham)$ would express class membership ('Abraham is a father'), and $hasChild(Abraham, Isaac)$ a relationship between individuals ('Isaac is Abraham's child'). A collection of such assertions is called an ABox, and TBox and ABox together make up a Knowledge Base (KB).

1.2 Reasoning services

The reason for proposing Description Logic as the foundation for the Semantic Web is that it allows for efficient reasoning services. Much effort is still going into the mathematical task of proving decidability and efficiency results for increasingly expressive dialects. Informally, the standard reasoning services are as follows:

1. **Class Satisfiability**: Checking whether in a given KB it is possible for a class to have at least one member.
2. **Subsumption**: Checking whether a given KB implies a specified subsumption relationship between two classes.
3. **Consistency**: Checking whether a given KB is consistent.
4. **Instance Checking**: Checking whether a given KB implies a specified ABox assertion that an individual a belongs to a class C.

Consider for instance the following miniature KB:

$man \sqcup woman \equiv person$
$man \sqsubseteq \neg woman$
$man(Abraham)$

In respect to this KB, a reasoning engine should be able to show (1) that the class $man \sqcap woman$ is unsatisfiable, (2) that man is subsumed by $person$ ($man \sqsubseteq person$), (3) that the KB is consistent, and (4) that the assertion $person(Abraham)$ holds.

The ability to perform these reasoning tasks efficiently can be exploited not only in applications that utilize knowledge in problem-solving, but in knowledge editing and natural language generation. For instance, when an ontology builder adds a new axiom to a KB, the editor can run the subsumption and consistency checks and give feedback on whether the axiom is informative, redundant, or inconsistent. Or when producing a natural language gloss for the class $\exists hasChild.female$, the generator could choose between 'something with at least one female child' and 'someone with at least one female child' by checking the subsumption relationship $\exists hasChild.female \sqsubseteq person$.

2 Aligning DL to CNL

We have explained informally the technical features of description logics. Briefly, they include rules for constructing classes, axioms, and assertions about individuals; the resulting expressions

are interpreted through a relatively simple model-theoretic semantics (Baader et al., 2005). They also include efficient algorithms for performing reasoning tasks. We now turn to issues in the design of Controlled Natural Languages (CNLs) which can be aligned with specific DLs, and thus serve as useful interfaces for working with complex formalisms like OWL and RDF.

As a provisional list of requirements, we would suggest the following:

1. **Completeness**: A sentence (or text) can be generated for any axiom permitted by the DL.
2. **Uniqueness**: Different sentences are generated for different axioms.
3. **Transparency**: Sentences in the CNL are accurately interpreted by human readers.
4. **Fluency**: Sentences in the CNL are interpreted easily by human readers and judged natural.
5. **Interpretability**: Sentences conforming to the CNL can be automatically interpreted to recover the corresponding DL axiom.
6. **Editability**: Interactive texts in the CNL can be manipulated by domain experts in order to extend and revise the KB.
7. **Extensibility**: Domain experts can extend the CNL by linking lexical entries to new individuals, classes or properties in the KB.

Note that these are essentially practical requirements, which concern the CNL's role as an interface for a particular DL. We see no reason to insist that the alignment between DL and CNL should conform to general theories of natural language semantics.

2.1 Completeness

If we propose to use generated CNL as an interface to a knowledge base, it is important that generation should be reliable. A minimal test of reliability is that the grammar and lexicon are complete, in the sense that they produce a text for any well-formed semantic input. Elsewhere, we have described a generation method that allows completeness to be checked by a computer program (Hardcastle and Power, 2008). For any non-trivial DL the set of classes is infinite (e.g., through recursion on $C \sqcap D$ or $\exists R.C$); however, completeness can be proved through an enumeration of all local contexts for which a linguistic realisation rule is needed. As well as guaranteeing reliability, completeness checking is obviously useful as an aid to grammar development.

2.2 Uniqueness

Although necessary, completeness is not a sufficient condition on the grammar of a CNL, since it could be trivially met by generating the same string (perhaps 'Hallo World') for any semantic input. It would also be useful to have an automatic check that the same sentence is not generated for two different semantic inputs — i.e., that every sentence in the CNL has a unique meaning. This seems a harder problem than completeness, and we have not seen any proposals on how it could be done.

To pose this problem precisely we would need to define what is meant by 'different' semantic inputs. Complex class descriptions can be manipulated by well-known logical equivalences like De Morgan's laws: for instance, $\neg(C \sqcap D)$ is equivalent to $(\neg C) \sqcup (\neg D)$. Should these be treated as different inputs or the same input? We think users would probably prefer them to be treated as different, but the issue needs to be investigated further.

