Let's Talk in Description Logic via Controlled Natural Language

Rolf Schwitter and Marc Tilbrook

Centre for Language Technology, Macquarie University
Sydney NSW 2109, Australia
{schwitt, marct}@ics.mq.edu.au

Abstract. In this paper, we will argue that a well-defined subset of English can be used to express the same kind of information as the description logic layer of the Web Ontology Language OWL DL. We will first show what kind of problems current notations for OWL have and then discuss how an OWL ontology can be constructed alternatively in a controlled natural language (CNL). In particular, we will present the details of a CNL which offers the same expressive power as OWL DL for describing the facts, axioms and restrictions which may occur in an OWL ontology. An ontology specification written in CNL can be unambiguously translated into OWL abstract syntax and vice versa with the help of a bi-directional grammar. The users of the CNL do not need to learn the rules of the language, since the writing process is supported by an intelligent authoring tool.

1 Introduction

Most information on the existing Web is designed for humans to read but not for computers to manipulate in a meaningful way. The goal of the Semantic Web is to give information explicit meaning making it is easier for computers to automatically process and share this information [3].

In this context, it is the task of a formal ontology to give information explicit meaning. A formal ontology describes the intensional information of a specific domain via a set of terminological axioms. These axioms define the classes to which the individuals in the domain may belong to, the relationships which may exist among these individuals, and the properties these individuals may have. The extensional information of a domain can then be provided via a collection of facts in a knowledge base which make concrete assertions about instances of classes, relations, and properties. Depending on the expressive power of the underlying formal language, inference rules can be used for answering questions over the knowledge base as well as for checking whether or not the formalized knowledge is correct, meaningful, or redundant.

Since ontology construction is a recurrent task and since ontologies will be distributed over the Web, the World Wide Web Consortium (W3C) developed the Web Ontology Language (OWL) for defining and instantiating ontologies [20].

OWL consists of a family of languages and provides three increasingly expressive sublanguages: OWL Lite, OWL DL, and OWL Full.

OWL DL is interesting, since it has a version of description logics as its formal underpinning and is the most expressive of these three sublanguages which does not compromise completeness and decidability [9]. Because an OWL DL ontology can be translated into a description logic representation, it is possible to perform automated reasoning tasks over the formalized knowledge using state-of-the-art description logic reasoners (for example [8]).

Although there exist a number of good ontology development tools, for example Protégé [12, 14] or SWOOP [11], experience shows that OWL DL ontologies are difficult to construct from scratch. It has been convincingly argued that the exact meaning of expressions in OWL should first be paraphrased in explicit natural language to get a clear understanding of the logical meaning and the potential inferences before trying to encode these expressions in a formal notation [17].

We will argue in this paper that if it is possible to paraphrase an OWL ontology precisely in explicit natural language, then it is also possible to write this ontology with adequate tool support [18, 22] in a controlled natural language (see [10] for an introduction) and then translate this controlled language automatically into OWL abstract syntax [15] and vice versa.

2 Features of OWL

OWL relies on XML [4] for syntax and is semantically layered on top of RDF/RDFS [13] from where the three sublanguages borrow different sets of constructors which affect their expressive power. In particular, OWL uses RDF URI references (URIrefs) as a naming mechanism, many built-in datatypes from XML Schema, and the basic fact-stating capabilities of RDF and the class- and property-structuring abilities from RDFS [5, 9]. OWL DL extends the RDFS vocabulary and makes it possible to specify, for example:

- arbitrary logical combinations of classes and restrictions via set operations;
- classes via direct enumeration of their members;
- properties as transitive, symmetric, functional, inverse functional or as the inverse of other properties;
- two classes or two properties as equivalent;
- a set of classes as disjoint from each other;
- individuals as pairwise identical or different;
- restrictions on how properties behave that are local to a class.

OWL DL includes all language constructs of OWL, but some of them can only be used under certain restrictions to retain computational completeness and decidability.

