An Information Extraction System for English Ontology Identifier Names

Sandra Williams
The Open University, Milton Keynes, MK7 6AA, U.K.

Abstract

I describe a system, `Txt2ids`, that uses a series of regular expressions to extract suggestions for ontology identifier names from English text and classify them as (i) class names, (ii) individual names, (iii) object property names, or (iv) data property names. As well as being of practical use as a tool in an ontology authoring system, it also functions as a theoretical model of the syntactic organisation of identifier names.

Regular expressions were derived from part-of-speech patterns in identifier names in a corpus of over 500 ontologies. Since ontology identifier names have syntactic structures that differ from natural English, the regular expressions were adapted. Extracted phrases were post-processed to comply with the structure of OWL Simplified English.

A system sanity test achieved acceptable results when comparing identifiers extracted by `Txt2ids` (from texts that had been automatically generated by an ontology verbaliser from a large corpus of ontologies) with the original identifiers from the same corpus. `Txt2ids` tends to generate greater numbers of identifiers than were present in the original ontology; however, many of the additional ones seem reasonable suggestions.

To assist in the design of a future system evaluation, a pilot study was conducted in which identifier names extracted by `Txt2ids` from short, expository texts compared favourably with those created by human users when building ontologies from the same texts.

The system has been deployed in an ontology editor developed for the `SWAT`[1] project.

1 Introduction

Ontologies are employed extensively in the semantic web as a means of representing human knowledge. Expressed in ontology languages such as Web

[1]Semantic Web Authoring Tool (SWAT), a project funded by the U.K. Engineering and Physical Science Research Council (EPSRC grant no. G033579/1).

Ontology Language (OWL), they are underpinned by simple description logics, which include statements (axioms) such as: $A \sqsubseteq B$ (A is a subclass of B, e.g., every man is a human), $A \sqsubseteq \exists P.B$ (A is a subclass of things that have property P (relationship) with at least one B, e.g., every pizza is covered in a cheese topping), $[a, b] \in P$ ('a' and 'b' are individuals related by the property P, e.g., John loves Mary), and $a \in A$ ('a' is a member of class A, e.g., John is a man).

Building new ontologies is a difficult and time-consuming task requiring expertise in an ontology language and an understanding of the underlying logic. The SWAT project therefore constructed a number of software tools (Stevens et al., 2010; Power, 2012a; Third, 2012) that will help to make ontology authoring simpler and quicker for people faced with the daunting prospect of translating human knowledge into ontology language.

A useful starting point for ontology authoring is an expository natural language text that expresses some of the knowledge to be represented. For example, supposing I want to represent knowledge about spiders, I would first need to think of words and phrases that would become class names, property names, and so on, in the ontology—a tedious task. Alternatively I could begin by finding an article about spiders and then extracting the relevant words and phrases from it, but again it would be a tedious task. Why not automate part of this process? Such automation is precisely the subject of this report: automatically extracting potential ontology identifier names from natural language text.

This technical report describes a natural language processing system which recognises words and phrases in a text that could serve as ontology identifier names. The input to the program is unrestricted English text in a plain text file, and the output is a file containing list of suggestions for identifier names classified into (i) classes, (ii) individuals, (iii) object properties, and (iv) data properties. So, for example, for the input text:

> Spiders are arachnid arthropods. They have eight legs, and most make silk. Some of them have six eyes, some of them have eight eyes, some have four eyes, two eyes, or no eyes. Spiders are the biggest group in arachnids, after Acarina, the mites and ticks. Some spiders are as small as 0.5 mm[1] and some are as large as 10cm (4 inches) in body length. Most spiders eat insects, other spiders, etc. They catch the insects and other prey in several ways. Some build a spider web, and some use a thread of silk that they throw at the insect. Some kinds of spiders hide in holes in the ground, then run out and grab an insect that walks by. Others will make web "nets" to throw at passing insects. Or they will even go out and attack them with no webs at all. Spiders do not have wings. Some can jump quite well and hunt by sneaking close to an insect and then jumping on it. (First paragraph of 'spider' entry in SimpleWikipedia[2])

[2]http://simple.wikipedia.org/wiki/Spider accessed September 13, 2012.

the program produces as output the lists of suggestions for identifier names shown in table 1. Some suggestions are more useful than others since the part-of-speech tagger that the system employs is far from perfect (e.g., the class 'other' is extracted from the sentence 'Others will make ...' where 'others' is tagged as a plural noun). However, it is our intention to present the output to a user so that he/she can select or modify the ones needed, discarding the others. That is, we intend the program to be used as a *support tool* rather than as a completely automatic system.

Table 1: Output phrases extracted and classified into identifier types by the `Txt2ids` system from an unrestricted English input paragraph about spiders

Classes	arachnid	inch	prey in way
	arachnid arthropod	insect	silk
	body length	kind	spider
	eye	kind of spider	spider web
	ground	leg	thread
	group	mite	thread of silk
	group in arachnid	mm	tick
	hole	net	way
	hole in the ground	other	web
	hunt	prey	wing
Individuals	Acarina		
Object Properties	attacks	is kind of	is thread of
	builds	is prey in	is web at
	catches	is silk that	jumps
	grabs	is spider are	makes
	is group in	is spider do	runs
	is hole in	is spider eat	throws
	is hunt by	is spider hide	uses
Data Properties	has eyes in number	has inches in number	has mm in number
	has legs in number		

The following section describes the approach; section 3 explains in detail how the system was implemented, section 4 describes system sanity testing, section 5 a pilot evaluation, and section 6 compares our system to related work.

2 The approach

While the task of the `Txt2ids` system might be compared to that of terminology extraction systems, in fact it has distinct differences. The aim of terminology extraction is to locate *technical* terms (essentially noun phrases such as 'arachnid arthropod' (i.e., ontology class names) and proper names such as 'Incy Wincy Spider' (named individuals) that are typical of a subject area. This is also the aim of `Txt2ids` but, additionally, it aims to extract

verbs and verb groups such as 'eats' and 'has as number of eyes' as property names and also more generic terms for class names, e.g., 'web' as well as 'spider web'. Thus `Txt2ids` is an information extraction system adapted for the special purpose of extracting ontology identifier names.

