SPRAT: a tool for automatic semantic pattern-based ontology population

Diana Maynard, Adam Funk, and Wim Peters

Department of Computer Science, University of Sheffield
Regent Court, 211 Portobello Street, Sheffield, UK
diana@dcs.shef.ac.uk

Abstract. Ontology generation and population is a crucial part of knowledge base construction and maintenance that enables us to relate text to ontologies, providing a rich and customised ontology related to the data and domain with which we are concerned. SPRAT combines aspects from traditional named entity recognition, ontology-based information extraction and relation extraction, in order to identify patterns for the extraction of a variety of entity types and relations between them, and to re-engineer them into concepts and instances in an ontology. When augmented with richer knowledge such as WordNet semantic categories and terminological information, the results are greatly improved. SPRAT can either modify an existing ontology or create a new ontology from scratch, which is specific to the corpus of texts processed. Preliminary results are very promising, although more refinement of the patterns is still necessary.

1 Introduction

Ontology generation and population is a crucial part of knowledge base construction and maintenance that enables us to relate text to ontologies, providing on the one hand a customised ontology related to the data and domain with which we are concerned, and on the other hand a richer ontology which can be used for a variety of semantic web-related tasks such as knowledge management, information retrieval, question answering, semantic desktop applications, and so on.

Ontology population is generally performed by means of some kind of ontology-based information extraction (OBIE) [1]. This consists of identifying the key terms in the text (such as named entities and technical terms) and then relating them to concepts in the ontology. Typically, the core information extraction is carried out by linguistic pre-processing (tokenisation, POS tagging etc.), followed by a named entity recognition component, such as a gazetteer and rule-based grammar or machine learning techniques.

The work described here combines aspects from traditional named entity recognition, ontology-based information extraction and relation extraction, in order to identify patterns for the extraction of a variety of entity types and relations between them, and to re-engineer them into concepts and instances via

ontology creation and population. The application is developed in GATE, an architecture for language engineering [2]. In this paper, we first investigate some linguistic patterns for relation finding, and then discuss their implementation in the SPRAT tool, by means of the NEBOnE plugin that we have created in order to perform the ontology editing part. Finally we discuss some evaluation and further work.

2 Patterns for NE recognition

Traditional rule-based Named Entity (NE) recognition and OBIE applications in GATE are based on a set of linguistic patterns which aim to identify the relevant entities in text. These rely largely on gazetteer lists which provide all or part of the entity, or clues to its existence, in combination with linguistic patterns. For example, a typical rule to identify a person's name consists of matching the first name of the person via a gazetteer entry (e.g. *John*), followed by an unknown proper noun (e.g. *Smith*, which is POS-tagged as a proper noun).

However, identifying ontological concepts and/or relations requires a slightly different strategy. While we can still make use of known terms (either via a gazetteer or by accessing the class, instance and property labels in an existing ontology), this is often not sufficient for a variety of reasons:

- the concept may not be in the ontology already;
- the concept may exist in the ontology only as a synonym or linguistic variation (singular instead of plural, for example);
- the concept may be ambiguous;
- only a superclass of the concept may exist in the ontology.

We therefore need to make more use of linguistic patterns and also contextual clues, rather than relying on gazetteer lists. We have identified three sets of patterns which can help us identify concepts, instances and properties to extend the ontology: the well-known Hearst patterns (Section 2.1), the Lexico-Syntactic Patterns developed in the NeOn project[1] corresponding to Ontology Design Patterns (Section 2.2), and some new contextual patterns defined by us (Section 2.3).

2.1 Hearst patterns

The Hearst patterns are a set of lexico-syntactic patterns that indicate hyponymic relations [3], and have been widely used by other researchers. Typically they achieve a very high level of precision, but quite low recall: in other words, they are very accurate but only cover a small subset of the possible patterns for finding hyponyms and hypernyms. The patterns can be described by the following rules, where NP stands for a Noun Phrase and the regular expression symbols have their usual meanings[2]:

[1] http://www.neon-project.org

[2] `()` for grouping; `|` for disjunction; `*`, `+`, and `?` for iteration.

- `such NP as (NP,)* (or|and) NP`
  Example: ... works by such *authors* as *Herrick*, *Goldsmith*, and *Shakespeare*.
- `NP (,NP)* (,)? (or|and) (other|another) NP`
  Example: *Bruises*, *wounds*, or other *injuries*...
- `NP (,)? (including|especially) (NP,)* (or|and) NP`
  Example: All *common-law countries*, including *Canada* and *England*...