3 Notations for OWL

To put it simply, an OWL ontology is an RDF graph consisting of a set of RDF triples. As it is the case for any RDF graph, an OWL ontology graph can be written in many different notations. The three most important notations for OWL are: N-Triples notation [1], RDF/XML exchange syntax [6], and OWL abstract syntax [15].

3.1 N-Triples Notation

The N-Triples notation is a line-oriented, plain text format for encoding RDF graphs. In this notation, each statement in a graph is written as a triple of ordered URIrefs which distinguish the subject, predicate, and object in a statement and indicate which (external) vocabularies are referred as Fig. 1 illustrates:

```
<http://www.example.org/pizza.owl#VegetarianPizza>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.w3.org/2002/07/owl#Class> .
```

Fig. 1. Full N-Triples Notation

An OWL ontology usually includes a set of XML namespace declarations which provide an abbreviation mechanism for URIrefs and make an ontology more readable. Although this abbreviated notation is more compact, it still has a number of severe shortcomings: Firstly, the N-Triples notation breaks down many OWL constructs – for example property restrictions and enumerations – into several triples which makes the notation difficult to read. Secondly, it is hard to reconstruct and understand how the individual triples belong together, since grouping is not possible in this notation. Thirdly, all triples are accessible and therefore it is not possible to exclude circular or other unusual structures [9].

Notation 3 [2] is an alternative but non-standard notation for RDF graphs and provides a grouping mechanism and a shorthand notation for a few constructors.

3.2 RDF/XML Exchange Syntax

The need to maintain maximum upward compatibility with existing Web languages based on XML and RDF required the use of an RDF/XML based syntax for OWL [1, 6]. Unlike N-Triples, which are intended as a shorthand notation, RDF/XML is the normative exchange syntax for publishing and sharing OWL ontologies on the Web. This notation is extremely difficult to read and write for humans. It was primarily designed to be processed by machines and not intended for human consumption.

3.3 OWL Abstract Syntax

Using the N-Triples notation or the RDF/XML exchange syntax does not pay enough attention to the readability of an ontology. The goal of the OWL abstract syntax is to fix this problem and to provide a mapping from this abstract syntax to RDF graphs [15]. OWL abstract syntax uses a frame-like notation, where the information about a class or property is grouped together in one large syntactic chunk. Below is an example of a class axiom and property axiom written in OWL abstract syntax.

Fig. 2 shows a **class axiom** which defines a *VegetarianPizza* class in OWL abstract syntax. Here, the class axiom states that the class *VegetarianPizza* is exactly equivalent (= *complete*) to the conjunction of the superclass *Pizza* and a number of restrictions (which both occur on the right hand side of the notation). The two set constructors *complementOf* select all individuals that do not fall under the specified restrictions (*restriction*). These negated restrictions describe that there exists at least one value (*someValueFrom*) for the *hasTopping* property that belongs both to the *MeatTopping* class and also to the *FishTopping* class.

```
Class(VegetarianPizza complete
      complementOf(restriction(hasTopping
                    someValuesFrom(MeatTopping)))
      complementOf(restriction(hasTopping
                    someValuesFrom(FishTopping)))
      Pizza)
```

Fig. 2. Definition of *VegetarianPizza* Class in OWL Abstract Syntax

Fig. 3 shows a **property axiom** which describes the *hasTopping* property in OWL abstract syntax. Here, the property *hasTopping* is defined as an *ObjectProperty* taking only individual values but no datatypes as objects. Furthermore, the *hasIngredient* property is stated to be a superproperty of the *hasTopping* property. Properties have a domain and a range, and link individuals from the domain to individuals from the range. In our case, the *hasTopping* property links individuals of the *Pizza* class to individuals of the *PizzaTopping* class. The *inverseOf* property characteristic specifies that *hasTopping* is the inverse of *isToppingOf*. That means *hasTopping* implies *isToppingOf* and *isToppingOf* implies *hasTopping* or simply: *hasTopping* is equivalent to *isToppingOf*.

