Abstract

In this paper I will agitate for a paradigm shift in knowledge acquisition and knowledge representation for the Semantic Web. Instead of using formal languages based on RDF that are difficult to use for non-specialists, I will argue for employing a controlled natural language as interface language to the Semantic Web. I will first show how a controlled natural language can be used to describe resources on the Web and to construct a variant of an OWL ontology, and then discuss how rules can be expressed in controlled natural language for reasoning with this ontology. There is no need to formally encode assertional and terminological knowledge or rules in an RDF-based language. Everything can be described on the level of the controlled natural language provided that we support the user with an intelligent writing tool. The specified knowledge can be automatically translated into a subset of first-order predicate logic and the formulae can be further processed by a model builder. The task of the model builder consists in attempting to generate a (finite) satisfying model for these formulae. In doing so, the model builder produces as side-effect all entailed statements in controlled natural language. The generated model builds the prerequisite for question answering in controlled natural language.

1 Introduction

The vision of the Semantic Web is to extend the current Web in a way in which information is given well-defined meaning enabling computers and people to work in cooperation (Berners-Lee et al., 2001). To a certain degree, cooperation between computers can be achieved by annotating information (assertional knowledge) on the Web with machine-processable data and by linking these annotations to ontologies (terminological knowledge). In the ideal case, ontologies can be combined with rule languages so that new (entailed) information can be inferred and questions about assertional (and terminological) knowledge can be answered with the help of reasoning services (Grosof et al., 2003).

However, cooperation between computers and people on a world-wide scale can hardly be achieved via RDF-based formal languages such as RDFS (Brickley and Guha, 2004) and OWL (Smith et al., 2004). People (in particular non-specialists) need to be able to add new machine-readable information to a Web site. They need to be able to express their questions in a familiar notation, and to read and understand information derived from a piece of potentially distributed knowledge.

What is urgently needed is a high-level interface language to the Semantic Web that abstracts away from these RDF-based formal notations. I will show that a well-designed controlled natural language (CNL) is an ideal candidate to increase the transparency of the Semantic Web and to empower non-specialists with a “seemingly informal” language to work in cooperation with computers.
Usually a CNL is defined as a well-defined subset of a natural language that has been restricted with respect to its grammar and its lexicon. Grammatical restrictions result in less complex and less ambiguous sentences, while lexical restrictions reduce the size of the lexicon and the meaning of the lexical entries for a particular domain (Huijsen, 1998). This definition is a good starting point for our undertaking but we need to keep in mind that we are going to design a CNL that has the same formal properties as an ontology language layered on top of RDF. This seriously restricts the expressivity of this CNL.

The reminder of this paper is organised as follows: In Section 2, I will briefly introduce OWL and OWL Lite and then discuss why the formal properties of OWL Lite are a promising for reasoning with rules. In Section 3, I will show by a number of examples that a CNL can completely replace OWL Lite statements and that even rules can be stated directly in CNL. Moreover, I will demonstrate how the writing of statements and rules can be supported with the help of an intelligent writing tool. In Section 4, I will describe how statements and rules in CNL can be translated into a subset of first-order predicate logic that can then be processed by a standard model builder. In Section 5, I will provide a complete rule set written in CNL for reasoning with assertional and terminological knowledge that has the same expressive power as OWL Lite. Taking the examples introduced in Section 3 and the rule set from Section 5, I will investigate in Section 6 which entailed statements can be inferred by the model builder and what kind of questions can be answered over the generated model. Finally, in Section 7, I will conclude and summaries the advantages of the presented approach.

2 OWL and OWL Lite

The Web Ontology Language OWL is based on description logic and comes in three increasingly expressive layers: OWL Lite, OWL DL, and OWL Full (Smith et al., 2004). Ironically, OWL Full and OWL DL are not suitable for reasoning over large and distributed ontologies, since there exist no efficient reasoning algorithms for these languages (de Bruijn et al., 2004). Even OWL Lite, the least expressive layer of these languages, has constructors (e.g. equality, disjunction and negation) that considerably complicate the implementation of efficient reasoning algorithms.