Hearst actually defined five different patterns, but we have condensed some of them into a single rule. Also, where Hearst defines the relations as hyponym-hypernym, we need to be more specific when translating this to an ontology, as they could represent either instance-class or subclass-superclass relations. To make this distinction, we tested various methods. In principle, POS-tagging should be sufficient, since proper nouns generally indicate instances, but our tagger mistags capitalised common nouns (at the beginning of sentences) as proper nouns frequently enough that we cannot rely on it for this purpose. We also looked at the presence or absence of a determiner preceding the noun (since proper nouns in English rarely have determiners) and whether the noun is singular or plural, but this still left the problem of the sentence-initial nouns. Finally, we decided to pre-process the text with the named entity recognition application ANNIE, and only consider certain types of named entities (*Person*, *Location*, *Organization*, and potentially some unknown entity types) as candidates for instances; all other NPs are considered to be classes. This gave much better results, occasionally missing an instance but almost never overgenerating.

2.2 Lexico-Syntactic Patterns

The second type of patterns investigated was the set of Lexico-Syntactic Patterns (LSPs) corresponding to Ontology Design Patterns (ODPs) [4]. We implemented a number of these patterns in our application. Some of the more complex relation types we did not include because the functionality does not currently exist either in the GATE ontology API or in NEBONE. For each relation, there are several possible patterns: mostly these are all combined into a single rule in our grammars, but we separate them here for ease of comprehension. The grammars are written in JAPE [5]: further details are discussed in Section 3.1.

In the following rules, `<sub>` and `<super>` are like variable names for the subclasses and superclasses to be generated; `CN` means *class of*, *group of*, etc.; `CATV` is a classification verb[3]; `PUNCT` is punctuation; `NPlist` is a conjoined list of NPs ("X, Y and Z").

1. Subclass rules
   - `NP<sub> be NP<super>`
   - `NPlist<sub> be CN NP<super>`
   - `NPlist<sub> (group (in|into|as) | (fall into) | (belong to)) [CN] NP<super>`

[3] E.g., *classify in/into*, *comprise*, *contain*, *compose (of)*, *group in/into*, *divide in/into*, *fall in/into*, *belong (to)*.

   - `NP<super> CATV CV? CN? PUNCT? NPlist<sub>`

   Example: *Frogs* and *toads* are kinds of *amphibian*.
2. Equivalence rules
   - `NP<class> be (the same as|equivalent to|equal to|like) NP<class>`
   - `NP<class> (call | denominate | (designate by|as) | name) NP<class>` (where the verb is passive)
   - `NP<class> have (the same|equal) (characteristic | feature | attribute | quality | property) as NP<class>`

   Example: *Poison dart frogs* are also called *poison arrow frogs*.
3. Properties
   - `NP<class> have NP<property>`
   - `NP<instance> have NP <property>`

   Example: *Birds* have *feathers*.

While these patterns are quite productive (for example `X is a Y`), most of them are potentially ambiguous and susceptible to overgeneration. For example, in the following sentence:

> Mistakenly, some artists and writers have penguins based at the North Pole.

the patterns produced the inference that *writers have penguins*, recognising *penguin* as a property of *writer*. Clearly it is ludicrous that every expression of the form *X has Y* should result in the relation *Y is a property of X*. The difficulty is deciding where to draw the line between acceptable patterns and those that just overgenerate. To start with, we took the simple patterns which generated new basic instances, subclasses and properties. After an initial evaluation, however, we found that these overgenerated quite considerably, so we refined them by adding some restrictions using semantic categories from WordNet. This is discussed in more detail in Section 6.

2.3 Contextual patterns

We also defined a set of rules designed to make use of contextual information in the text about known entities already existing in the ontology (unlike the ODP lexico-syntactic patterns which assume no previous ontological information is present). These rules are used in conjunction with the OntoRootGazetteer plugin in GATE, which enables any morphological variant of any class, instance or label in the ontology to be matched with (any morphological variant of) any word or words in the text. Which elements from the ontology are to be considered (e.g., whether to include properties, and if so which ones) is determined in advance by the user when setting up the application. Initially we use the following rules to find new classes and instances:

1. **Add a new subclass**: `(Adj|N) NP<class>` → `NP<subclass>`.
   This matches a class name aready in the ontology preceded by an adjective or noun, such as adjective preceding a known type of fish, which we assume is a

more specific type. For example, when we encounter the phrase ... Japanese flounder... in a text and *flounder* is already in the ontology, we add *Japanese flounder* as a subclass of *flounder*.