The OWL abstract syntax improves the readability in contrast to the other two notations. However, it requires an understanding of the meaning of the constructors and their logical implications which is not always obvious and straightforward to memorise for a novice user.

```
    ObjectProperty(hasTopping super(hasIngredient)
                              domain(Pizza)
                              range(PizzaTopping)
                              inverseOf(isToppingOf))
```

Fig. 3. Definition of *hasTopping* Property in OWL Abstract Syntax

4 Controlled Natural Language (CNL)

As an alternative to these notations, we suggest using a controlled natural language (CNL) which offers the same expressive power as OWL DL for describing the facts and axioms which are part of an OWL ontology. A CNL is a well-defined subset of a natural language with a restricted grammar and a restricted vocabulary [10]. In contrast to full natural language, these restrictions usually reduce the ambiguity of the language, increase the readability for humans, and improve the processability for machines [7, 19, 21]. As we will see, the users of the CNL do not need to learn the rules of the language, since they are supported by an intelligent authoring tool that guides the writing process [18, 22].

4.1 Expressing Facts in CNL

In our context, facts are either used to state information about a particular individual or to make individuals pairwise identical or distinct.

Assertions. The first kind of facts can be used to make assertions which assign an individual to a specific class or inform about the properties and values of that individual. In the simplest case, these assertions can be expressed in CNL as simple sentences consisting of a subject-predicate-object pattern, for example:

1. *France is a country.*
2. *Luc orders exactly three pizzas.*

Sentence *1* ties an individual to a class of which it is a member via an indefinite noun phrase and sentence *2* specifies the exact number of elements which can occur in the object position of a binary relation via a cardinal noun phrase.

In CNL, it is possible to build composite sentences from simpler sentences with the help of coordinators as in sentence *3* and subordinators as in *4*:

3. *France is a country and Paris is a city.*
4. *Luc who orders exactly three pizzas lives in Paris.*

In CNL, variables can be used to identify anonymous resources and provide the necessary connectivity between more than one simple sentences:

5. *Luc has X as an address. X has Grande Rue as a street name. X has 36 as a street number.*

In the example above, *Grande Rue* is a Unicode string and *36* is a positive integer. The authoring tool of the CNL distinguishes between plain literals (Unicode strings) and typed literals (which occur with an internal URIref) and makes them distinct during the writing process.

Equality and Inequality. The second kind of facts provides a mechanism for making two individuals pairwise identical as in sentence *6*, one individual different from another one as in *7*, or a number of individuals mutually different as in *8*:

6. *Luc is identical to Lucien.*
7. *Australia is different from Austria.*
8. *England and Italy and France are different.*

Note that if two individuals have different names, but have not been specified as different, then it may still be possible to derive by inference that they must be identical, since there exists no unique name assumption in OWL.

4.2 Expressing Axioms in CNL

In our context, axioms are used to provide terminological information about classes and properties and to build hierarchical structures.

Class Axioms in CNL. Basically, class axioms are used to specify that a class is a subclass of another class thereby building a class hierarchy or to state that a class is exactly equivalent to the conjunction of a superclass and a set of restrictions thereby stating that a class definition is complete.

Class Hierarchies. In CNL, an *if-then* construction is used to relate a subclass to a superclass and to organise classes in a hierarchical structure, for example:

9. *If X is a vegetarian pizza then X is a pizza.*
10. *If X is a pizza then X is a dish.*

The two sentences *9* and *10* only provide partial definitions describing where the classes are located in a class hierarchy, but they do not define these classes completely via a set of necessary and sufficient conditions.

Definitions. To state a class definition in CNL as complete, an *iff-then* construction is used to indicate that a class is equivalent to the conjunction of a superclass plus a set of restrictions, for example:

11. *Iff X is a vegetarian pizza then X is a pizza that does not have some meat as a topping and does not have some fish as a topping.*

Here, the relative clause *that ...* together with the corresponding verb phrase triggers OWL's `restriction` constructor as illustrated in Fig. 2, the negation *does not* corresponds to OWL's `complementOf` constructor, and the quantifier *some* corresponds to OWL's `someValueFrom` property restriction.