Recently, it has been argued that the intersection of description logic with logic programs can provide a straightforward computational pathway for reasoning and interoperability on the Semantic Web (Horrocks and Patel-Schneider, 2003; Grosof et al., 2003). In particular, OWL Lite has been identified as the maximal subset of OWL which can be expressed in the deductive database language Datalog (de Bruijn et al., 2004). Datalog corresponds to Horn clauses with a range-restriction on variables (all variables in the head of a clause occur also in the body) and without function symbols (of arity greater than zero) and without negation (see (Garcia-Molina et al., 2002)). It has been shown that about 77% of all current ontologies developed for the Semantic Web fall under the OWL Lite subset (Volz, 2004).

OWL Lite is layered on top of RDF which relies on eXtensible Markup Language (XML) for syntax, Uniform Resource Identifiers (URIs) for naming and RDFS Schema (RDFS) for describing meaning and relationships of terms. OWL Lite uses RDF and RDFS constructors whenever the required functionality already exists for a lower layer. In a nutshell: OWL Lite consists of the following constructors whose meaning will become clear in the subsequent discussion:

**Individuals:**

- rdf:type

**Simple classes:**

- owl:Class
- rdfs:subClassOf

**Simple properties:**

- owl:objectProperty
- rdfs:subPropertyOf
- rdfs:domain
- rdfs:range

**Property characteristics:**

- owl:TransitiveProperty
- owl:SymmetricProperty
- owl:inverseOf

**Property restrictions:**

- owl:Restriction
- owl:onProperty
- owl:allValuesFrom
Additionally, OWL Lite has two constructors (owl:equivalentClass and owl:equivalentProperty) that can be used to map between classes and properties from different ontologies.

3 OWL Lite plus Rules in CNL

As discussed in the last section, the syntax of OWL Lite relies on RDF. RDF is based on the idea of identifying things using URIs and describing these things (= resources) in terms of simple properties and property values.

Imagine trying to state that someone named Nora Yuen supervises someone named John Smith. This can be encoded as follows in RDF:

```xml
<rdf:Description rdf:ID='nora_yuen'>
  <ex:supervise rdf:resource='john_smith'/>
</rdf:Description>
```

Here nora_yuen is the identified resource, supervise is a simple property, and john_smith is a property value. The URIs for the resource and the property value are the ones of the current document and the prefix ex indicates that the URI for the property is the one specified in the namespace declaration of the document. A straightforward way to express the above statement in CNL is:

Nora Yuen supervises John Smith.

In CNL (as well as in RDF terminology), the subject ‘Nora Yuen’ identifies the resource of the statement, the predicate ‘supervises’ identifies the property of the statement and the object ‘John Smith’ identifies the value of the statement. The namespaces for these terms are not displayed here. As we will see later in Section 3.4, namespaces are handled by an intelligent text editor that supports the writing process of CNL. Please note that the user does not need to learn the syntactic rules of the CNL, since these rules are enforced by the text editor via a lookahead mechanism (Schwitter and Tilbrook, 2004).

3.1 Assertional Statements

The basic syntactic structure for making assertional statements in CNL consists of a simple subject-predicate-object pattern and variations that can be mapped into “triples”. Here are a few assertional statements in CNL (that we will feed later for illustration purposes to the model builder):

Nora Yuen is a linguist and supervises John Smith. John Smith who is a friend of Kylie Miller is a PhD student. Kylie Miller is drilled by Nora Yuen.

The first statement with the coordinator and is a compound one and is equivalent to two simple statements:

Nora Yuen is a linguist. Nora Yuen supervises John Smith.

The second statement – mentioned above – with the subordinator who is a complex one and corresponds to the two simple statements:

John Smith is a friend of Kylie Miller. John Smith is a PhD student.

The third statement is a passive construction and is interpreted as the inverse of the statement:

Nora Yuen drills Kylie.

if the corresponding relationship between drill and be drilled by is specified in the ontology via a terminological statement (see next section).

3.2 Terminological Statements

In contrast to assertional statements that are most likely to be made by users with different kind of computational background, terminological statements will most probably be uttered by knowledge engineers in order to construct an ontology. Terminological statements speak about classes, properties, instances of classes, and various kinds of relationships between instances, classes and properties. The following statement

The property ‘supervise’ has the type ‘object property’.

talks for example about the ‘supervise’ property and assigns a specific type to it. This statement can be written in an abbreviated form in CNL:

'supervise' has the type 'object property'.