Because of these additional requirements, our approach differs from a single-pass system (see also the discussion in section 6). Essentially, the input text is analysed several times by a series of regular expression grammars (REGs), each designed to extract a particular type of identifier name. Multiple passes are required because phrases for different types of identifers overlap; e.g., in '... spins a thread of silk' the property name 'spins [a] thread of' would overlap with the class name 'thread of silk'. A similar technique has also been used by Maedche and Staab (2000).

Chunk parsing, see section 7.2 in the NLTK reference book (Bird, Klein, and Loper, 2009), is a technique for extracting sequences of part-of-speech (POS) tags that conform to a REG. It is particularly suitable for our purpose since identifier names are single words or short sequences of words. However, from our analyses of existing ontology identifiers (Williams, 2013), it is clear that the required phrases do not correspond directly to English syntactic phrases; in fact, they tend to be telegraphic in nature, missing out determiners. Consequently, whilst linguistic knowledge of typical English constituents can be employed in designing REGs for parsing English, REGs must be specially adapted for this particular task and extracted phrases require post-processing so that they conform to OWL Simplified English (Power, 2012a), the controlled natural language used in SWAT tools.

To locate class names, for instance, we would want to search for sequences similar to noun phrases. For example, with the sentence:

> Some build a spider web, and some use a thread of silk that they throw at the insect.

we want to extract 'spider web', 'insect', and so on, as class names, but my analysis of human-written class names presented in Williams (2013), shows that we do not require the determiners found conventional noun phrases. When the sentence has been POS tagged (see table 3 for descriptions of the POS tags), the result is:

> Some_DT build_VBP a_DT spider_NN web_NN ,_, and_CC some_DT use_VBP a_DT thread_NN of_IN silk_NN that_IN they_PRP throw_VBP at_IN the_DT insect_NN ._.

It is now easy to see that to extract 'spider web' we need to include the POS sequence 'NN NN' in our REG.

But going further, we also extract nouns on their own i.e., 'web' and 'spider', that could form useful superclass identifiers (Third, 2012) and nouns followed by prepositions (e.g., 'thread of') which are commonly used by ontology developers as property names (e.g., 'is thread of'). *Adjectives* could be extracted as potential class names e.g., 'six-eyed' could form the class of

'six-eyed things'[3], however, to reduce over-generation of spurious identifiers, `Txt2ids` does not currently generate an identifier name suggestion from every adjective.

Table 2 shows examples of the types of rules present in `Txt2ids` (full details are provided in section 3). In the left-hand column, the first rule extracts adjectives followed by nouns, the second column shows an example, 'crested newts'. When processed further these become singular suggested names, e.g., 'crested newt' (third column) for the identifier type 'class' (fourth column). Another rule extracts proper nouns that become suggestions for named individuals (e.g., 'London'), another extracts verbs followed by NPs, e.g., 'play the role', which is processed further to become third-person-singular suggestions for object property names 'plays the role', and so on. At the bottom of the table is a rule that extracts numbers followed by nouns (e.g., '2mm') which are further processed to become suggestions for data property names (e.g., 'has mm in number').

Table 2: Chunk parser iterations applied by `Txt2ids`

Regular expression rule	**Example phrases extracted**	**Post-processed phrase**	**Ontology identifier type**
adjective(s) noun(s)	crested newts	crested newt	class
noun(s) preposition NP	newts with a crest	newt with a crest	class
noun(s) no(s). noun(s)	type 2 ions	type 2 ion	class
foreign word(s)	exerpes asper	exerpe asper	class
proper noun(s)	London	London	individual
p. noun(s) no(s).	Sarah Rever 1846	Sarah Rever 1846	individual
noun(s) prep. p. NP	brother of the Pope	brother of the Pope	individual
verb (present) NP (prep.)	plays the role	plays the role	object property
verb (present not 3rd) NP	play the role	plays the role	object property
verb (past participle) prep.	are eaten by	is eaten by	object property
verb (past) prep.	kept in	is kept in	object property
noun(s) prep.	distinct part of	is distinct part of	object property
verb (base)	surround	surrounds	object property
no(s). noun(s)	2mm	has mm in number	data property

A distinctive feature of the approach is the design of the RE rules. These were developed from an analysis of common patterns in existing ontology identifiers (Williams, 2013) with some additional intuitions from linguistics.[4] Previously in the SWAT project, we had annotated a small corpus

[3]Further research is required to determine which combinations of adjectives and nouns are the most frequently used as ontology identifiers from a given noun phrase in a text. For example, suppose a text contained 'spotted, shorthorned grasshopper', which (if any) of 'spotted, short-horned things', 'spotted things', 'short-horned things', 'spotted grasshopper', and 'short-horned grasshopper' would be useful as ontology class names?

[4]This raises another question about whether POS patterns from natural English text do

of ontology identifiers extracted from around 50 ontologies (Power, 2010a) with the CLAWS POS tagger. Our initial experiments with `Txt2ids` used the POS tag patterns resulting from that analysis which were converted to Penn tags and used to derive rules. However, a second annotation using the Stanford POS tagger (Toutanova et al., 2003) on a much larger corpus of 521 ontologies gave us what we hope are more accurate frequencies for POS tag patterns, which were employed in the design of subsequent versions of the system.

3 Implementation

The system is implemented in Java using the Java regular expression library. It operates by performing the following processes:

1. Read in an input natural English text file and split it into sentences and words.
2. Annotate each sentence with POS tags.
3. Apply regular expression (RE) chunk parsers.
4. Post-process extracted phrases.

These are explained in the following subsections where the chunk parsing subsection includes design of the REs.

3.1 Sentence and Word Splitting

The program reads in the input file line-by-line assuming that each line is a single paragraph and therefore that no line break or line feed characters will occur mid-sentence or mid-paragraph. Each paragraph is split into sentences and words using the Java String class built-in sentence and word tokenisers.