Enumerations. In CNL, classes can be defined via a direct enumeration of their members and this class membership can be expressed with the help of an *either-or* construction, for example:

12. *Iff X is a country then X is either England or Italy or France.*

Here, no other individual than either *England* or *Italy* or *France* belongs to the *country* class. Note that these individuals need to be asserted to be all different from each other – as already mentioned in the discussion of sentence *8*.

Equivalent and Disjoint Classes. Classes can be made equivalent to other classes as in sentence *13* or disjoint from other classes as in sentence *14*:

13. *Iff X is a vegetarian pizza then X is a veggie pizza.*
14. *Iff X is a mushroom pizza then X is not a cheese pizza.*

Equivalent classes take the same individuals as instances and can be used to create synonymous classes. Disjoint classes guarantee that an individual which is a member of one class cannot at the same time be an instance of a specified other class.

Property Axioms in CNL. Basically, property axioms assert general facts about the members of a class and specific facts about individuals. In CNL, there are two different types of properties available: properties which take individuals as values in object position and properties which take datatypes as values in object position.

Property Hierarchies. Like classes, properties can be arranged in a hierarchy, for example:

15. *If X has Y as a topping then X has Y as an ingredient.*
16. *If X has Y as an ingredient then X has Y as a part.*

Properties can be restricted in various ways in their subject and object position as we will discuss below.

Equivalent Properties. Like classes, properties can be made equivalent to other properties using an *iff-then* construction, for example:

17. *Iff X has Y as an ingredient then X has Y as a component.*

Stating property equivalence provides a mechanism for defining synonyms which is particularly useful if two ontologies need to be merged.

Property Restrictions in CNL. As in OWL various forms of restrictions can be used in CNL to specify global, local or cardinality constraints on properties.

Global Restrictions. Properties can be given global domain restrictions in their subject position as well as global range restrictions in their object position, for example:

18. *If X has Y as a topping then X is a pizza and Y is a pizza topping.*

Here, the *pizza* class is the domain and the *pizza topping* class is the range of the property which relates instances of the two classes to each other. These restrictions are called global, since they are stated on the properties and not just on a property when it is associated with a particular class.

Local Restrictions. Properties can be given local restrictions in the object position specifying that all values, at least one value, or a specific value is a member of a particular class:

19. *Iff X is a meaty pizza then X is a pizza that has only meat as topping.*
20. *If X is a choc sundae then X has some chocolate as a topping.*
21. *If X is a red wine then X has red as a colour.*

The quantifier *only* in sentence *19* guarantees that all values of the *topping* property belong to the *meat* class. The quantifier *some* in *20* ensures that there is at least one value for the *topping* property that belongs to the *chocolate* class. And the value *red* in *21* specifies a specific value for the *colour* property.

Cardinality Constraints. There exist three cardinality constraints in CNL (*at least N, at most N* and *exactly N*) which allow for specifying the number of elements that can occur in the object position of a sentence – as already mentioned in the discussion of sentence *4*.

Property Characteristics in CNL. Properties can be further specified via a number of property characteristics: properties can be made transitive as in sentence *22* or symmetric as in sentence *23*:

22. *If X is located in Y and Y is located in Z then X located in Z.*
23. *Iff X is adjacent to Y then Y is adjacent to X.*

A property can also be specified as the inverse of another property and the logical meaning of this property characteristic can then be made explicit as follows using an *iff-then* construction:

24. *Iff X has Y as a topping then Y is a topping of X.*

Furthermore, properties can be made functional as in sentence *25* or inverse functional as in sentence *26*:

25. *If X has Y as a base and X has Z a base then Y is identical to Z.*

26. *If Y is the base of X and Z is the base of X then Y is identical to Z.*

If a property is declared to be functional, then it does not have more than one value in the object position for a specific instance which occurs in the subject position. If a property is declared to be inverse functional, then the object of a property uniquely determines the subject.