Below are a number of terminological statements with the corresponding “naturalized” OWL Lite constructors (e.g., 'has the domain' or 'is a subproperty of' in predicate position:

'supervise' has the domain 'professor'.
supervise has the range 'PhD student'.
'teach' has the type 'object property'.
teach' has the domain 'academic'.
teach' has the range 'student'.
'drill' is the inverse of 'be drilled by'.
'drill' has the equivalent property 'instruct'.
'drill' is a subproperty of 'teach'.
'professor' is a subclass of 'academic'.
'linguist' is a subclass of 'researcher'.
'researcher' is a subclass of 'scientist'.
'PhD student' is a subclass of 'student'.
'be a friend of' is the inverse of 'have as friend'.

This terminological knowledge is not only used by the model builder for reasoning purposes but also by the look-ahead text editor to guide the writing process of assertional statements.

3.3 Conditional Statements
In contrast to OWL Lite, CNL allows for expressing rules in form of conditional statements to build an axiomatic framework for reasoning. The antecedent (and consequent) of a conditional statement can be complex, for example:

If E has a property P whose value is V
and R has the type 'range restriction'
and R is on P
and R has all values from C
then E has the type R.

As this example shows, the CNL allows for variables in rules that directly translate into variables in the formal representation. Furthermore note that complex consequents will be automatically distributed during the translation.

3.4 Writing in CNL
The writing process in CNL is supported by a look-ahead text editor that can be used either to write an assertional specification, to construct an ontology or to build an axiomatic rule set. The user does not need to learn the rules of the CNL explicitly, since she is guided by the look-ahead editor while the text is written (Schwitter and Tilbrook, 2004).

For an assertional specification, the user first selects the ontologies she wants to work with via a menu. Thereby the text editor becomes "ontology-aware" and guides the writing process via look-ahead categories. Let's imagine that the user wants to express the subsequent assertional statement:
Nora Yuen supervises John Smith.

The editor first displays a look-ahead category for the subject position:

[INDIVIDUAL]

After entering the name ‘Nora Yuen’, the editor displays further look-ahead categories (partially) derived from the syntactic information of the grammar and from the available terminological knowledge:
[who — supervises — teaches — drills ... ]

The user either directly types one of these words or selects it from a context menu. After entering the verb ‘supervises’, the look-ahead editor asks for the value of the statement, and so on.

4 Processing CNL
Specifications in CNL are translated into a format that can directly be processed by SATCHMO (Manthey and Bry, 1988; Bry and Yahya, 2000). SATCHMO is a model-generation based theorem prover that takes a set of first-order formulae as input and tries to generate a finite satisfying model for them by combining a forward chaining algorithm for normal cases with a backward chaining one for special cases.

4.1 Syntax of SATCHMO Rules
In SATCHMO all first-order formulae are uniformly represented in an implicational rule format of the form:

ANTECEDENT ---> CONSEQUENT.

The antecedent ANTECEDENT of a rule is either true or a single atomic formula or a conjunction of atomic formulae. The consequent CONSEQUENT is either a single atomic formula (or false or a disjunction of atomic formulae). Since we are working with the Datalog subset of first-order logic, we will end up with only one single atomic formula in the consequent.

Let’s have a look at a few examples: for instance, the assertional statement
Nora Yuen is a linguist and supervises John Smith.

is translated into two SATCHMO rules of the form:

true --->
term([Nora’,’Yuen’],[rdf:type],
    [ex:linguist]).

true --->
term([Nora’,’Yuen’],[ex:supervise],
    [John’,’Smith’]).
A terminological statement such as
'supervise' has the domain 'professor'.

results in a similar translation:
true ---\(\rightarrow\)\(\text{term}([\text{ex:supervise}], [\text{rdfs:domain}], [\text{ex:professor}]).\)

And finally a conditional statement such as
If E has a property P whose value is V
and R has the type 'range restriction'
and R is on P
and R has all values from C
then E has the type R.
is translated into the following rule
\[
\text{term}(E, P, V),
\text{term}(R, [\text{rdf:type}], [\text{owl:restriction}]),
\text{term}(R, [\text{owl:onProperty}], P),
\text{term}(R, [\text{owl:AllValuesFrom}], C) ---\(\rightarrow\)\(\text{term}(E, [\text{rdf:type}], R).\)
\]

Please note that variables in SATCHMO rules are
range-restricted in the same way as in Datalog.