3.2 Part-of-Speech Tagging

Part-of-speech tagging, i.e., the task of annotating the input text with parts of speech, is the chief limitation of the current system. The program uses the Stanford POS tagger (Toutanova et al., 2003) which is ready-trained on large corpora and uses the Penn tagset (Santorini, 1990) reproduced in table 3. Developers of the Stanford POS tagger claim a token accuracy of over

in fact correspond to ontology identifier POS patterns. This question could be investigated through the NLTK toolkit (Bird, Klein, and Loper, 2009) which provides large corpora of POS-tagged English text and built-in functions for extracting all phrases matching a given tagstring pattern, so it would be relatively easy to write a program to print out the tagstring patterns together with numbers of matching phrases to find out how common they are in natural English.

Table 3: Penn part-of-speech tagset

Tag	Description	Tag	Description
CC	conjunction, coordinating	PRP$	pronoun, possessive
CD	numeral, cardinal	RB	adverb
DT	determiner	RBR	adverb, comparative
EX	existential there	RBS	adverb, superlative
FW	foreign word	RP	particle
IN	preposition, conjunction	SYM	symbol
JJ	adjective or numeral, ordinal	TO	'to' prep. or infin. marker
JJR	adjective, comparative	UH	interjection
JJS	adjective, superlative	VB	verb, base form
LS	List item marker	VBD	verb, past tense
MD	modal auxiliary	VBG	verb, pres. participle, gerund
NN	noun, common, sing. or mass	VBN	verb, past participle
NNP	noun, proper, sing.	VBP	verb, pres. tense, not 3rd sing.
NNPS	noun, proper, plural	VBZ	verb, pres. tense, 3rd sing.
NNS	noun, common, plural	WDT	WH-determiner
PDT	pre-determiner	WP	WH-pronoun
POS	genitive marker	WP$	WH-pronoun, possessive
PRP	pronoun, personal	WRB	Wh-adverb

97% (i.e., two or three mis-tagged words per 100) for test texts from the same corpus as the texts on which it was trained. Accuracy for unknown words is claimed at over 80%.

The Stanford POS tagger[5] is easily deployed within a Java program by implementing an external system call (in effect, the same as calling it from the command line in a Linux shell).

Apart from mis-tags, other disadvantages of the tagger are that it is slow and it requires a lot of memory, especially for texts of 1MB or more (I allocated 2GB of RAM in the system call).

3.3 Chunk parsing

Chunk parsing extracts relevant phrases from a text according to regular expression grammars which define legal sequences of POS tags. In the following sections, I describe the REs that I designed to extract each type of ontology identifier[6]. RE symbols used in this section are detailed in table 4.

Since every type of human-written identifier analysed in Williams (2013) comprises a high proportion of POS tag patterns that are common nouns, I had to make a design decision about which category to assign to these patterns. The choice I made was to include common noun patterns in the

[5]I used the December 2011 version of the Stanford POS tagger, downloaded from http://nlp.stanford.edu/software/tagger.shtml in May 2012

[6]They were implemented using the Pattern and Matcher classes in the Java regular expression library, java.util.regex

Table 4: Regular expression symbols

<>	delimit an element
$*$	zero or more occurrences, e.g., $< JJ > *$
$+$	one or more occurrences, e.g., $< NN > +$
?	optional element, e.g., $< DT >?$
.	wildcard matches any single character
$.*$	zero or more characters, e.g., $NN.*$
?	the previous character is optional, e.g., $NNS?$

REGs for classes because classes are by far the most common category in the corpus (over 62,000 of over 88,000 in total). This decision biases identifier extraction in favour of class names. Thus I would expect `Txt2ids`'s performance to be higher on class identifiers than on other types.

My implementation matches each RE in turn to every sentence POS string in the text. Once an RE has been matched to a POS string, there is no backtracking.

3.3.1 Extracting class identifiers

To extract class identifiers, four REs shown in table 5 were employed. They recognise: (i) phrases that are made up of zero or more adjectives and one or more singular or plural nouns; (ii) as (i) but with the addition of one or more prepositional phrases consisting of a preposition, optional adjectives, and one or more nouns, (iii) zero or more adjectives followed by one or more numerical phrases consisting of one or more singular or plural nouns, and one or more numerals; and (iv) one or more foreign words. Thus, in table 1, which shows system output for the spider text, it extracts the class 'spider web' from the POS tag sequence 'NN NN' from RE (i), the class 'thread of silk' from the sequence 'NN IN NN' from (ii), and so on.

Table 5: CLASS IDENTIFIER REGULAR EXPRESSIONS. The four REs shown account for 91% of all class identifiers.

Regular Expression	**No. Patterns**	**Freq**
<JJ> $*$ <NNS?> $+$	45	46483
<JJ> $*$ <NNS?> $+$ <<IN> $+$ <JJ> $*$ <NNS?> $+$ > $+$	143	7117
<JJ> $*$ <<NNS?> $+$ <CD> $+$ > $+$	6	2792
<FW> $+$	8	620
Total (above REs)	202	**57012**
Grand Total (class identifiers)	1357	**62788**

The first RE in the table might at first sight appear to be surplus to requirements, but remember that it is needed to ensure that head nouns

on their own *and* postmodified nouns are extracted. Thus, the program extracts 'thread' and 'silk' as well as 'thread of silk'.

REs were designed by analysing the POS tag patterns for class identifiers as illustrated in table 5. Numbers of POS tag patterns that each RE covers are shown in the second column (headed 'no. patterns'). The table also shows accumulated frequencies of tag patterns that are covered by the RE in the 3rd column (headed 'Freq'). Note that although there are 1357 unique patterns in the corpus and the four REs cover only 202 of them, however, the total coverage of identifiers is 57012 of the 62788 class identifiers in the corpus, or **91%**. So, very few REs account for the vast majority of identifiers in the corpus.

3.3.2 Extracting named individual identifiers

Table 6: NAMED INDIVIDUAL REGULAR EXPRESSIONS. The REs shown account for 43% of all named individual identifiers.