Given this information, we can now fully specify the `hasTopping` property in CNL which corresponds to the OWL abstract syntax definition in Fig. 3:

27. *If X has Y as a topping then X has Y as an ingredient and X is a pizza and Y is a pizza topping and Y is a topping of X.*
28. *If Y is a topping of X then X has Y as a topping.*

The interesting thing here is that the logical meaning of the `inverseOf` property characteristic is made explicit on the level of the CNL in contrast to the OWL abstract syntax notation.

5 A Walkthrough Example

The following section illustrates in a walkthrough example how a terminological axiom written in CNL can be translated into OWL abstract syntax. Since the grammar is bi-directional, it can alternatively take an expression in OWL abstract syntax as input and generate a sentence in CNL as output. To keep things simple here, we present the grammar in definite clause grammar (DCG) format [16] and use an attribute-value notation to describe feature structures in the grammar rules. We do not focus in our discussion on how a slightly modified version of this grammar can be used by a chart parser to harvest look-ahead information for the predictive text editor of the CNL.

Sentence *11* (repeated below as *29*) is a terminological axiom which defines the *vegetarian pizza* class in CNL. This definition uses an *iff-then* construction that connects the class to be defined (= definiendum) on the left hand side and the defining superclass and restrictions (= definiens) on the right hand side:

29. *Iff X is a vegetarian pizza then X is a pizza that does not have some meat as a topping and does not have some fish as a topping.*

This sentence is interesting, because it combines a number of different linguistic phenomena which the grammar needs to cover. In particular, this complex sentence consists of a simple sentence which describes the definiendum and a complex sentence consisting of a simple sentence and a dependent relative clause which describe the definiens. The relative clause contains two coordinated verb phrases which are both negated.

After processing this sentence, the resulting translation should look as follows in OWL abstract syntax:

```
'Class'([vegetarian,pizza],complete,
        [[pizza],
         complementOf(restriction([has,topping],
                      someValuesFrom([meat,topping]))),
         complementOf(restriction([has,topping],
                      someValuesFrom([fish,topping])))])
```

A major problem that we face when we process this terminological axiom is that we do not know in advance what the form of the content words (for example, *vegetarian pizza*) look like. In the TBox mode, we build up the terminology and define the content words (or identifiers) and in the ABox mode we then use these content words for making assertions or classifying instances. For this reason, we distinguish two types of grammar rules: rules which are used in the TBox mode (*tbox*) and rules which we are used in the ABox mode (*abox*).

Below is the top-level grammar rule which processes sentence *29*. It basically splits up the complex sentence into two parts: a sentence with a simple structure (*X is a vegetarian pizza*) which describes the definiendum (*dfm*) and a sentence with a complex structure (*X is a pizza that ...*) which describes the definiens (*dfs*):

```
s2( tbox:B, owl:['Class'(I1,complete,I2)], tok:[T1|T2] ) -->
    ['Iff'],
    s( tbox:dfm, coord:n, owl:[]-I1, tok:[]-T1 ),
    [then],
    s( tbox:dfs, coord:n, owl:[]-I2, tok:[]-T2 ),
    ['.'].
```

In this DCG notation, attribute-value pairs represent feature structures and are written as *attribute:value*. Variables occur in uppercase and constants in lowercase. Some attribute take difference lists (for example *[]-I1* or *[]-T1*) as values. In general, a difference list uses a pair of lists to represent a list whereas the first list is the full list and the second list the tail of the first list.

The feature structure *owl:['Class'(I1,complete,I2)]* on the left hand side of the abovementioned grammar rule is important, since it is the place where the expression in OWL abstract syntax will be built up for an input sentence or where an existing expression in OWL abstract syntax can be used to drive the generation of a sentence in CNL. The predicate *'Class'(I1,complete,I2)* which builds the scaffolding for a final class definition contains two variables: *I1* and *I2* which are both instantiated during parsing. They will hold the relevant information that has been derived from the definiendum and definiens of the input sentence.

The grammar rule uses an additional feature structure (*tok:[]-T1*) and *tok:[] -T2* to collect all content words which are part of the definition. Once collected, these content words are added to the knowledge base and can then be used as lexical entries in the ABox mode to process assertional statements.

