User-friendly ontology authoring using a controlled language

Valentin Tablan, Tamara Polajnar, Hamish Cunningham, Kalina Bontcheva

Department of Computer Science
University of Sheffield
Regent Court, 211 Portobello Street
S1 4DP, Sheffield, UK
{V.Tablan, T.Polajnar, H.Cunningham, K.Bontcheva}@dcs.shef.ac.uk

Abstract

In recent years, following the rapid development in the Semantic Web and Knowledge Management research, ontologies have become more in demand in Natural Language Processing. An increasing number of systems use ontologies either internally, for modelling the domain of the application, or as data structures that hold the output resulting from the work of the system, in the form of knowledge bases. While there are many ontology editing tools aimed at expert users, there are very few which are accessible to users wishing to create simple structures without delving into the intricacies of knowledge representation languages. The approach described in this paper allows users to create and edit ontologies simply by using a restricted version of the English language. The controlled language described within is based on an open vocabulary and a restricted set of grammatical constructs. Sentences written in this language unambiguously map into a number of knowledge representation formats including OWL and RDF-S to allow round-trip ontology management.

1. Introduction

Research work in recent years in the fields of Semantic Web (SW) and Knowledge Management (KM) has produced tools that are useful in a variety of contexts and scenarios. Many of these tools require a structuring of the information space most commonly in the form of ontologies. A growing trend is the use of natural language processing (NLP) methods, from the area of Information Extraction, for automatic derivation of semantic meta-data, from existing textual documents, in order to bootstrap the layer of formal knowledge required by Semantic Web tools. Outside of the SW field, ontologies are also a popular method of representing domain information used internally by NLP systems. All these create a need for user-friendly methods of authoring ontological data.

A number of ontology editing tools, such as Protégé (Noy et al., 2001), are available to knowledge engineers; however, these tools require some training and an understanding of underlying representation formalisms. The ontology representation languages such as OWL (Dean et al., 2004) or RDF-S (Lassila and Swick, 1999) are quite complex and descriptive, but not easy to comprehend for non-expert users. Our experience has shown that in many cases where ontologies are used within NLP systems, only a few of the features supported by languages such as OWL are actually used. In most cases the requirements only include representing a taxonomy of classes, being able to define properties that apply to class members, and the ability to describe new instances for the classes and the property values for them; however, in order to assure compatibility, the format used for data interchange needs to comply with the standards. This creates a situation where relatively simple information, i.e. that people would find easy to *put into words*, needs to be represented using an advanced mechanism that requires a deep understanding of the intricacies of the representation formalism.

The approach proposed by this paper bridges that gap by defining a controlled language which, while restricted, still feels natural to people and at the same time is unambiguous and simple enough for the machines to process. This allows non-specialists to create and edit ontological data while insulating them from the complex syntax and semantics of the ontology representation language.

2. Context and Related Work

Human language is the most natural method of communication for people; however, it has very complex structures and a large degree of ambiguity. This makes it difficult to process automatically and machines can currently only extract a limited amount of the information contained within. People are able to understand language by using the context of the discourse or extraneous information such as our world-knowledge, clues like facial expressions, or tone of voice. On the other side of the coin, formal data that is rigidly structured is easily processed by machines but hard and unnatural for people to use.

One way of resolve this dichotomy is by defining controlled languages. A controlled language (CL) is a subset of a natural language which is generally designed to be less complex than the complete language and to include only certain vocabulary terms and grammar rules which are relevant for a specific task. The reduced complexity and the lack of ambiguity makes it "understandable" by machines. Designing a CL requires finding the right balance between power of expression and simplicity, so that, while some linguistic choices are restricted, the resulting language still feels natural to the people using it.

The idea of controlled languages is not new, early controlled languages can trace their roots to 1970's Caterpillar Fundamental English (CFE). CFE was designed to restrict the complexity of the language used (it only had 850 terms) so that the text is unambiguous enough that it can reliably translated automatically into a variety of other languages. Further examples are the Caterpillar Technical English (CTE) which had 70,000 carefully chosen domain-specific terms, the KANT (Kamprath et al., 1998) system developed at Carnegie Mellon University, ClearTalk[1] (Skuce, 2003),

[1] http://www.factguru.com

or Attempto Controlled English which is directly translatable into first order logic (Schwertel, 2000). The approach taken here differs from the previous work in that the grammar is quite small while the vocabulary is quite broad. CTE, KANT, ClearTalk, and Attempto all have more complex grammars and limited vocabularies. The advantage of our approach is that virtually no training is required to master using a grammar which only allows a few constructs.