Regular Expression	**No. Patterns**	**Freq**
<DT> ∗ <JJ> ∗ <NNS?> ∗ <NNPS?> + <NNS?> ∗	21	5226
<DT> ∗ <JJ> ∗ <NNS?> ∗ <NNPS?> + <NNS?> ∗ <CD> +	8	486
<NNS?> + <NNPS?> ∗ <IN> + <DT> ∗ <NNPS?> +	6	94
Total (above REs)	35	**5806**
Grand Total (named individual identifiers)	1327	**13598**

Three REs extract individual names (table 6). These are similar to the grammars for class extraction except that a compulsory element is the singular or plural proper noun (POS tags NNP or NNPS). The first (i) allows optional adjectives and nouns before one or more proper noun(s) and followed by optional common nouns, or (ii) as (i) but with one or more final numerical elements, and (iii) with optional nouns, one or more proper nouns, a preposition, and one or more proper nouns.

These grammars allow the program to pick out proper names such as 'Northern Hemisphere', 'family Apidae', 'Queen of England', 'William of Orange', 'Mary Shelly 1234', and so on.

Coverage of individual identifiers by the REs are given in table 6, i.e., 5806 out of the cover 43% of all 13598 patterns in the corpus. Coverage is low because a large number of named individual identifiers consist of common noun string patterns which overlap with those of *classes*. As before, a very small number of REs (3) account for a substantial proportion of identifiers (even though only a small number, 35, of the total unique patterns, 1327, are covered).

To distinguish them from class identifiers, I confined REs for individuals to those containing proper nouns. Patterns of noun phrases *without* proper nouns are covered under *class* REs because NP patterns such as optional adjective(s) with common noun(s) are significantly more common in class identifier names (around 46,000 occurrences in about 63,000 class identifiers) in the corpus than in those of named individuals (around 4,000 in about 14,000 identifiers). That is, in class identifier names the proportion is much greater at over 70% compared to around 30% in named individuals. Thus, we decided it was better to include identifier names formed from common nouns only in REs that extract class names.

3.3.3 Extracting object property identifiers

Table 7: OBJECT PROPERTY REGULAR EXPRESSIONS. REs shown account for 59% of object property identifiers.

Regular Expression	**No. Patterns**	**Freq**
<VBZ> + <DT> ∗ <JJS? > ∗ <NNS? > + <IN> ∗	19	3030
<VBZ> ∗ <VBN> + <DT> ∗ <NNS? > ∗ <IN> ∗ <TO> ∗	7	719
<JJ> ∗ <NNS? > + <IN> +	6	571
<VBD> + <DT> ∗ <NNS? > ∗ <IN> ∗ <TO> ∗	10	525
<JJ> ∗ <NNS? > ∗ <VBP> +	9	330
<VB> +	8	128
Total (above REs)	59	**5303**
Grand Total (object property identifiers)	696	**8918**

Similarly, for object property REs, we decided to exclude common noun phrase patterns even though the pattern <JJ> ∗ <NNS? > + was quite common (around 1,300 in about 9,000 identifiers, or 14%). Also, we adapted the REGs to include optional determiners that do not occur in the identifiers in the corpus but do occur in natural English (e.g., 'spins a web'). Determiners are removed in post processing (e.g., 'spins web').

The REs in table 7 pick out object property identifiers that are composed of various types of verbal group, the most common of which are those that begin with the VBZ tag, such as 'is imported by' or 'has great uncle'. Once again, we have not included common noun phrases in the REG, even though they are fairly common, apart from those ending in a preposition such as 'locus of'. The final RE, <VB> +, was included for completeness to extract base form verbs, even though they occured with a low frequency in the corpus.

In table 7, the coverage of the property identifiers that people write is 5303 out of 8918, or **59%**.

Table 8: DATA PROPERTY REGULAR EXPRESSION GRAMMAR Patterns shown account for 0.3% of data property identifiers, others are accounted for by object property REGs.

Regular Expression Grammar	**No. Patterns**	**Freq**
<CD> + <NNS? > +	6	13
Total (data property identifiers)	474	**3501**

3.3.4 Extracting data property identifiers

Only one type of data property identifier is extracted (see RE in table 8), the type that links to a literal integer value. The RE matches a numerical expression containing one or more cardinal numbers (CD POS tag) followed by one or more singular or plural nouns, e.g., 'five eyes.

I have not considered non-numerical data properties such as those that link to string values, e.g., 'has name' linked to the string 'Fred Bloggs'. Indeed, it would be difficult to distinguish these from object properties since many of their POS tag patterns are more-or-less identical (Williams, 2013). Therefore, coverage is very low at 0.3% and we leave it to the ontology developer to distinguish between the two.

3.4 Post-processing identifiers

Identifier names are post-processed for two reasons: (i) to remove spurious ones, and (ii) to make them complient with the controlled natural language of the SWAT ontology editor known as 'OWL Simplified English' (Power, 2012b). Each type is post-processed as follows:

3.4.1 Post-processing class identifiers

The tutorial for editing OWL Simplified English contains the following criteria for construction of a class name:

> A class name is formed from any sequence of the following words: noun, adjective, proper name, preposition, definite article, number, or string. It must not contain a verb, or conjunction, or relative pronoun, or indefinite article. (Power, 2012b)

In addition, a class name cannot begin with the definite article. After extraction, `Txt2ids` adjusts potential class names as follows:

- Convert to lower case.
- Remove 'the' if first word
- Remove all occurrences of: few, many, other, several (POS tag JJ)

- Remove all occurrences of: a, an (POS tag DT)
- Remove all coordinating conjuctions (POS tag CC)
- Convert to singular form

See table 2 for examples of post-processed class names.

3.4.2 Post-processing named individual identifiers

The tutorial for editing OWL Simplified English contains the following criteria for construction of a named individual identifier:

> An individual name must begin either with a proper name, or with the definite article the. This may be followed by one or more of the words permitted in class names (i.e., noun, adjective, proper name, preposition, definite article, untensed verb, number, string). If the opening word is the, at least one such word must be added. (Power, 2012b)

`Txt2ids` post-processes extracted individual names as follows:

- Remove: few, many, other, several (POS tag JJ) if first word
- Remove all occurrences of: a, an (POS tag DT)
- Remove all coordinating conjuctions (POS tag CC)
- Capitalise first word

See table 2 for examples of post-processed individual names.