2.3 Transparency

Transparency is obviously at the heart of the enterprise: completeness and uniqueness proofs are no help if the generated texts are unclear to human readers. Unlike the preceding requirements, transparency is a matter of degree: we cannot expect, far less prove, that every sentence in the CNL will be accurately understood by all target users on all occasions. Transparency can only be assessed, and gradually improved, through experiments and user feedback.

2.4 Fluency

Fluency is another aspect of readability: whereas transparency concerns *accuracy* of interpretation, fluency concerns *ease*. These requirements potentially conflict. For instance, to express the axiom $parent \sqsubseteq \exists hasChild.\top$ fluently we could say 'every parent has a child', while for transparency we might prefer the pedantic 'every parent has one or more children'. In a CNL designed for editing a KB, transparency will have priority, but one can imagine other purposes (e.g., an informal report) for which fluency would matter more.

2.5 Interpretability

This is an essential requirement for knowledge editors that rely on automatic parsing and interpretation of texts typed in by human authors (Schwitter and Tilbrook, 2004; Bernstein and Kaufmann, 2006). A recent innovation has been to pursue the goal of 'roundtripping' (Davis et al., 2008), so that a CNL text can be generated from an existing ontology, revised in a text editor, and then interpreted automatically to obtain an updated ontology in the original format. For our approach, which relies entirely on generation, automatic interpretability is not essential (although one can imagine contexts in which it would be useful, for instance to allow knowledge encoding outside the NLG-based editing environment).

2.6 Editability

The key feature of Conceptual Authoring (WYSIWYM) is that editing operations are defined on the semantic input, not the text. This means that authors cannot produce a text in the normal way by typing in words from left to right. Some kind of non-specific initial configuration has to be gradually refined through semantic distinctions made by choices from menus (an example will be given later). To validate the approach, it has to be shown that this editing process is efficient and easily learned. Usability results from ABox editing applications have been encouraging (Hallett et al., 2007), but whether similar results can be achieved for KB editing (TBox as well as ABox) remains unproven.

2.7 Extensibility

Ontology development requires that authors should be able to introduce new terms for individuals, classes and properties. The designer of a CNL-based editor cannot foresee what these terms will be, and therefore cannot provide a mapping to suitable lexical entries. This must be done by the ontology developer, and take-up accordingly depends on making this task not only feasible but easy (Hielkema et al., 2007). We will explore two ideas on how this might be done: (a) providing a wide-coverage lexicon from which users can select words to extend the CNL, and (b) using conventions for controlling the naming of classes and properties, so that the two decisions (term name, CNL lexical entry) become essentially a single decision.

3 Editing process

As a first experiment we have written a Prolog program which allows a KB to be built up from scratch for a very simple DL with only one kind of statement ($C \sqsubseteq D$), four class constructors (A, $\top$, $\exists R.C$, $\{a\}$), and one property constructor (property inversion, which will be explained shortly). Using just these resources we can formulate ABox assertions as well as TBox axioms by the trick of representing individuals through enumerated classes. For instance, $man(Abraham)$ can be asserted through the axiom $\{Abraham\} \sqsubseteq man$ (the class containing only Abraham is a subclass of the class of men).

A generic grammar is provided for realising axioms and complex class descriptions (a handful of rules suffices); the grammar assumes that the words for realising individuals, atomic classes and properties will conform to the following (very strict) regulations:

1. Individuals are realised by proper names
2. Atomic classes are realised by count nouns
3. Properties are realised either by transitive verbs or by count nouns

We also simplify by assuming that the name of every atomic term in the DL is identical to the root form of the word realising the term — for instance, the count noun realising the class $person$ will be 'person'.

When the editor is launched there are no individuals, atomic classes or properties, and the only word in the lexicon is 'thing', which denotes the class $\top$ (i.e., the class containing all individuals). The KB is construed as a sequence of axioms, and to start the ball rolling it is seeded with a single vacuous axiom $\top \sqsubseteq \top$. The program generates a sentence expressing this axiom and adds a list of editing options as follows:

```
1: Every thing/1 is a thing/2.

t   Add a new term
a   Add a new axiom
A/C Edit class C in axiom A
A/d Delete axiom A
```

Note that in every sentence expressing an axiom, the head word of every span denoting a class is given a numerical label; in a simple Prolog interface this allows the class to be selected for editing. There is no point in attempting any editing yet, since no terms have been introduced.

Word	Syntax	Type
Mary	name	individual
pet	noun	class
animal	noun	class
own	verb	property

Table 2: Lexical entries for terms

The user should therefore choose option `t` to add a new term. This is done by specifying three things: a word (any string), a syntactic category (either `name`, `noun`, or `verb`), and a logical type (`individual`, `class`, or `property`). In this way the user might define the set of terms in figure 2 from the people+pets domain, which will be familiar to students of Description Logic.