4.2 Model Generation
Starting from the empty interpretation, SATCHMO
operates by attempting to generate a model of its
input rules by trying to find repeatedly new conse-
quences and to satisfy them. A new consequence
of a rule instance \(A \rightarrow C\) is the consequent \(C\) for
which the antecedent \(A\) matches the current (Prolog)
knowledge base, while the consequent \(C\) does not.
In the simplest case, a new consequent is satisfied
by adding a single atomic formula to the knowledge
base. The result is a new set of interpretations that
results from expending the current interpretation (for
further details see (Abdennadher et al., 1995)).

5 Rule Set for Reasoning in CNL
In this section, I will introduce the rule set – writ-
ten in CNL – that is required for reasoning over as-
sertional and terminological knowledge. Note that
this rule set is not available in OWL Lite" and
would have to be developed in a separate rule lan-
guage (Harth and Decker, 2004).

5.1 Class and Property Hierarchies
Subclass property. The most important taxonomic
construction for classes is the subclass property.
This property relates a specific class to a more gen-
eral class. The subclass property is transitive and can
be used to model class hierarchies via the following
rule in CNL:

If C1 is a subclass of C2
and C2 is a subclass of C3
then C1 is a subclass of C3.

This rule guarantees, for example, that if ‘linguist’
is a subclass of ‘researcher’ and ‘researcher’ is a
subclass of ‘scientist’, then ‘linguist’ is a subclass
of ‘scientist’.

Additionally, we need a rule that makes sure that
the type of an individual takes the class hierarchy
into account:

If C1 is a subclass of C2
and E has the type C1
then E has the type C2.

This rule ensures, for example, that if ‘linguist’
is a subclass of ‘researcher’ and ‘Nora’ has the type
‘linguist’, then ‘Nora’ has the type ‘researcher’.

Subproperty. The subproperty relation behaves
in a similar way as the subclass property and allows
for defining hierarchies of properties in CNL using
the following rule:

If P1 is a subproperty of P2
and P2 is a subproperty of P3
then P1 is a subproperty of P3.

This rule states, for example, that if ‘drill’ is a
subproperty of ‘teach’ and ‘teach’ is a subproperty
of ‘inform’, then ‘drill’ is a subproperty of ‘inform’.

Additionally, we need a rule that applies a prop-
erty hierarchy to individuals:

If P1 is a subproperty of P2
and E has P1 whose value is V
then E has P2 whose value is V.

This rule ensures, for example, that if ‘drill’ is a
subproperty of ‘teach’ and ‘Nora’ has the property
‘drill’ whose value is ‘Kylie’, then ‘Nora’ has the
property ‘teach’ whose value is ‘Kylie’.

Domain restriction. In OWL Lite" the domain
of a property can be restricted to a specific class en-
suring that only individuals of this class occur in the
subject position. The following rule handles this:

If E has the property P whose value is V
and P has the domain D
then E has the type D.

This rule guarantees, for example, that if ‘Nora’
has the property ‘supervise’ whose value is ‘John’
and the property ‘supervise’ has the domain ‘profes-
sor’, then ‘Nora’ has the type ‘professor’.

Range restriction. The range of a property can be
restricted in a similar way to a specific class making
sure that only individuals of this class occur in the object position:

\[
\begin{align*}
\text{If } E \text{ has the property } P \text{ whose value is } V \\
\text{and } P \text{ has the range } R \\
\text{then } V \text{ has the type } R.
\end{align*}
\]

This rule states, for example, that if ‘Nora’ has the property ‘supervise’ whose value is ‘John’ and the property ‘supervise’ has the range ‘PhD student’ then ‘John’ has the type ‘PhD student’.