3.4.3 Post-processing object property identifiers

The tutorial for editing OWL Simplified English contains the following criteria for construction of property identifiers:

> A property name must begin with a verb in the present tense: either is (or plural are) or has (have), or a word declared as a verb. This may be followed by any sequence of the following words: noun, adjective, preposition, untensed verb. If the opening word is an auxiliary (is or has or their plurals), at least one further word must be included. (Power, 2012b)

`Txt2ids` post-processes extracted object property names as follows:

- Convert to lower case.
- Remove: few, many, other, several (POS tag JJ) if first word
- Remove all determiners (POS tag DT)
- Remove all coordinating conjuctions (POS tag CC)

- Convert all nouns to singular form
- If first word is a verb, convert it to 3rd person singular
- If first word is 'has', add second word 'as'
- If first word is a common noun, add 'has as' before it
- If first word is NOT a noun or verb, add 'is' before it

See table 2 for examples of post-processed object property names.

3.4.4 Post-processing data property identifiers

Data property identifier are constructed as object properties (above). `Txt2ids` post-processes extracted data property names as follows:

- Convert to lower case.
- Remove: few, many, other, several (POS tag JJ) if first word
- Remove all determiners (POS tag DT)
- Remove all coordinating conjuctions (POS tag CC)
- Add 'has as' before the phrase
- Add 'in number' after the phrase

See table 2 for examples of post-processed data property names.

4 Sanity test

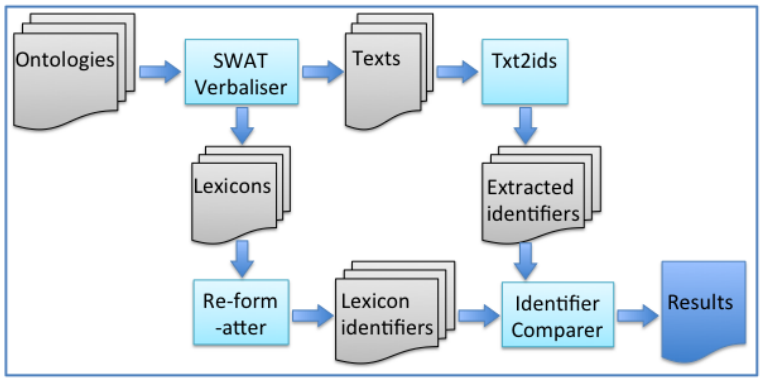

Figure 1: Sanity test architecture

4.1 Method

Whilst it is not a serious evaluation, a sanity test checks whether software performs as expected. As a sanity test here, we compared identifiers extracted from SWAT verbaliser texts with the ones in the SWAT verbaliser lexicons, which resemble closely those written by authors for the ontology. That is, we used `Txt2ids` to extract identifiers from verbalisations produced by the SWAT system from a slightly reduced version[7] of the corpus of ontologies that was used for development of the chunk parser grammars and compared them to the indentifiers used in the ontologies. Of course, this method does not provide `Txt2ids` with a realistic task, because, as discussed earlier, ontology identifiers differ from natural English phrases.

Table 9: Comparing identifiers in the lexicon built from the ontology at ontology.dumontierlab.com with identifiers extracted from the SWAT tools generated text by `Txt2ids`

Generated text	
A double proton emission has as participant a proton. A double proton emission has as participant exactly two protons. A force is a process. A gamma decay has as participant a photon. An internal conversion has as participant an electron. A neutron emission has as participant a neutron. A nuclear reaction is a process. A proton emission has as participant a proton.	
Lexicon identifiers	**Identifiers extracted by `Txt2ids`**
CLASS double proton emission	CLASS double proton emission
CLASS electron	CLASS electron
CLASS force	CLASS force
CLASS gamma decay	CLASS gamma decay
OBJECTPROPERTY has as participant	DATAPROPERTY has protons in number
CLASS internal conversion	CLASS internal conversion
CLASS neutron	OBJECTPROPERTY is process
CLASS neutron emission	CLASS neutron
CLASS nuclear reaction	CLASS neutron emission
CLASS photon	CLASS nuclear reaction
CLASS process	CLASS participant
CLASS proton	CLASS photon
CLASS proton emission	CLASS process
	CLASS proton
	CLASS proton emission

[7]The reduced version of the corpus contains 455 ontologies, 99 fewer than the original. Ontologies were discarded because they contained no identifiers, or identifiers that were mostly numerical, foreign, or nonsense phrases (e.g., 'Id 31005 3453', 'Valorizzazione Di Materia Primo', and 'pers mat assoc ix').

An architecture diagram for the sanity test is depicted in figure 1. The 455 ontologies in the corpus[8] were verbalised by the SWAT verbaliser to produce 455 texts and 455 lexicons. The Txt2ids program extracts identifiers from the texts, i.e., it produced 455 files containing lists of identifiers from the 455 texts. A small 'Re-formatter' program extracts identifiers with their types ('class', 'named individual', 'object property', 'data property' or 'literal') from the lexicons, i.e., it produced 455 files from the 455 lexicons that are in the same format as those extracted by Txt2ids. I developed another module, 'Identifier Comparer' that compares identifiers in two files from *the same ontology:* (i) the file output by Txt2ids and (ii) the file from the lexicon reformatter. The comparison algorithm ignores an initial 'the' in named individuals (so 'Isles of Scilly' matches 'The Isles of Scilly' and in property names, it replaces initial 'has' and 'has as' with 'is' before the comparison (so 'has code' and 'is code' are treated as the same name).

As an example, tabel 9 shows a text generated from an ontology at ontology.dumontierlab.com and the two sets of identifiers that are compared.