2. **Add a new class** (a more generic version of the Hearst patterns). Here we postulate that an unknown entity amidst a list of known entities is likely to be also an entity of the same type. For example, if we have a list of classes of fish, and there is an unknown noun phrase in amongst the list, we can presume that this is also a class of fish. To decide where to add this new class in the ontology, we can look for the Most Specific Common Abstraction (MSCA) of all the other items in the list (i.e. the lowest common superclass of all the classes in the list) and add the new entity as a subclass of this class. However, this has not currently been implemented due to the complexities of implementation in NEBOnE, but is planned for the future. Currently therefore, we just add it as a new subclass of *Thing* (top level) and leave it to the user to move it to a more appropriate place.
   Example: *Hornsharks*, *leopard sharks* and *catsharks* can survive in aquarium conditions for up to a year or more.
   where *hornshark* and *leopard shark* are classes in the ontology and *catshark* is unknown, so we can recognise *catshark* as a subclass with the same parent as that of *hornshark* and *leopard shark*, in this case *shark*.
3. **Add an alternative name as a synonym**: a name followed by an alternative name in brackets is a very common pattern in some kinds of text. For example in texts about flora and fauna we often get the common name followed by the Latin name in brackets, as in the following sentence:
   Example: *Mummichogs* (*Fundulus heteroclitus*) were the most common single prey item.
   If we know that one of the two NPs is a class or instance in the ontology, we can predict fairly accurately that the other NP is a synonym.

3 SPRAT application

SPRAT (Semantic Pattern Recognition and Annotation Tool) is composed of a number of GATE components: some linguistic pre-processing followed by a set of gazetteer lists and the JAPE grammars described above. The components are as follows:

- Tokeniser: divides the text into tokens
- Sentence Splitter: divides the text into sentences
- POS-Tagger: adds part-of-speech information to tokens
- Morphological Analyser: adds morphological information (root, lemma etc.) to tokens
- NP chunker: divides the text into noun phrase chunks
- Gazetteers: looks up various items in lists
- OntoRootGazetteer (optional): looks up items from the ontology and matches them with the text, based on root forms
- JAPE transducers: annotates text and adds new items to the ontology

The application can either create an ontology from scratch, or modify an existing ontology. The ontology must be loaded with the application (in the former case, a blank ontology is loaded; in the latter, the ontology to be modified) and referenced by the grammar via the runtime parameter. The ontology used is the same one for the whole corpus: this means that if a number of documents are to be processed, the same ontology will be modified. If this is not the desired behaviour, then there are two options:

1. A separate corpus is created for each document or group of documents corresponding to a single output ontology. The application must be run separately for each corpus.
2. A processing resource can be added to the application that clears the ontology before re-running on the next document. This of course requires that the ontology is saved at the end of the application, after processing each document.

3.1 Implementation of patterns

The patterns are implemented in GATE as JAPE rules. On the left hand side (LHS) of the rule is the pattern to be annotated. This consists of a number of pre-existing annotations which have been created as a result of pre-processing components (such as POS tagging, gazetteer lookup and so on) and earlier JAPE rules. The right hand side (RHS) of the rule invokes NEBOnE and creates the new items in the ontology, as well as adding annotations to the document itself. This part of the rule first gets the relevant information from the annotations (using the labels assignFigureed on the LHS of the rule), then adds the new information to the ontology and finally adds annotations to the entities in the document. NEBOnE is responsible also for ensuring that the resulting changes to the ontology are wellformed: this is described in more detail in Section 3.2. Figure 1 shows a screenshot from GATE of an ontology created from a document about sharks.

3.2 NEBOnE

The SPRAT application uses the specially developed NEBOnE plugin for GATE in order to generate the changes to the ontology. NEBOnE (Named Entity Based ONtology Editing) is an implementation for processing natural language text and manipulating an ontology. It is derived from the CLOnE plugin [6] for GATE.