Editing of the axiom $\top \sqsubseteq \top$ can now begin. Assuming that the target is $pet \sqsubseteq animal$, the user first selects the first class in the first axiom by typing `1/1` (in a GUI this would be done simply by clicking on the word). The program returns a menu of substitutions computed from the current ontology and expressed in English phrases adapted to the context of the selected class:

```
1 Mary
2 Every pet
3 Every animal
4 Everything that owns one or more things
5 Everything owned by one or more things
```

These phrases express respectively the classes $\{Mary\}$, pet, $animal$, $\exists own.\top$ and $\exists own^{-1}.\top$ which can be formed from the terms in figure 2. Note that the last class results from the inversion of the property own: if $own(a, b)$ means that a owns b, the inverse $own^{-1}(a, b)$ means that b owns a — a relationship that can conveniently be expressed by passivisation (a is owned by b).

When the user chooses option 2 (in a GUI this would of course be done by clicking on the menu item), the program updates the knowledge base and regenerates:

```
1: Every pet/1 is a thing/2
```

Selecting the second class by typing `1/2` now yields the same menu of options, differently phrased to suit the different context of the class in the axiom:

```
1 Mary
2 a pet
3 an animal
4 owns one or more things
5 is owned by one or more things
```

Choosing option 3 completes the first axiom, after which the user can use the option `a` (see above) to obtain a second default axiom for editing:

```
1: Every pet/1 is an animal/2
2: Every thing/1 is a thing/2
```

A similar series of operations on the second axiom (starting by selecting `2/1`) might then yield the following:

```
1: Every pet/1 is an animal/2
2: Mary/1 owns/2 one or more pets/3
```

Even in such a simple example, we can see how editing could be supported by the reasoning services. For instance, if the user added a third axiom 'Mary owns one or more animals', the program could point out that this is redundant, since $\{Mary\} \sqsubseteq \exists own.animal$ can be deduced from $pet \sqsubseteq animal$ and $\{Mary\} \sqsubseteq \exists own.pet$.

4 Discussion

We have shown through a small prototype how a KB could be built up from scratch using an NLG-based authoring tool, with the lexicon almost entirely specified by the ontology developer. Although modest in scope, the prototype extends previous work on Conceptual Authoring (WYSIWYM) in several ways:

- It allows editing of the TBox as well as the ABox, by defining editing operations on classes rather than individuals (with individuals treated as singleton enumerated classes).
- It allows users to extend the CNL through the constrained choice of words/phrases to express new individuals, classes and properties.
- It allows feedback based on reasoning services (e.g, on whether a new axiom is inconsistent, informative or redundant).

An obvious objection to our approach is that we are increasing the load on users by requiring them to build not only a KB but also a CNL lexicon. Much will therefore depend on the tools that support users in the latter task. Ideally, the construction of a lexical entry would depend on making a single selection from a wide-coverage lexicon that has already been built by computational linguists. However, although this ideal is feasible for classes and properties like pet and own which can be mapped to single words, any encounter with real ontologies is likely to reveal terms like $hasDietaryPreference$ that would have to be

expressed by a phrase. Probably there are solutions to this problem — one could imagine for instance an algorithm that builds new entries in a phrasal lexicon from examples — but they remain to be demonstrated and tested.

More generally, an important question is whether such a method will scale up. It seems to work reasonably well in the above example with a handful of class constructors, terms and axioms, but what happens when we tackle an expressive DL like OWL Full, and support the editing of a KB with thousands of terms and axioms?

As regards more expressive DLs, we have already cited promising work on developing CNLs for OWL. As one might expect, the Boolean class constructors ($C \sqcap D$, $C \sqcup D$, $\neg C$) can lead to problems of structural ambiguity, e.g. in a description like $old \sqcap (man \sqcup woman)$. Here an NLG-based editor should have the advantage over one that requires human authoring of texts, since it can apply the best available aids of punctuation and formatting (Hallett et al., 2007), a task that would require great care and skill from human authors.

Increasing the number of terms would mean that during editing, classes had to be selected from thousands of alternatives; some kind of search mechanism would therefore be needed. A simple solution already used in WYSIWYM applications (Bouayad-Agha et al., 2002; Hallett et al., 2007; Evans et al., 2008) is a menu equipped with a text field allowing users to narrow the focus by typing in some characters from the desired word or phrase. In an ontology editor this search mechanism could be enhanced by using the ontology itself in order to pick options that are conceptual rather than orthographic neighbours — for instance on typing in 'dog' the user would obtain a focussed list containing 'poodle' and 'pekingese' as well as 'doggerel'.