4.2 Results

Table 10: Results of a sanity test of `Txt2ids` on identifiers in 455 ontologies (where R is recall, P is precision, and F is the F-measure)

Measure	**Lexicon**	**Txt2ids**	**Lexicon ∩ Txt2ids**	**P**	**R**	**F**
All identifiers	103054	171429	72556	0.42	0.70	0.53
All (excluding type)	103054	171429	78296	0.46	0.76	0.57
Data Properties	525	3311	0	0.00	0.00	0.00
Object Properties	9079	40591	4557	0.11	0.50	0.18
Named Individuals	12056	28695	6931	0.24	0.57	0.34
Classes	78608	101606	61068	0.60	0.78	0.68

Results from the sanity test are shown in table 10 in which recall (R) represents the proportion of identifier names in the ontology lexica that were extracted from the texts by `Txt2ids`. It is calculated by dividing the number in the intersection of identifier names in the lexica and those extracted by `Txt2ids` (fourth column) by the number from the lexicon (second column). Precision (P) is the proportion extracted from the texts that are in the lexica; calculated by dividing the number in the fourth column by the number from `Txt2ids` (third column). The F-measure (F) is shown in the final column; it is calculated as $2.\frac{P.R}{P+R}$. The top two rows show results (i) for all identifiers (exact matches), and (ii) for all disregarding the type (e.g., 'CLASS red

[8] I have also considered re-running the sanity test with different ontologies from the ones used for grammar development in future research.

spider' will match 'NAMEDINDIVIDUAL red spider'). Rows three to six show results for each type of identifier.

4.3 Discussion

Overall, classes have the best results for precision and recall, as expected; this is partly because of the design decision to assign all POS tag patterns with common nouns to type class in preference to other identifier types, even though these patterns were also common in other types. This is demonstrated clearly when we exclude the type from the matching algorithm and nearly 5000 additional matches are found (see row two in table 10).

On the whole, `Txt2ids` tends to overgenerate (it extracts far more identifiers than were present in the lexica). This is particularly the case with object properties where 40,676 were in the lexica, but only 11,900 of these were produced by the program.

Many of the some identifier names that the program suggests that were not in the original ontologies could actually be useful as, e.g., superclass names such as 'spider' as well as 'six-eyed spider'. The class relationships could be inferred by a system such as Third's (Third, 2012).

5 Exploratory Evaluation

To evaluate the system in a realistic manner, I would require a gold standard corpus of naturally-occurring English expository texts (e.g., wikipedia texts) in which potential class, individual, and property identifiers had been reliably annotated. However, 'reliable annotation' is not realistically possible because there is no single, correct answer as to how to annotate potential identifier names in text and even if there were, annotators could disagree on the category of the identifier, e.g., class versus named individual and object vs. data property. In any case, to my knowledge, no such corpus exists and it would require considerable effort to create one.

Consequently, I decided to evaluate the system by comparing identifier names written by a small number of human experts that were given the task of creating an ontologies from short, expository texts, with those extracted from the same texts by `Txt2ids`. Criteria for measuring system performance are then defined as follows:

1. If an identifier name is used by *all* human authors then we would expect the system to suggest the same name (provided, of course, that it is present in the text).

2. If an identifier name suggested by the system matches an identifier name written one or more of the humans, it is considered to be a reasonable suggestion.

Performance criteria are measured by recall and precision. For good performance, we would expect the system to extract a high number of identifiers that are common to all authors (the system should *recall* a high proportion of identifiers in the *intersection* of the sets of those produced by human authors) and we would expect high *precision*, i.e., a high proportion of the identifiers the system extracts should match those produced by at least one human author (the *union* of the sets produced by human authors).

I report here on a small, preliminary study for this kind of evaluation.[9] Three human authors (myself and two other members of the project team) used the SWAT ontology editor to write two small ontologies from two short texts (the first two paragraphs from Wikipedia texts about arthropods and acid).

Table 11: Results of a preliminary evaluation comparing identifier names extracted by `Txt2ids` to those written by human authors (where R is recall, P is precision, and F is the F-measure)

Text	**Human Author(s)**	**Human Written**	**Txt2ids ∩ Human**	**Txt2ids Generated**	**R**	**P**	**F**
1	A1	67	27	98	0.40	0.28	0.33
	A2	114*	84*	98	0.74*	0.86*	0.80*
	A3	67	26	98	0.39	0.27	0.32
	A1∩A2∩A3	21	20	98	**0.95***	0.20	0.33
	A1∪A2∪A3	185	88	98	0.48	**0.90***	0.63
	A1∩A3	29	23	98	**0.79**	0.23	0.36
	A1∪A3	105	30	98	0.29	**0.31**	0.30
2	A1	40	12	48	0.30	0.25	0.27
	A2	39	20	48	0.51	0.42	0.46
	A3	39	11	48	0.28	0.23	0.25
	A1∩A2∩A3	13	10	48	**0.77**	0.21	0.33
	A1∪A2∪A3	76	20	48	0.26	**0.42**	0.32

Table 11 shows overall results for the two texts. It shows results by text (where the texts were each a couple of paragraphs from Wikipedia entries for arthropods, in the case of text 1, and acid, in the case of text 2). The second column shows the ID of the human author, the third column the number of identifiers written by the human(s), the third is the intersection of the set of identifiers written by the human(s) with that produced by `Txt2ids` (ones that were completely identical, including the type), the fourth column shows the total number produced by `Txt2ids`, the fifth column shows recall (the proportion of the ones written by the human author that were also produced by `Txt2ids`), the sixth shows precision (the proportion of the

[9] A larger study will be conducted as future research with ontology expert and non-expert participants.

ones produced by `Txt2ids` that were also human-written), and the final column shows the F-measure, calculated as $2.\frac{P.R}{P+R}$, as before. Where author ID is given as A2∩A2∩A3, it means identifier names produced by all three human authors, or the intersection of the three sets; where it is A1∪A2∪A3, it means identifiers produced by one or more of the authors, or the union of the sets.