In CLOnE, input sentences are analysed deterministically and compositionally with respect to a given ontology, which the software consults in order to interpret the input semantics. CLOnE allows users to design, create, and manage information without knowledge of complicated standards (such as XML, RDF and OWL) or ontology engineering tools. It is implemented as a simplified natural language processor that allows the specification of logical data for semantic knowledge technology purposes in normal language, but with high accuracy and reliability. The components are based on GATE's existing tools for

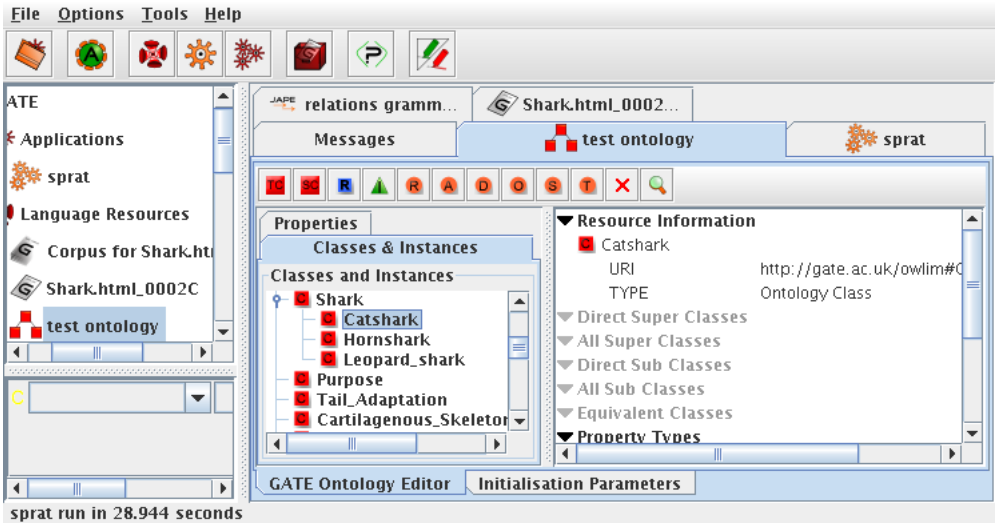

Fig. 1. Generated ontology in GATE

information extraction and NLP natural language processing [7, 8]. Because the parsing process is deterministic, accuracy is not an issue: as long as the user specifies their input in correct controlled language, the system always produces correct output.

Because CLOnE was designed to be used with Controlled Language textual input, it is quite restricted in the patterns it can process and in how it generates the ontological data from them. We therefore developed NEBOnE in order to deal specifically with free text input. As its name suggests, it is based on named entity recognition rather than a restricted set of keywords and noun phrases. For example, in CLOnE, the only way to derive a new subclass is by using a specific pattern containing the restricted keywords `type of` followed by the name of the class, e.g., "a dog is a type of animal". In free text, however, there are many ways in which a subclass could be stated, e.g., "animals such as dogs" or "dogs are animals", and so on (as described in Section 2); the use of a controlled language avoids ambiguity between syntactic structures. CLOnE also imposes strict rules on the order of input sentences and the creation of resources in the ontology. For example, the function to create a subclass required the superclass to exist already, and the function to create an instance made the same stipulation about the new instance's class. We cannot avoid this problem in NEBOnE: this is the sacrifice made for the gain in flexibility of input, which is essential with real world texts.

NEBOnE is based on the same underlying principles as CLOnE and is realised as another GATE plugin. The idea behind NEBOnE is that once a text has been annotated using Named Entity recognition techniques, these annotations can be

used to generate new concepts, instances and properties in the ontology. CLOnE uses so-called *chunks* from the input sentences as candidates for inclusion in the ontology as classes, instances and properties: these are noun phrases previously created by a chunker in GATE. In NEBOnE, however, a *chunk* can be any annotation previously created, and does not need to correspond to a noun phrase, thereby ensuring a great deal more flexibility. When the NEBOnE plugin is installed, actions concerning the ontology are implemented on the RHS of JAPE rules, such as adding or deleting new classes, instances, subclasses, properties and so on.

If an item is selected for addition to the ontology as a new class, NEBOnE first checks to see whether the item in question already exists in the ontology: if it already exists in the place where it is scheduled to be added, NEBOnE will do nothing. If the item exists as a class elsewhere in the ontology, NEBOnE will add the new class (because it supports multiple inheritance). If the requested parent class and subclass both exist and are class names, NEBOnE will make the second a subclass of the first and print a message. If either is already an instance, or the parent class does not exist yet, NEBOnE will print a warning message.