Increasing the number of axioms has no effect on the editing process, since we are assuming that axioms will be realised by separate sentences, each generated without regard to context. However, a text comprising a long list of unorganised axioms hardly makes for easy reading or navigation. There is therefore potential here for a more interesting application of NLG technology that would draw on topics like generation of referring expressions, pronominalisation, aggregation, discourse planning, and summarisation. Presenting a KB through a fluent and well-organised report would give users a valuable return on their efforts in linking terms to lexical entries, and would address a pressing problem in ontology building — how to maintain trasparency in an ontology as it expands, possibly through contributions from multiple users.

In a word, the advantage of applying NLG in this area is *flexibility*. Once we have a mapping from logical terms to lexical entries in English or another natural language, we can tailor generated texts to different tasks in knowledge management, using fluent organised reports for purposes of overview and navigation, and short pedantically precise sentences for editing — backed up if necessary with footnotes explaining unintuitive logical implications in detail, or painstakingly formatted Boolean constructions that avoid potential structural ambiguities.

References

Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. 2003. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

F. Baader, I. R. Horrocks, and U. Sattler. 2005. Description logics as ontology languages for the semantic web. *Lecture Notes in Artificial Intelligence*, 2605:228–248.

T. Berners-Lee, J. Hendler, and O. Lassila. 2001. The semantic web. *Scientific American*, 284(5):34–43.

A. Bernstein and E. Kaufmann. 2006. GINO – a guided input natural language ontology editor. In *Proceedings of the 5th International Semantic Web Conference*, Athens, Georgia.

Nadjet Bouayad-Agha, Richard Power, Donia Scott, and Anja Belz. 2002. PILLS: Multilingual generation of medical information documents with overlapping content. In *Proceedings of the Third International Conference on Language Resoures and Evaluation (LREC 2002)*, pages 2111–2114, Las Palmas.

Brian Davis, Ahmad Ali Iqbal, Adam Funk, Valentin Tablan, Kalina Bontcheva, Hamish Cunningham, and Siegfried Handschuh. 2008. Roundtrip ontology authoring. In *International Semantic Web Conference*, volume 5318 of *Lecture Notes in Computer Science*, pages 50–65. Springer.

Paolo Dongilli. 2007. Discourse Planning Strategies for Complex Concept Descriptions. In *Proceedings of the 7th International Symposium on Natural Language Processing*, Pattaya, Chonburi, Thailand.

R. Evans, P. Piwek, L. Cahill, and N. Tipper. 2008. Natural Language Processing in CLIME, a Multilingual Legal Advisory System. *Journal of Natural Language Engineering*, 14(1):101–132.

Gottlob Frege. 1879. *Begriffsschrift*. Halle.

Catalina Hallett, Donia Scott, and Richard Power. 2007. Composing queries through conceptual authoring. *Computational Linguistics*, 33(1):105–133.

D. Hardcastle and R. Power. 2008. Fast, Scalable and Reliable Generation of Controlled Natural Language. In *Proceedings of SETQA-NLP Workshop at the 46th Annual Meeting of the Association for Computational Linguistics*, Ohio, US.

F. Hielkema, C. Mellish, and P. Edwards. 2007. Using WYSIWYM to create an open-ended interface for the semantic grid. In *Proceedings of the 11th European Workshop on Natural Language Generation*, Schloss Dagstuhl.

Helmut Horacek. 1999. Presenting Proofs in a Human-Oriented Way. In *Proceedings of the 16th International Conference on Automated Deduction*, pages 142–156, London, UK. Springer-Verlag.

J. Pool. 2006. Can controlled languages scale to the web? In *5th International Workshop on Controlled Language Applications (CLAW'06)*, Boston, USA.

R. Power and D. Scott. 1998. Multilingual authoring using feedback texts. In *Proceedings of the 17th International Conference on Computational Linguistics and 36th Annual Meeting of the Association for Computational Linguistics*, pages 1053–1059, Montreal, Canada.

Aarne Ranta. 1994. Type theory and the informal language of mathematics. In *Proceedings of the 1993 Types Worshop, Nijmegen, LNCS 806*, pages 352–365. Spinger Verlag.

Alan Rector, Nick Drummond, Matthew Horridge, Jeremy Rogers, Holger Knublauch, Robert Stevens, Hai Wang, and Chris Wroe. 2004. OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors and Common Patterns. In *14th International Conference on Knowledge Engineering and Knowledge Management*, pages 63–81.

R. Schwitter and M. Tilbrook. 2004. Controlled natural language meets the semantic web. In *Proceedings of the Australasian Language Technology Workshop*, pages 55–62, Macquarie University.

C. Thompson, P. Pazandak, and H. Tennant. 2005. Talk to your semantic web. *IEEE Internet Computing*, 9(6):75–78.