Given the aim to develop a user-friendly ontology authoring system for NLP research, the choice of using a controlled language to serve that purpose is reinforced by several factors. The previous work in CLs proves that it can be done and provides pointers on how to approach the task. The task of parsing CL input is essentially a language processing task so it can make use of the existing infrastructure for NLP. This also makes it easily customisable by language engineers. While ontology authoring is mainly a task for system developers, situations can be foreseen where the end user will need to perform it as well, e.g. for customising an existing system or in order to manually correct the results of NLP systems. Because the input is text, the step of ontology maintenance will be easier to integrate in the overall work-flow of the system which is designed to work with text anyway.

3. From Text to Ontologies

The controlled language proposed here is modelled to allow maximum expressibility within the smallest set of syntactic structures. The limited number of allowed syntactic sentence structures makes the language easier to learn, and much easier to use than OWL, RDF, or SQL. While the syntactic structure of the sentences is constrained, the vocabulary permitted is unrestricted: apart from a small number of key-phrases that are used to mark phenomena of interest, any terms can be used freely. This allows for the ontologies created to be open-domain.

The possible types of actions are: definition of new classes, creation of hierarchies between classes, definition of object and data-type properties, creation of instances, and setting of property values for instances.

The greatest advantage of this approach is that it requires essentially no training; there are no complicated user interfaces to be learnt, there are no complex formalisms to be understood. The user can simply start from a simple example which shows all of the types of utterances accepted by the system and continue the ontology authoring work by re-using and modifying the provided examples. After the editing is finished, the resulting ontology can then be previewed using a simple ontology viewer implemented for this purpose. Once the output has been validated, the ontology can be saved into a variety of formats including RDF-S and OWL variants.

The language analysis is carried out by a Controlled Language Information Extraction (CLIE) application based on the GATE language processing framework (Cunningham et al., 2002a). The application is a pipeline comprising the GATE English tokeniser, the Hepple part-of-speech tagger, a morphological analyser, a transducer that identifies quoted strings, a list look-up component that recognises useful key-phrases, followed by two more finite-state transducers, one for finding noun phrases and the last one that actually parses the CL text and generates the ontology. All the transducers are implemented using GATE's JAPE pattern matching language (Cunningham et al., 2002b). The work-flow of the application is depicted in Figure 1.

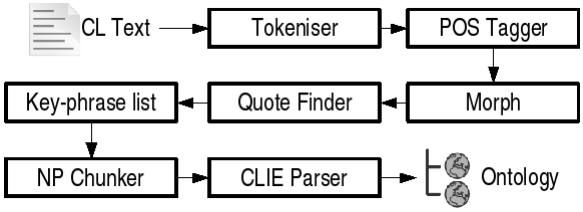

Figure 1: The CLIE application.

```
There are publications and authors.
Papers, articles and books are a type
of publication.  Publications have
authors.  Publications have textual
titles.
Paper 1 is a paper.  John Smith is an
author.  Paper 1 has author with value
John Smith.  Paper 1 has title with
value "A Paper About Something".
```

Figure 2: Example of input text

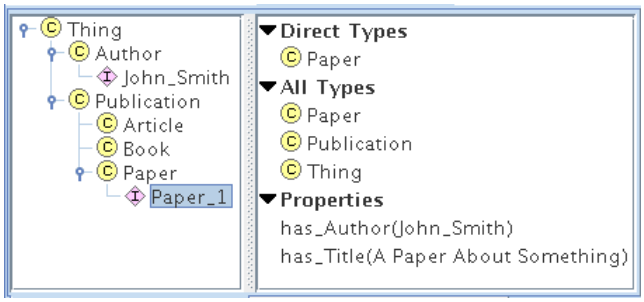

Figure 3: The resulting ontology.