Similarly for an instance, if it exists elsewhere as an instance, NEBOnE will add the new instance but generate a notification message. If the item already exists as a class, and an instance of the same name is to be added, or vice versa, then NEBOnE will not generate the new instance/class and will produce a warning message. Thus NEBOnE ensures consistency in the ontology, avoiding the need to run a checker after the ontology has been modified. A user can of course choose to ignore any potential inconsistencies, by checking the generated messages and then manually adding any offending items or making other changes to the ontology.

3.3 Implementation of NEBOnE

Once the text has been pre-processed, a JAPE transducer processes each sentence in the input text and manipulates the ontology appropriately. This PR refers to the contents of the ontology in order to analyse the input sentences and check for errors; some syntactically identical sentences may have different results if they refer to existing classes, existing instances, or non-existent names, for example.

The *canonical* feature—from which the name of a new class or instance is derived—is the concatenation of the string values of the tokens and underscores for the space-tokens (which can represent literal spaces, tabs or newlines), or the token lemmas.

The Java code that tests chunks in the input text against existing classes and instances in the ontology returns a match if any of the three features of the chunk (canonical, root or string) is case-insensitively equal to any of those features of an existing class or instance; for example, the chunks `'multiword expressions'` and `Multiword expressions` match each other, although the class name in the ontology varies according to which one is first used to create the class.

Unlike CLOnE, NEBOnE's functions will create classes and superclasses as required in order to accommodate instances and subclasses, respectively; it does not require every class to be be explicitly before it is used. The NEBOnE library does, however, reject function calls that would otherwise try to create an instance with the same name as an existing class, or the other way round.

4 Evaluation

We evaluated the accuracy of the lexical patterns using a corpus of 25 randomly selected wikipedia articles about animals. We ran SPRAT and examined the results in some detail. In total, SPRAT generated 201 classes, 21 instances, and 98 synonyms, and 107 other properties. Table 1 shows the results for each type. Note that, unlike in traditional named entity recognition evaluation, we use a strict method of scoring where a partially correct response, i.e. one where the span of the extracted entity is too short or too long, is considered as incorrect. This is because for ontology population, having an incorrect span is generally a more serious error than in named entity recognition.

Relation	Total Extracted	Correct	Precision
Subclass	163	79	48.5%
Instance	21	10	47.6%
Synonym	98	47	48.0%
Property	107	24	22.4%

Table 1. Results of relation extraction on 25 wikipedia documents

4.1 Subclass Relations

In total, 163 subclass relations were generated, of which 79 were correct (48%). However, 15 of these were not really useful classifications: e.g., *turtle* as subclass of *local creature* makes sense only in a very specific context. Of the subclass relations, 69 were found by the Hearst patterns, of which 58 were correct (84%). Of the incorrect relations generated, most contained at least one correct subclass out of a list. For example, in the phrase "by disturbing the natural state of pasture, sheep and other livestock..." both *natural state of pasture* and *sheep* were recognised as subclasses of *livestock*, of which the former is incorrect but the latter is correct. Some refinement of the rules (for example, avoiding NPs containing *of*) could help improve the results. The remaining patterns accounted for 94 of the subclass relations, of which 21 were correct (22%).

We experimented also with restricting possible classes and subclasses to terms found by TermRaider, a term selection algorithm based on linguistic filtering and tf-idf scoring. If we narrow the results to match only subclasses which are

also terms, this improves the precision but lower the recall a little. Adjusting TermRaider's parameters to be a little more flexible with patterns should improve the recall, however. For subclass relations which are also terms, we find 50 occurrences, of which 31 are correct (62%). Of these, the Hearst patterns are almost entirely correct, though lacking a little in recall (95% correct, but only 20 are found in total), while the patterns found by the other rules are 40% correct, which is double the previous score, although only 30 are found in total.

4.2 Instances

The recognition of instances, on the other hand, was quite low in recall though with the same precision as that of subclasses. In total, 21 instances were found, of which 10 were correct. Many of the incorrect results were either as a result of erroneous named entity recognition (e.g. *Harmonia axyridis Pupal stage eggs Coccinellidae* was wrongly extracted as a named entity) or due to instances which are somewhat irrelevant (e.g. *San Francisco Bay* was extracted as an instance of *principal area* in the phrase "some of the principal areas are San Francisco Bay, Richardson Bay, Tomales Bay and Humboldt", which is factually correct but not useful to extract). The main reason for missing instances is that they were wrongly extracted either as subclasses or synonyms.