Finally, the feature structure *coord:n* indicates that both sentences on the right hand side of the arrow cannot be coordinated on the sentence level.

The next grammar rule deals with the definiendum and the definiens and describes them both as consisting of a noun phrase (*X*) and a corresponding verb phrase (*is a vegetarian pizza* or *is a pizza that ...*):

```
s( tbox:B, coord:n, owl:I, tok: T) -->
   np( tbox:B, coord:n, func:subj, quant:Q, tok:_ ),
   vp( tbox:B, coord:n, func:pred, owl:I, tok:T ).
```

The feature structure *func:subj* is used to constrain the form of the noun phrase in the subject position and the feature structure *func:pred* is used to control the form of the verb phrase in the predicate position. Also here, the feature structure *coord:n* indicates that both phrases cannot be coordinated on this level.

In our case, the two noun phrases in subject position consist of a simple variable and these variables serve as a handle to refer to the same entity, but do not contribute anything to the final representation:

```
np( tbox:B, coord:n, func:subj, quant:Q, tok:T ) -->
      ([‘X’] ; [‘Y’] ; [‘Z’]).
```

The verb phrases of both sentences consist of a copula (*is*) which is followed by an indefinite noun phrase (*a vegetarian pizza* and *a pizza*) in the object position. The following grammar rule describes this structure:

```
vp( tbox:B, coord:n, func:pred, owl:I, tok:T ) -->
    [is],
    np( tbox:B, coord:n, func:obj, quant:exist(a), owl:I, tok:T).
```

Here, the feature structure *quant:exist(a)* indicates that an existential quantifier is required which can be realised in form of an indefinite determiner. The noun phrase in object position cannot be coordinated as the feature structure *coord:n* shows.

In the case of the definiendum, the noun phrase (*a vegetarian pizza*) in object position consists of a determiner (*a*) and a nominal complex (*vegetarian pizza*):

```
np( tbox:dfm, coord:n, func:obj, quant:Q, owl:I, tok:T) -->
    det( quant:Q ),
    nc(  owl:I, tok:T ).
```

In our case, the nominal complex consists of an adjective (*vegetarian*) and a noun (*pizza*). But the grammar does not distinguish between these word forms and treats the entire expression as a single concept, since there is no lexical information available which could help here. The following grammar rules process the determiner and the nominal complex:

```
det( quant:exist(a) ) --> ( [a] ; [an] ).

nc( owl:I1-[T2|I1], tok:T1-[T2|T1],  L1, L2 ) :-
    append([T2],L2,L1).
```

```
nc( owl:I1-[T3,T2|I1], tok:T1-[T3,T2|T1], L1, L2 ) :-
    append([T3,T2],L2,L1).
```

Processing the determiner is straightforward but the nominal complex requires some consideration. Since we do not know what the nominal complex looks like beforehand, it is a good idea to constrain its length to speed up the processing. In our example, the grammar rules specify that a nominal complex can either consist of one token (*T2*) or two tokens (*T3* and *T2*), but not more.

In the case of the definiens, the noun phrase (*a pizza that ...*) in object position consists not only of an indefinite determiner and a nominal complex but contains an complex relative clause (*that does not have ...*) which functions as a modifier:

```
np( tbox:dfs, coord:n, func:obj, quant:Q, owl:I1-[I3|I2],
    tok:T1-[T3|T2]) -->
    det( quant:Q ),
    nc( owl:[]-I3, tok:[]-T3 ),
    rc( tbox:dfs, func:mod, owl:I1-I2, tok:T1-T2 ).
```

The feature structure *func:mod* indicates that the remaining part of the noun phrase is a modifier realised by a relative clause:

```
rc( tbox:dfs, func:mod, owl:I, tok:T ) -->
    [that],
    vp( tbox:dfs, coord:_, func:mod, owl:I, tok:T ).
```

In our example, the relative clause consists of two verb phrases (*does not have some meat ... and does not have some fish ...*):

```
vp( tbox:dfs, coord:y, func:mod, owl:I1-I3, tok:T1-T3 ) -->
    vp( tbox:dfs, coord:n, func:mod, owl:I2-I3, tok:T2-T3 ),
    [and],
    vp( tbox:dfs, coord:_, func:mod, owl:I1-I2, tok:T1-T2 ).
```

The two feature structures *coord:y* and *coord:n* handle the coordination of the verb phrases and at the same time exclude left recursion.