The first three components are standard NLP tools and are used here to pre-process the input text while the quote finder and the NP chunker are simple JAPE transducers. The work of actually parsing CL input is mainly performed by the CLIE transducer together with the key-phrase finder. The key-phrase finder is implemented using the GATE Gazetteer component and has the role of finding and annotating words and phrases that are part of the controlled language. After all of the processing by the components upstream is done, the CLIE parser needs only to look for sentences that contain pre-determined types of key-phrases at given locations inside. The remaining tokens from the sentence that are not part of the key-phrases are used to generate the names of the ontological objects that are created.

An example of valid CL input is presented in Figure 2 while the resulting ontology is shown in Figure 3. The full list of constructs known by the CLIE parser is included in Table 1.

Sentence Pattern	Usage	Example
There are <class>.	Declares new classes	There are publications.
<sub-class> is a type of <super-class>.	Declares a new class as a subclass of an existing class.	Book is a type of publication.
<sub-class>, ..., <sub-class> are a type of <super-class>.	Declares several new classes as subclasses of an existing class.	Books and articles are a type of publication.
<class> (can) have <class>.	Declares a new property linking two classes.	Publications have authors.
<class> (can) have textual <property name>.	Declares a new datatype property of type string.	Publications have textual title.
<instance name> has <instance name>.	Gives a value for a property of an instance. The name of the property is obtained from the domain and range restrictions of the known properties.	Book1 has John Smith.
<instance name> has property <property name> with value <instance name>.	Gives a value for a specified property of an instance. This variant can be used when there are several properties that might apply for the given instance types.	Book1 has author with value John Smith.

Table 1: Controlled language constructs

The Controlled Language IE application (CLIE) employs a deterministic approach, so that each sentence can be parsed in one way only. Allowed sentences are unambiguous, so each sentence type is translated to one type of statement. If parsing fails it will result in a warning and no ontological output.

The use of linguistic analysis allows for small variations in the surface form used to name objects, for instance the use of plurals where it feels appropriate from a linguistic point of view, without affecting the capability of the system to identify different references for the same entity. For instance, in the example provided in Figure 2, the sentence `''There are publications.''` will lead to the creation of a new ontology class called `'Publication'`. The sentence `''Papers, articles and books are a type of publication.''` will create three new ontology classes that are sub-classes of `'Publication'`. The class `'Publication'` is referred to in two different ways – one in plural and capitalised and another one in singular form and all lowercase. The use of the morphological analyser allows the system to recognise that the two words have the same root. As already exemplified above, listing several ontological objects in the same statement is also allowed.

The names of ontological objects (classes, properties and instances) are normalised – first letters are capitalised, spaces are replaced with underscores and the head word in the case of noun phrases is shown un-inflected. If this is undesirable, names can be included in single quotes which will cause them to be used as they appear in the text.

The growing ontology that is generated from the parsed text is also used as a parsing aid. The CLIE parser verifies each statement that it reads for inconsistencies with regard to the data already in the ontology. For example if we were to remove the `''Publications have authors''` sentence from the input, then an error message stating `''Error (hasPropertyValue): Property does not exist: has_Author''` will be issued when the penultimate sentence is reached. The types of the objects used for values of properties are also checked for validity.

In order to provide feedback during editing, a simple ontology viewer (seen in Figure 3) has been implemented. The left panel shows the ontology as a tree containing the class hierarchy and the instances, while the right panel shows details for the node currently selected in the tree.

4. Round-trip Ontology Authoring

The previous section described how one can use CL text to generate a new ontology. The tool described in this paper, however, is a round-trip ontology authoring tool which also allows an existing ontology to be translated into natural language. This is useful for either enriching an existing ontology or for manually correcting the output of a system that saves its results into an ontology.

In order to address this use case, a language generator has been implemented that can produce text starting from an ontology. The generator together with the CLIE application form a round-trip ontology authoring environment: one can start with either an empty ontology or an existing one, generate CL text from it using the generator, modify the text as required and parse the resulting text back into an ontology using the CLIE application. This process can be repeated as necessary until the required result is obtained.