4.3 Synonyms

98 synonyms were found, of which 47 were correct (again 48%). Of the incorrect responses, some of the so-called synonyms were actually instances. Many of the incorrect synonyms were due to the relationship between the two items only holding in a very particular context. For example, in the sentence "A modified slit called a spiracle is located just behind the eye", the system identifies *modified slit* as a synonym of *spiracle*. Clearly not all modified slits are spiracles: in fact, *spiracle* should be extracted as a subclass of *modified slit*. Since there are many of these examples, we need to look more closely at the rules governing these.

4.4 Properties

Aside from synonyms, the system found 107 class and instance properties. Of these, we analysed the first 50, of which 17 were correct, 13 were incorrect, 5 correct but irrelevant, and 15 were correct but had the wrong span of either the domain or range.

5 Discussion

We can see that the patterns implemented are far from foolproof, since unlike with a controlled language such as CLOnE, we cannot rely on a one-to-one correspondence between a simple syntactic structure and its semantics. First we have the problem of overgeneration. Already, we have discarded some potential

patterns (such as some of the LSPs) that we consider to generate too many errors. Further refinement is still necessary here, either to remove other patterns or to reimplement them in a different way.

One of the main causes of overgeneration is caused by the span of the noun phrase describing the concept to be added to the ontology. We have experimented with different possibilities. A larger span provides finer distinctions and thus better classes, but overgenerates considerably, while a smaller span produces more general classes but better accuracy (does not overgenerate so much). For example, in the following sentence:

> The individuals communicate using a variety of clicks, whistles and other vocalizations.

variety of click is recognised as a subclass of *vocalization*. While this is technically correct, a better interpretation would be simply the subclass *click*.

On the other hand, if we reduce the span of the noun phrase, we risk losing some important information. For example, in the sentence:

> Mygalomorph and Mesothelae spiders have two pairs of book lungs filled with haemolymph

if we do not identify the full noun phrase *two pairs of book lungs*, we can end up with a rather uninformative property value. A closer analysis of the spans is needed, which may help identify which patterns require longer spans than others, for example.

Second, lexical patterns tend to be quite ambiguous as to which relations they indicate. For example, `NP have NP` could indicate an object property or a datatype property relationship. Often, further context is also crucial. For example, in the sentence "Both African males and females have external tusks", it is not very useful to extract the concept *females* with the property *have external tusks* unless you know that *females* actually refers to female African elephants. To extract this information would require also coreference matching.

6 Pattern refinement

In this section, we describe some of the refinements we have made to the patterns as a result of our preliminary evaluation. First, we have experimented with the incorporation of deeper semantic relations using semantic classes from VerbNet[9] and WordNet[10] in order to look for verbal patterns connecting terms in a sentence, and to restrict the kinds of noun phrase extracted. We make use of the ANNIC plugin in GATE [11] to search for frequently occurring annotation patterns. We aim not only to reduce the number of errors, but also to eliminate the kind of general relations which while not incorrect, are not very useful. For example, knowing that a turtle is a local creature is not of much interest unless more contextual information is provided (i.e. in which region it is local).

We are also investigating the use of TermRaider for restricting the number of candidates for extraction. Finally, we plan to incorporate combinations of

Hearst patterns and statistically derived collocational information, because its combination with lexico-syntactic patterns has proven to improve precision and recall [12]. Integration of a full parser has also been investigated, but discarded on the grounds of speed (full parsing is extremely computationally expensive in this situation). In particular, we found that the sentences in Wikipedia articles, which we have used for training and testing, are quite hard to parse well, because they frequently exhibit a long and complex sentence structure which is highly ambiguous to a parser. This causes not only speed but also accuracy problems.

We took inspiration also from some currently unpublished research carried out at DFKI in the Musing project[4], which looks at deriving T-Box Relations from unstructured texts in German. In this work, attention is focused primarily on deriving relations between parts of German compound nouns, but we can make use of similar restrictions. For example, in their work they might derive from the compound noun "bank manager" that there is a property "has manager" belonging to "bank", and that a "bank manager" is a subclass of "manager".

6.1 Restrictions on Noun Phrases

We prevent certain stop words occurring as part of a noun phrase recognised in the patterns. These stop words are a combination of some words given the wrong grammatical category by the part of speech tagger, and some very common words which we do not want to recognise as adjectives. This list of stop words was determined heuristically and can be augmented as necessary with further iterations of testing.