In our example both verb phrases are negated and make a complex contribution to the OWL abstract syntax representation as the value of the *owl* attribute shows:

```
vp( tbox:dfs, coord:n, func:mod,
    owl:I1-[complementOf(restriction([has|I2],someValuesFrom(I3)))|I1],
    tok:T1-[T3|T1] ) -->
    [does,not],
    [have],
    np( tbox:dfs, coord:_, func:mod, quant:exist(some),
        owl:I2-I3, tok:T2-T3 ),
    pp( tbox:dfs, func:mod, quant:exist(a), owl:[]-I2, tok:[]-T2 ).
```

In particular, the negation (*does not*) triggers the `complementOf` constructors and the verb (*have*) together with the noun phrase and the prepositional phrase contribute to the representation of the `restriction` constructor. It is the determiner (`some`) of the noun phrase which triggers the `someValuesFrom` constructor. The interesting thing here is that the nominal information derived from the prepositional phrase (*topping*) needs to be distributed into two different lists (`[has,topping]` and `[meat,topping]`) for the final representation.

The following grammar rule deals with the subsequent noun phrases (*some meat* and *some fish*) which are part of the modifying structure and which provide the `meat` and `fish` class information:

```
np( tbox:dfs, coord:n, func:mod, quant:Q, owl:I, tok:T ) -->
    det( quant:Q ),
    nc( owl:I, tok:T ).
```

And finally the next grammar rule deals with the prepositional phrase (*as a topping*) which is also part of the modifying structure and which contributes the `topping` information to the OWL representation:

```
pp( tbox:dfs, coord:C, func:mod, quant:Q, owl:I, tok:T ) -->
    [as],
    np( tbox:dfs, coord:C, func:mod, quant:Q, owl:I, tok:T ).
```

The rule for the determiner of the modifying noun phrase is straightforward:

```
det( quant:exist(some) ) --> [some].
```

and completes our walkthrough example for the terminological axiom.

6 Writing in Controlled Natural Language

The writing of an ontology in CNL is supported by a predictive text editor which generates lookahead information while a text is written [18]. The text editor can be used either to express terminological axioms in the TBox mode or to assert factual information about a specific domain in the ABox mode. The user of the editor does not need to learn the rules of the CNL explicitly, since the writing process is completely guided by the editor.

If the user plans to specify a set of terminological axioms in CNL which describe the intensional aspects of a particular domain, then he or she uses the TBox mode of the text editor. Once a set of terminological axioms has been specified, then the resulting terminology can be used in the ABox mode to specify instance data. From the terminological information available in the ontology, new lookahead information can be harvested to guide the writing process in the ABox mode.

7 Conclusion

In this paper, we argued that those notations which are currently available for the Web Ontology Language OWL are difficult to write and understand for humans. Although the N-Triples notation is compact, it does not allow for grouping of triples. The RDF/XML exchange syntax is a normative notation designed for machines, but it is not suitable for human consumption. The OWL abstract syntax is a frame-like notation, but it does not make the meaning of the syntactic constructors explicit enough for novice users. We showed that a controlled natural language is able to overcome these shortcomings and can serve as a high-level language for specifying ontologies provided that the user is supported by an intelligent authoring tool. Ontologies in controlled natural language can be translated into OWL abstract syntax and vice versa via a bi-directional grammar. The presented approach improves the readability and understandability of an ontology without compromising its precision and formality.

Acknowledgments

The research reported here is supported by the Australian Research Council, Discovery Project No. DP0449928 and has been partially conducted while the first author was working as a visiting professor at the Centre Tesnière in Besançon, France.