The text generation component has been implemented as a GATE resource. It is configured using an XML file that contains text templates that are instantiated and filled in with values from the ontology. An example of a template that is used to generate the text for declaring new top-level classes is presented in Figure 4. The process of converting an ontology to text has several phases. First the ontology is converted into triples of type <subject, property, object> in

```
<!-- Template for defining top classes -->
<template>
  <in>
    <triple id="t1">
      <property ns="rdf" name="type"/>
      <object ns="owl" name="Class"/>
    </triple>
  </in>
  <out>
    <singular>
      <phrase>There are <ref ref="t1.subject" number="plural"/>. </phrase>
    </singular>
    <plural>
      <phrase>There are <ref ref="t1.subject" number="plural"/>. </phrase>
    </plural>
  </out>
  <ignoreIf>
    <triple id="t2">
      <subject ref="t1.subject"/>
      <property ns="rdfs" name="subClassOf"/>
    </triple>
  </ignoreIf>
</template>
```

Figure 4: Example of a generation template.

the style of RDF. Next all generation templates are used to find triples that match their input specification. For all the triples that have been matched with a template, a grouping process, which finds sets of triples that can be used together to generate a single sentence, is performed. Finally, for each set of triples matched to a template, a sentence is generated and written to the output.

The generation of triples from the input ontology is fairly trivial. First, the class hierarchy is traversed in breadth-first manner to create a list of classes where the super-classes are always mentioned before their corresponding subclasses. For each class in the list, one or more triples are generated that include the declaration of a new class, the link with any super-class through the `rdfs:subClassOf` property and the list of all the instances for the new class. After the processing of the classes and their instances has been finished, the properties are next to be converted. Finally, triples that declare property values for all the instances are generated. At the end of this process, the entire ontology has been converted into a list of triples.

The next step matches the generation templates from the configuration file with triples from the list obtained in the previous step. A generation template (see the example in Figure 4) has three main components: an `in` element containing a list of triple specifications, an `out` element containing phrases that are generated when a successful match has occurred, and an optional `ignoreIf` element that can be used to specify additional triple specifications that cause the match specified in the `in` element to be ignored if the conditions are satisfied.

The `in` part of a template consists of a list of triple specifications, where each triple may have a subject, a property, and an object element. A triple specification should be seen as a restriction – it lists the conditions required for an input triple to match this template. If there is more than one triple included in the `in` element, they should be seen as a conjunction of restrictions: the template will only match when several triples have been found, one for each triple specification included. References from one triple to another are permitted: one can specify for instance that the second triple should have the same `object` as the `subject` of the first triple. The triples in the `in` element are matched in the order they were listed in the template so only backward references are permitted – a triple can only refer to a previously defined one. An example of using references can be seen in the `ignoreIf` element of the template presented in Figure 4.

The `out` section of a template specification describes the text that gets generated when a successful match occurs. It contains phrase templates that have text elements and references to values that were matched from the `in` part. The phrases are divided into singular and plural forms, the plural variants being used when several triples are grouped together and are used to generate a single sentence using a list of ontology objects (e.g. saying "`There are publications and authors.`" will declare two new top classes in the same sentence). The text elements inside a phrase template are simply copied to the output while the reference elements are replaced with the actual values based on the triples that were matched with the specifications from the `in` section. The `out` section of a template can include several phrase templates for each of the `singular` and `plural` sections. These are seen as alternative variants of expressing the same message and are used by rotation. This facility was added to avoid tedious repetitions of similar phrases.

Generation templates can also include an `ignoreIf` section which is similar to the `in` section in the sense that

it also contains a list of triple specifications. References to triples matched in the `in` section are permitted. For a template that contains an `ignoreIf` section, there is an additional step before a match is accepted. If the restrictions from the `in` section are satisfied, then the restrictions from the `ignoreIf` section are also checked. If additional triples are found to satisfy those conditions, then the match is cancelled and the generation template is not applied.

After a generation template has been used to check for matches in the list of triples, the next step groups the matches together into sets that can be expressed together by a single phrase using the plural variant. For this to be possible, the condition is that the only difference between the matches in a set occurs in only one of the references used in the phrase template, i.e. that the singular variants of the phrases that would be generated for each individual match would only differ by one value. If that is the case, then all the matches in the set can be grouped into a plural phrase. Once all the matched triples have been grouped into sets, an output sentence is written for each resulting set.