6.2 Restrictions on Subclass Patterns

We modified the subclass rule `(Adj|N) NP<class>` → `NP<subclass>` from the set of contextual patterns, such that either the superclass must already exist in the ontology as a recognised class, or such that certain semantic restrictions apply. For example, one restriction states that both the proposed subclass and superclass must have the semantic category "animal". For example, this enables us to recognise "carrot weevil" as a subclass of "weevil". This rule in particular has very high accuracy (98%) and only seems to cause errors as a result of incorrect semantic categories from WordNet.

6.3 Restrictions on Properties

One of the most error-prone rules was the Property rule `X has Y` from the Lexico-Syntactic Patterns set, which was clearly far too general. We restricted this to again use semantic categories of WordNet. For example, for animals we can state that X must be an animal and Y must be a body part. This gave much better results (75% accuracy, although low recall). Another restriction is the type

[4] http://www.musing.eu

of thing that can be considered a property. We experimented with restricting the range of the property to the following semantic categories from WordNet: plant, shape, food, substance, object, body, animal, possession, phenomenon, artifact, and found much improved results.

7 Related work

Lexico-syntactic pattern-based ontology population has proven to be reasonably successful for a variety of tasks [13]. The idea of acquiring semantic infomration from texts dates back to the early 1960s with Harris' *distributional hypothesis* [14] and Hirschman and Sager's work in the 1970s [15], which focused on determining sets of sublanguage-specific word classes using syntactic patterns from domain-specific corpora. A detailed description and comparison of lexical and syntactic pattern matching can be found in [16], In particular, research in this area has been used in specific domains such as medicine, where a relatively small number of syntactic structures is often found, for example in patient reports. Here the structures are also quite simple, with short and relatively unambiguous sentences typically found: this makes syntactic pattern matching much easier.

Text2Onto [17] performs synonym extraction on the basis of patterns. It combines machine learning approaches with basic linguistic processing such as tokenisation or lemmatisation and shallow parsing. Since like SPRAT it is based on the GATE framework, it offers flexibility in the choice of algorithms to be applied. Compared with SPRAT, it has a smaller number of lexico-syntactic patterns. On the other hand, it applies additional statistical clustering and parsing for relation extraction All in all, this leads to more data, but not necessarily to an improvement of the resulting ontology in terms of precision.

Within the range of activities required for ontology learning, SPRAT covers a number of intermediate stages in the process of ontology acquisition, namely term recognition and relation extraction. In the initial acquisition stage, it will recognise terms from the corpus only if they participate in any of the patterns. This guarantees termhood only up to a certain extent. Further term filtering results in improved precision. For relation extraction, SPRAT does not make use of a parser. There are many applications that make use of syntactic dependencies e.g. [18, 19]. Our approach differs from this in that our patterns are defined at low levels of syntactic constituency, such as noun phrases, and by means of finite state transducers. Identifying and engineering on the basis of the linguistic building blocks that our relevant for each ontology editing task eliminates the need for a parser. Patterns are encoded locally, i.e. not embedded into a syntactc structure., but bottom-up created by combinations of finite-state patterns. Pattern variations can therefore be easily encoded. This bottom-up approach is much faster and less error-prone than a parser, because it robustly identifies syntactic building blocks rather than complete syntactic parses. Our approach is more in line with the ontology bootstrapping approach advocated in [20].

8 Conclusions and Further Work

Because the innovative character of the paper lies in engineering rather than in research, we need to emphasise that, in this phase, the strength of the approach lies in its fundamental approach to linguistically motivated ontology engineering. An increasing number of atomic ontology editing operations are associated with lexico-syntactic patterns, according to the ontological information these patterns and the participating entities contribute. The flexibility of this association enables us to approach the transformation of linguistic structures into lightweight ontological knowledge in an incremental fashion. Also, the opportunity to incorporate any kind of additional knowledge into the system allows us to experiment with different settings, and use SPRAT as a research platform rather than a black box product. This sets it apart from partial approaches such as Hearst, because it offers a platform to dynamically include new algorithms.

In summary, the SPRAT tool assists the user in the generation and/or population of ontologies from text, using linguistic patterns. We have developed a number of new GATE plugins, including NEBOnE for editing the ontology, and TermRaider for finding new terms. In the first pass, we found that the rules gave good recall, but precision is a little low. After our modifications, we achieved very good precision, but much lower recall. Future work lies therefore in finding some kind of balance between these two.