With text 1, A2 used `Txt2ids` to suggest identifier names as well as his/her own, resulting in an unusually high number of identifiers, a high degree of overlap between this author's identifiers and the system's and abnormally high precision and recall values (marked * in the table). Therefore, I calculated results for A1 and A2 only in the final two rows for text 1, giving more realistic figures. `Txt2ids`'s recall of names that both authors chose is fairly high (0.79) whilst its precision in suggesting names that one or more of the authors chose is lower (0.31).

With text 2, none of the authors used `Txt2ids` to suggest identifiers names. Values for recall of names that all three authors chose is again fairly high (0.77) whilst precision amongst names than one or more authors chose is again lower (0.42).

Overall, `Txt2ids` suggests substantial numbers of identifier names that would be useful to ontology authors, including some that they may not have thought of themselves. If precision amongst the union of authors' identifier names had been closer to 50%, I would have been more satisfied with its performance. I leave it for future work to investigate whether the relatively high numbers of useless suggestions is annoying or counterproductive.

6 Related work

6.1 Term extractor systems

Existing ontology identifier extractors (Navigli and Velardi, 2004; Maynard, Li, and Peters, 2008) have a different purpose from `Txt2ids` in that they are modules of ontology population systems that aim to build ontologies entirely automatically from text rather than act as tools to aid ontology authors. As such, the identifier names that they produce are actually used in the ontology rather than being mere suggestions that a user can accept or reject. Thus, a higher degree of conformity with a model of what a 'correct' identifier name should be is required.

Navigli and Velardi's OntoLearn system is more sophisticated than `Txt2ids` in that it not only extracts terms but also filters them, interprets them semantically, arranges them in hierarchies, and uses them to update an existing ontology (e.g., WordNet). Its input is not a single text, but collections of documents. It was not evaluated as a stand-alone component but as part of an overall concept-heirarchy-building task that attempted to reconstruct wikipedia's taxonomy from wikipedia texts. Terms are extracted from in-

put texts by parsing them and extracting noun phrases. The extracted noun phrases are filtered by applying one metric that measures whether the phrase is more common in the relevant domain than in a collection of domains and another that measures whether the phrase is generally agreed on in a wide variety of documents in the relevant domain. Hierarchical subtrees are constructed first by considering simple string-inclusion (e.g., 'car ferry service' is a subtype of 'ferry service') where inclusion is always to the right side of the string and later, and then by semantically disambiguating complex phrases with reference to semantic categories in WordNet. The term extraction part of the system was later developed as a Web application called TermExtractor (Sclano and Velardi, 2007). It was evaluated by attempting to reconstruct three wikipedia domain taxonomies (animals, plants and vehicles) from their corresponding collections of wikipedia texts. Results were precision between 90.9 and 97.0 and recall between 38.3 and 48.7 (Navigli, Velardi, and Faralli, 2011).

A recent comparison of five concept extraction systems rated TermExtractor the worst of five systems on precision scores in a term extraction task if proper names were included, or the third best if they were omitted (Piao et al., 2010). The best system overall was C-value (Frantzi and Ananiadou, 1999). In future work, we could compare the performance of `Txt2ids` with the same five systems since their implementations have been made available for other researchers[10].

6.2 Comparison with other online systems

I found a number of more general term extraction systems which were available as on-line services for comparison with `Txt2ids`:

6.2.1 Fivefilters.org

Fivefilters.org[11] provides an on-line web service for term extraction. It is described as a free alternative to Yahoo's Term Extraction service using Topia's Term Extraction Python package[12] and simplejson[13], a Python package to boost processing speed with JavaScript Object Notation. Paste in the same spider text as in section 1 and it gives the following output:

> spiders, eyes, insect, spider web, arachnid arthropods, body length, insects, web, silk, ticks, cm, spider, webs, arthropods, arachnid, nets, ground, use, group, acarina, ways, mites, arachnids, attack, prey, legs, body, kinds, inches, others, wings, thread, mm, hunt, holes, length

[10] Java Automatic Term Extraction library at staffwww.dcs.shef.ac.uk/people/Z.Zhang/resource.html
[11] http://fivefilters.org/term-extraction/
[12] http://pypi.python.org/pypi/topia.termextract/
[13] http://pypi.python.org/pypi/simplejson/

A limitation is that it only extracts nouns and NPs. It orders the results by frequency. Notice that it does not distinguish between singular 'spider' and plural 'spiders'. It extracts one term that `Txt2ids` does not (cm) and three part-terms (arthropod, body, length) and two terms that `Txt2ids` recognises as property names (attacks, uses); while `Txt2ids` extracts an additional six multi-word class names (group in arachnid, hole in the ground, kind of spider, prey in way, thread of silk) and an additional 21 property names (see table 1). Fivefilters could be of potential use to ontology developers. `Txt2ids` performs slightly better.

6.2.2 AchemyAPI's Keyword/Terminology Extraction

AchemyAPI[14] provides another free online term extraction service. Their website states 'We employ sophisticated statistical algorithms and natural language processing technology to analyze your data, extracting keywords that can be utilized to index content' but provides no further information. For the spider text, it finds only 14 terms:

> Tags (14) spiders, insect, Most spiders, arachnids, arachnid arthropods, body length, biggest group, silk, spider web, Acarina, ticks, mites, grab, webs

As before, a limitation is that it finds only NPs.

6.2.3 Yahoo's Content Analysis

Yahoo claim that 'Our recently released Content Analysis Web Service detects entities/concepts, categories, and relationships within unstructured content'. However, I couldn't find a way to try it out.

6.2.4 TerMine

The National Centre for Text Mining's term extraction program is called TerMine[15]. Its output from the spider text is:

> body length (rank 1, score 1), spider web (rank 1, score 1), big group (rank 1, score 1), arachnid arthropod (rank 1, score 1)

6.2.5 Translated Labs.

Translated Labs.'s on-line term extractor[16] produces the following output from the spider text:

> Top 7 terms: spiders 66%, arachnid arthropods 53%, spider web 53%, spiders hide 51%, insect 49%, mites 48%, arachnids 48%,

[14] http://www.alchemyapi.com/api/keyword/
[15] http://www.nactem.ac.uk/software/termine/#form
[16] http://labs.translated.net/terminology-extraction/

However, this application is not attempting to return all the terms in the text but only the ones that occur more frequently in the input text than in a corpus of one million words of English (the classic term extraction task). Results are given as estimates of the relevance of the terms in the document, thus the term ' arachnid anthropods' scores high even though it only occurs once in the text because it is rare in the 1m-word corpus.