### 5.2 Property Characteristics

**Object property.** In OWL Lite only object properties are allowed but no datatypes as in OWL Lite. That means that the subject and object position must be realised by an individual of a specific class:

- ‘object property’ has the domain ‘class’.
- ‘object property’ has the range ‘class’.

The rules for domain and range restriction introduced above cover the required inferences.

**Transitive property.** The next rule generalizes transitivity for properties:

\[
\begin{align*}
\text{If } E \text{ has the property } P \text{ whose value is } W \\
\text{and } W \text{ has } P \text{ whose value is } V \\
\text{and } P \text{ has the type } \text{‘transitive property’} \\
\text{then } E \text{ has } P \text{ whose value is } V.
\end{align*}
\]

The rule ensures, for example, that if ‘Nora’ has the property ‘is ancestor of’ whose value is ‘Carla’, and ‘Carla’ has also the property ‘is ancestor of’ but with the value ‘Fabian’, and the property ‘is ancestor of’ has the type ‘transitive property’, then ‘Nora’ has the property ‘is ancestor of’ whose value is ‘Fabian’.

**Symmetric property.** A symmetric property is a property that is true in both directions:

\[
\begin{align*}
\text{If } E \text{ has the property } P \text{ whose value is } V \\
\text{and } P \text{ has the type } \text{‘symmetric property’} \\
\text{then } V \text{ has } P \text{ whose value is } E.
\end{align*}
\]

This rule states, for example, that if ‘John’ has the property ‘is a friend of’ whose value is ‘Kylie’ and the property ‘is a friend of’ has the type ‘symmetric property’, then ‘Kylie’ has the property ‘is a friend of’ whose value is ‘John’.

Similar to a subproperty, we can restrict the range and the domain of a symmetric property:

\[
\begin{align*}
\text{If } P \text{ has the type } \text{‘symmetric property’} \\
\text{and the range } R \\
\text{then } P \text{ has the domain } R.
\end{align*}
\]

\[
\begin{align*}
\text{If } P \text{ has the type } \text{‘symmetric property’} \\
\text{and the domain } D \\
\text{then } P \text{ has the range } D.
\end{align*}
\]

**Inverse property.** A property is the inverse property of another property when the variables in the subject and object position of the first property switch their argument positions in the second property:

\[
\begin{align*}
\text{If } E \text{ has the property } P_1 \\
\text{whose value is } V \\
\text{and the property } P_2 \text{ is the inverse of } P_1 \\
\text{then } V \text{ has } P_2 \text{ whose value is } E.
\end{align*}
\]

This rule expresses, for example, that if ‘Nora’ has the property ‘drill’ whose value is ‘Kylie’ and the property ‘be drilled by’ is the inverse of ‘drill’, then ‘Kylie’ has the property ‘be drilled by’ whose value is ‘Nora’.

The following statements define that the inverse property is a symmetric property and that the subject position and object position of an inverse property are realised by individuals:

- ‘inverse of’ has the type ‘symmetric property’.
- ‘inverse of’ has the domain ‘object property’.
- ‘inverse of’ has the range ‘object property’.

**Class equivalence.** In order for ontologies to have the maximum impact, they need to be sharable and re-usable. When two ontologies describe the same class, then a mechanism is needed to state that the two classes are equivalent. That means we need to be able to systematically describe that every individual of one class is also an individual of the other class.

Here are two statements that describe specific characteristics of class equivalence:

- ‘equivalent class’ has the type ‘symmetric property’.
- ‘equivalent class’ has the type ‘transitive property’.

The first statement says that class equivalence is a symmetric property. The rules for symmetry introduced above take this statement into consideration. The second statement says that class equivalence is transitive. The general rule for transitivity introduced above takes care of this statement.

The following rule performs the required inferences on the type hierarchy and infers that an indi-
individual of one class is equivalent to an individual of another class:

If C1 is an equivalent class of C2
and E has the type C2
then E has the type C1.

The next rule is used for completion of the rule set and states that class equivalence is reflexive:

If C1 is an equivalent class of C2
and E has C1 whose value is V
then E has C1 whose value is V.

The subsequent three rules guarantee that all occurrences of equivalent classes in either subject, predicate, or object position of a statement are replaced:

If C1 is an equivalent class of C2
and E has C1 whose value is V
then E has C2 whose value is V.