Acknowledgements

The research is conducted as part of the EU-funded projects NeOn (IST-2005-027595) and Service-Finder (FP7-215876).

References

1. Maynard, D., Li, Y., Peters, W.: NLP Techniques for Term Extraction and Ontology Population. In Buitelaar, P., Cimiano, P., eds.: Bridging the Gap between Text and Knowledge - Selected Contributions to Ontology Learning and Population from Text. IOS Press (2008)
2. Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V.: GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In: Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02). (2002)
3. Hearst, M.A.: Automatic acquisition of hyponyms from large text corpora. In: Conference on Computational Linguistics (COLING'92), Nantes, France, Association for Computational Linguistics (1992)
4. de Cea, G.A., Gómez-Pérez, A., Ponsoda, E.M., Suárez-Figueroa, M.C.: Natural language-based approach for helping in the reuse of ontology design patterns. In: Proceedings of the 16th International Conference on Knowledge Engineering and Knowledge Management Knowledge Patterns (EKAW 2008), Acitrezza, Italy (September 2008)

5. Cunningham, H., Maynard, D., Tablan, V.: JAPE: a Java Annotation Patterns Engine (Second Edition). Research Memorandum CS–00–10, Department of Computer Science, University of Sheffield (November 2000)
6. Funk, A., Tablan, V., Bontcheva, K., Cunningham, H., Davis, B., Handschuh, S.: CLOnE: Controlled Language for Ontology Editing. In: Proceedings of the 6th International Semantic Web Conference (ISWC 2007), Busan, Korea (November 2007)
7. Maynard, D., Tablan, V., Ursu, C., Cunningham, H., Wilks, Y.: Named Entity Recognition from Diverse Text Types. In: Recent Advances in Natural Language Processing 2001 Conference, Tzigov Chark, Bulgaria (2001) 257–274
8. Maynard, D., Tablan, V., Cunningham, H., Ursu, C., Saggion, H., Bontcheva, K., Wilks, Y.: Architectural Elements of Language Engineering Robustness. Journal of Natural Language Engineering – Special Issue on Robust Methods in Analysis of Natural Language Data **8**(2/3) (2002) 257–274
9. Schuler, K.K.: VerbNet: A broad-coverage, comprehensive verb lexicon. PhD thesis, University of Pennsylvania (2005)
10. Fellbaum, C., ed.: WordNet - An Electronic Lexical Database. MIT Press (1998)
11. Aswani, N., Tablan, V., Bontcheva, K., Cunningham, H.: Indexing and Querying Linguistic Metadata and Document Content. In: Proceedings of Fifth International Conference on Recent Advances in Natural Language Processing (RANLP2005), Borovets, Bulgaria (2005)
12. Cederberg, S., Widdows, D.: Using LSA and noun coordination information to improve the precision and recall of automatic hyponymy extraction. In: Proceedings of the 7th conference on Natural language learning at HLT-NAACL, Morristown, NJ (2003) 111–118
13. Etzioni, O., Cafarella, M., Downey, D., Kok, S., Popescu, A., Shaked, T., Soderland, S., Weld, D.S., Yates, A.: Web-scale Information Extraction in KnowItAll. In: Proceedings of WWW-2004. (2004) `http://www.cs.washington.edu/research/knowitall/papers/www-paper.pdf`.
14. Harris, Z.: Mathematical Structures of Language. Wiley (Interscience), New York (1968)
15. Hirschman, L., Grishman, R., Sager, N.: Grammatically based automatic word class formation. Information Processing and Retrieval **11** (1975) 39–57
16. Maynard, D.G.: Term Recognition Using Combined Knowledge Sources. PhD thesis, Manchester Metropolitan University, UK (2000)
17. Cimiano, P., Voelker, J.: Text2Onto - A Framework for Ontology Learning and Data-driven Change Discovery. In: Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB), Alicante, Spain (2005)
18. Cimiano, P., Hartung, M., Ratsch, E.: Learning the appropriate generalization level for relations extracted from the Genia corpus. In: Proc. of the 5th Language Resources and Evaluation Conference (LREC). (2006)
19. Gamallo, P., Gonzalez, M., Agustini, A., Lopes, G., de Lima, V.: Mapping syntactic dependencies onto semantic relations. In: Proc. of the ECAI Workshop on Machine Learning and Natural Language Processing for Ontology Engineering. (2006)
20. Maedche, A.: Ontology Learning for the Semantic Web. Kluwer Academic Publishers, Amsterdam (2002)