6.2.6 Ultimate Research Assistant

Ultimate Research Assistant's information extraction engine on-line web service[17] Setting 'amount of concepts' to maximum and allowing single-word concepts, it produces the output:

> Spiders, silk, eyes, throw, arachnid arthropods, eight legs, six eyes, eight eyes, four eyes, biggest group, arachnids, Acarina, mites, ticks, small, mm, large, 10cm, inches, body length, spiders eat insects, etc, catch, insects count=1, prey, several ways, build, spider web, thread, kinds, spiders hide, holes, ground, run, grab, walks, Others, web, nets, passing insects, even go, attack, webs, wings, jump quite, hunt, sneaking close, jumping

7 Conclusions

I have described a natural language processing system that recognises words and phrases in English text that could serve as ontology identifier names. The input to the program is unrestricted English text in a plain text file and the output is a file containing list of suggestions for identifier names classified into (i) classes, (ii) individuals, (iii) object properties, and (iv) data properties. As well as being of practical use as a tool in an ontology authoring system, it also functions as a theoretical model of the syntactic organisation of identifier names.

Regular expressions for this system were designed from an analysis of part-of-speech patterns in identifier names in a corpus of over 500 ontologies. Since ontology identifier names have syntactic structures that differ from natural English in consistent ways, the regular expressions could be adapted accordingly. Resulting extracted phrases were post-processed to comply with the structure of OWL Simplified English (Power, 2012a).

A system sanity test achieved acceptable results when comparing identifiers extracted by `Txt2ids` (from texts generated from the ontologies) with the original identifiers from the same corpus. Although `Txt2ids` tends to over-generate, many of the additional identifiers seem to be reasonable suggestions.

[17] http://www.ultimate-research-assistant.com/oem/informationextractorservice.htm

To assist in the design of a future system evaluation, a pilot study was conducted in which identifier names extracted by `Txt2ids` from short, expository texts compared favourably with those created by human users when building ontologies from the same texts.

The system has been deployed in an ontology editor developed for the `SWAT` project.

References

Bird, Steven, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly, Beijing.

Frantzi, K. and S. Ananiadou. 1999. The C-value / NC-value domain independent method for multi-word term extraction. *Journal of Natural Language Processing*, 6(3):145–179.

Maedche, Alexander and Steffen Staab. 2000. Mining ontologies from text. In *Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management*, EKAW '00, pages 189–202, London, UK, UK. Springer-Verlag.

Maynard, Diana, Yaoyong Li, and Wim Peters. 2008. NLP techniques for term extraction and ontology population. In *Proceeding of the 2008 conference on Ontology Learning and Population: Bridging the Gap between Text and Knowledge*, pages 107–127, Amsterdam, The Netherlands, The Netherlands. IOS Press.

Navigli, R., P. Velardi, and S. Faralli. 2011. A graph-based algorithm for inducing lexical taxonomies from scratch. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011), Barcelona, Spain, July 19-22nd*, pages 1872–1877.

Navigli, Roberto and Paola Velardi. 2004. Learning domain ontologies from document warehouses and dedicated web sites. *Computational Linguistics*, 30:151–179.

Piao, Scott, Jamie Forth, Ricardo Gacitua, Jon Whittle, and Geraint Wiggins. 2010. Evaluating tools for automatic concept extraction: a case study from the musicology domain. In *Digital Futures 2010*, pages 16–18, Nottingham, U.K.

Power, Richard. 2010a. Analysis of identifiers and labels in ontologies. Technical Report Unpublished.

Power, Richard. 2010b. Complexity assumptions in ontology verbalisation. In *48th Annual Meeting of the Association for Computational Linguistics (ACL 2010)*, Uppsala, Sweden.

Power, Richard. 2012a. Owl simplified english: a finite-state language for ontology editing. In T. Kuhn and N.E. Fuchs, editors, *Proceedings of the Third International Workshop on Controlled Natural Language, Zurich*, pages 44–60, Heidelberg. Springer.

Power, Richard. 2012b. SWAT Editing Tool: a tutorial. Technical Report Unpublished.

Santorini, Beatrice. 1990. Part-of-speech tagging guidelines for the penn treebank project. Technical Report MS-CIS-90-47, Department of Computer and Information Science, University of Pennsylvania.

Sclano, Francesco and Paola Velardi. 2007. Termextractor: a web application to learn the common terminology of interest groups and research communities. In *Terminologie et Intelligence Artificielle, TIA-2007*, Sophia Antipolis, France.

Stevens, Robert, James Malone, Sandra Williams, and Richard Power. 2010. Automating class definitions from owl to english. In *Bio-Ontologies 2010: Semantic Applications in Life Sciences SIG at 18th Annual International conference on Intelligent Systems for Molecular Biology (ISMB 2010),*, Boston, USA.

Third, Allan. 2012. Hidden semantics: what can we learn from the names in an ontology? In *INLG 2012 Proceedings of the Seventh International Natural Language Generation Conference*, pages 67–75, Utica, IL, May. Association for Computational Linguistics.

Toutanova, Kristina, Dan Klein, Christopher Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of HLT-NAACL 2003*, pages 252–259.

Williams, Sandra. 2013. An analysis of POS tag patterns in ontology identifiers and labels. Technical Report TR2013/02, Department of Computing, The Open University, U.K.

Zhang, Ziqi, Jose Iria, Christopher Brewster, and Fabio Ciravegna. 2008. A comparative evaluation of term recognition algorithms. In *Proceedings of the sixth international conference on Language Resources and Evaluation, LREC-2008*, pages 2108–2113, Marrakech, Morocco.