References

1. Beckett, E.: RDF/XML Syntax Specification (Revised). *W3C Recommendation*, 04 February 2004.
2. Berners-Lee, T.: Notation 3. An RDF language for the Semantic, 2001. Available at: <http://www.w3.org/DesignIssues/Notation3.html>, 2001.
3. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. In: *Scientific American*, May 17, 2001.
4. Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., Yergeau, F.: Extensible Markup Language (XML) 1.0 (Third Edition). *W3C Recommendation*, 04 February 2004.
5. Brickley, D., Guha, R. V.: RDF Vocabulary Description Language 1.0: RDF Schema. *W3C Recommendation*, 10 February 2004.
6. Dean, M., Schreiber, G.: OWL Web Ontology Language Reference. *W3C Recommendation*, 10 February 2004.
7. Fuchs, N. E., Schwertel, U., Schwitter, R.: Attempto Controlled English - Not Just Another Logic Specification Language. *LNCS 1559*, Springer, 1999, 1–20.
8. Haarslev, V., Möller, R.: RACER system description. In: *Proceedings of the 2nd International Joint Conference on Automated Reasoning (IJCAR)*, LNAI 2083, Springer, 2001, 701–705.
9. Horrocks, I., Patel-Schneider, P. F., van Harmelen, F.: From SHIQ and RDF to OWL: The Making of a Web Ontology Language. In: *Journal of Web Semantics 1*, 2003, 7–26.

10. Huijsen,W. O.: Controlled Language - An Introduction. In: *Proceedings of CLAW 1998*, Pittsburgh, 1998, 1–15.
11. Kalyanpur, A., Parsia, B., Hendler, J.: A Tool for Working with Web Ontologies. In: *Proceedings of the International Journal on Semantic Web and Information Systems*, Vol. 1, No. 1, 2005.
12. Knublauch, H., Fergerson, R. W., Noy, N. F., Musen, M. A.: *The Protege OWL Plugin: An Open Development Environment for Semantic Web Applications*. Third International Semantic Web Conference - ISWC 2004, Hiroshima, Japan, 2004.
13. Manola, F., Miller, E.: RDF Primer. *W3C Recommendation*, 10 February 2004.
14. Noy, N., Sintek, M., Decker, S., Crubezy, M., Fergerson, R., Musen, M.: Creating semantic web contents with Protégé-2000. In: *IEEE Intelligent Systems*, 2001.
15. Patel-Schneider, P. F., Hayes, P., Horrocks, I.: OWL Web Ontology Language Semantic and Abstract Syntax. *W3C Recommendation*, 10 February 2004.
16. Pereira, F. C. N., Shieber, S. M.: Prolog and Natural-Language Analysis. CSLI Lecture Notes, Number 10, 1987.
17. Rector, A., Drummond, N., Horridge, M., Rogers, J., Knublauch, H., Stevens, R., Wang, H., Wroe, C.: OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors and Common Patterns. In: *Proceedings of the European Conference on Knowledge Acquisition*, Northampton, England, 2004, LNAI 3257, Springer-Verlag, 2004, 63–81.
18. Schwitter, R., Ljungberg, A., Hood, D.: ECOLE - A Look-ahead Editor for a Controlled Language. In: *Proceedings of EAMT-CLAW03*, May 15-17, Dublin City University, 2003, 141–150.
19. Schwitter, R.: A Layered Controlled Natural Language for Knowledge Representation. In: *Machine Translation, Controlled Languages and Specialised Languages*, Special Issue of Linguisticae Investigationes, Vol. 28, No. 1, 2005, 85–106.
20. Smith, M. K., Welty, C., McGuinness, D. L.: OWL Web Ontology Language Guide. *W3C Recommendation*, 10 February 2004.
21. Sowa, J. F.: Common Logic Controlled English. Draft, 24 February 2004.
22. Thompson, C.W., Pazandak, P., Tennant, H. R.: Talk to Your Semantic Web. In: *IEEE Internet Computing*, Vol. 9, No. 6, 2005, 75–79.