If C1 is an equivalent class of C2
and C2 has the property P
whose value is V
then C1 has P whose value is V.

If C1 is an equivalent class of C2
and E has the property P
whose value is C1
then E has P whose value is C2.

Property equivalence. Similar to class equivalence, we can formulate statements and rules in CNL that deal with property equivalence.

The subsequent two statements are straightforward and state symmetry and transitivity of equivalent properties:

‘equivalent property’ has the type ‘symmetric property’.
‘equivalent property’ has the type ‘transitive property’.

The following two rules deal with property hierarchies and reflexivity of equivalent properties:

If P1 is an equivalent property of P2
and E has P1 whose value is V
then E has P1 whose value is V.

If P1 is an equivalent property of P2
and E has P2 whose value is V
then E has P2 whose value is V.

and finally the next two rules take care of the replacement of equivalent properties in the subject and object position of a statement:

If P1 is an equivalent property of P2
and P2 has P whose value is V
then P1 has P whose value is V.

If P1 is an equivalent property of P2
and E has the property P
whose value is V
then E has P whose value is P2.

5.3 Property Restrictions

So far, we have seen how to restrict the range and the domain of properties in a global way. The following rule allows to set a local range restriction on a property taking both the property and the domain of a statement into account:

If E has a property P whose value is V
and R has the type ‘range restriction’
and R is on P
and R has all values from C
then E has the type R.

The rule ensures, for example, that if ‘Nora’ has the property ‘teach’ whose value is ‘Kylie’ and the class ‘academic’ has the type ‘range restriction’, and the restricted property is ‘teach’, and all values are from the domain ‘student’, then ‘Nora’ has the type ‘academic’. In brief, the rule licenses the inference: if Nora teaches Kylie and Kylie is a student then Nora is an academic.

6 Evaluation

Given the rule set introduced in the previous section, the terminological knowledge presented in Section 3.2, and the assertional knowledge in Section 3.1, SATCHMO can generate a satisfying model.

Since this model consists of terms of the form

\[ \text{term}([\text{\{'John','Smith'\}}, \text{[rdf:type]}, \text{[ex:student]})] \]

it is straightforward to extract all entailed assertional statements in CNL:

John Smith is a student.
Kylie Miller is a student.
Nora Yuen is a scientist.
Nora Yuen is a researcher.
Nora Yuen is an academic.
Nora Yuen is a professor.
Nora Yuen instructs Kylie Miller.
Nora Yuen teaches Kylie Miller.
Nora Yuen drills Kylie Miller.

and all entailed terminological statements (only a small subset of them is displayed here):

‘linguist’ is a subclass of ‘scientist’.
‘drills’ has the type ‘object property’.
‘drills’ is a subproperty of ‘instructs’.
‘is drilled by’ is the inverse of ‘drills’.
‘is drilled by’ has the type ‘object property’.

The generated model can now be used to answer questions in CNL, for example:

Who instructs Kylie?
Does Nora teach Kylie Miller and instruct John?
What does Nora do?
What type does John Smith have?

Questions are first translated into SATCHMO format, solution(s) are looked up in the model, and answers generated in CNL.

7 Conclusions

In this paper I presented a CNL that can be used to express the same sort of knowledge as the Web Ontology Language OWL Lite—but in a “seemingly informal” notation. In contrast to OWL Lite that does not have direct rule support, I showed how a complete rule set for reasoning with the assertional and terminological knowledge can be specified in CNL. Statements and rules in CNL can be automatically translated into a format that can be processed by a model builder. The model builder generates all entailed statements in CNL and allows for question answering over the generated model. The writing of statements and rules in CNL is supported by a look-ahead text editor. The user does not need to worry about the syntactic rules of the CNL and is guided while writing a specification. In summary, the take-home message is: CNL can replace an RDF-based ontology language, express rules for reasoning, and empower people to work in cooperation with computers.

Acknowledgments

The research reported here is supported by the Australian Research Council, Discovery Project DP0449928. The authors would also like to thank to all the folks at the Centre for Language Technology at Macquarie University for many valuable comments.

References