There are also a special type of generation template that include no `in` restrictions. These can be used to include text into the output that does not depend on any input triples. As the generation templates are applied in the order they were defined in the configuration file, an empty template will be applied after all the normal templates before it have been used.

5. Future Work

The current state of the work described here is that of proof-of-concept prototype. The system can generate ontologies starting from CL text and can also generate CL text starting from an ontology. Support is only provided for basic features of the ontological formalisms – the taxonomy of classes, instances of classes, properties and their values.

Currently only object and string datatype properties are supported. The system could be extended to support more types of datatype properties, such as numbers, dates, etc. This would require adding new constructs to the controlled language that will allow the declaration of such types and functionality for parsing values of these types.

Another possible improvement would be to add more alternatives of expressing the same message; currently only one or at most two constructs are supported for any operation. Adding more variants would make it more likely that the language contains whatever variant that a user might chose thus reducing the number of errors that new users get and minimising the amount of training required for using the system.

While these additions to the language would increase the number of features that are supported and might make the language feel more natural, care should be taken not to make the language too complex. The aim of this work is to simplify the problem up to a point where a simple solution can be applied, while keeping the results useful. A CL that covers all the features of an ontology representation language such as OWL is possible but probably not desirable – as it would be difficult to learn and use. A simple controlled language, such as the one described here, requires essentially no training. When starting from an existing ontology, the generator will convert it into CL text which can then be used as example for adding information to the ontology.

6. Conclusions

The use of ontologies in NLP systems is becoming more and more frequent, a trend encouraged by the rapid developments in the fields of Semantic Web and Knowledge Management. Formal standards for representing ontologies have now been created and ratified by bodies such as the World Wide Web Consortium and are being widely used in applications. These standards, such as RDF, RDF-S or OWL, are very rich in features which leads to very complex authoring environments which can be overwhelming to new users.

This work proposes an alternative to the graphical interfaces in the form of a controlled language that can be used to describe ontologies. Tools for automatically converting from the CL to the ontology and from an ontology to the CL have been implemented forming a text-based ontology authoring environment. As these tools have been integrated with the GATE language engineering platform, they make use of GATE's ontology API for reading and writing external ontologies, which means that standard representation formats are supported.

The use of a text-based interface for authoring ontologies can be advantageous in many scenarios: when it needs to integrate into an existing text-based work-flow, when it needs to run on platforms with limited graphical capabilities (e.g. withing the editor of a wiki-style web application) or when the target users would feel more comfortable with editing text rather than being faced with a complex graphical interface.

7. Acknowledgements

The research for this paper was conducted as part of the European Union Sixth Framework Program projects SEKT (EU IST IP 2003-506826) and PrestoSpace (FP6- 507336).

8. References

H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. 2002a. 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)*.

H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, and C. Ursu. 2002b. *The GATE User Guide*. `http://gate.ac.uk/`.

M. Dean, G. Schreiber, S. Bechhofer, Frank van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. 2004. OWL web ontology language reference. W3C recommendation, W3C, Feb. http://www.w3.org/TR/owl-ref/.

C. Kamprath, E. Adolphson, T. Mitamura, and E. Nyberg. 1998. Controlled Language for Multilingual Document Production: Experience with Caterpillar Technical English. In *Second International Workshop on Controlled Language Applications (CLAW '98)*.

O. Lassila and R.R. Swick. 1999. Resource Description Framework (RDF) Model and Syntax Specification. Technical Report 19990222, W3C Consortium, `http://www.w3.org/TR/REC-rdf-syntax/`.

N.F. Noy, M. Sintek, S. Decker, M. Crubzy, R.W. Fergerson, and M.A. Musen. 2001. Creating Semantic Web Contents with Protégé-2000. *IEEE Intelligent Systems*, 16(2):60–71.

Uta Schwertel. 2000. Controlling plural ambiguities in Attempto Controlled English. In *Proceedings of the 3rd International Workshop on Controlled Language Applications*, Seattle, Washington.

D. Skuce. 2003. A Controlled Language for Knowledge Formulation on the Semantic Web. http://www.site.uottawa.ca:4321/factguru2.pdf.