

Creating and Querying Linguistically Motivated Ontologies

Rolf Schwitter

Centre for Language Technology
Macquarie University
Sydney NSW 2109, Australia
Email: schwitt@ics.mq.edu.au

Abstract

This paper argues that a formal ontology (in our case a description logic knowledge base) should be created in a linguistically motivated way so that it can be queried easily by non-specialists. This can best be achieved by using a strict naming convention that is based on those linguistic expressions that occur in the application domain for which the ontology will be created. We will see that ABox and TBox statements that closely follow this naming convention can be written directly in a controlled natural language and that the same controlled natural language can be used to query the description logic knowledge base. Both ABox and TBox statements written in controlled natural language are translated automatically into the Knowledge Representation System Specification (KRSS) syntax and questions are translated into RacerPro's new query language nRQL and answered over the description logic knowledge base. Using a controlled natural language as a high-level interface language abstracts away from any formal notation and allows for true collaboration between humans and machines.

Keywords: ontology design, controlled natural languages, question answering, human-computer interfaces

1 Introduction

Formal languages are difficult to understand and use by non-specialists – ontologies are no exception in this respect. However, ontologies have a rather special status since their role is to specify a vocabulary with which assertions and queries are exchanged not only between machines but also between humans and machines. In the ideal case, ontologies should be human-readable as well as machine-processable since they are agreements among all participants to use a common vocabulary in a specific domain (Gruber 1993).

Recently, the use of machine-oriented controlled natural languages has been suggested to create ontologies in a human-readable and machine-processable way (Schwitter & Tilbrook 2004). These controlled natural languages look like natural language but they are in fact formal languages “in disguise”. They have a formal syntax and semantics and can be unambiguously translated into an existing formal language, for example into a version of description logics or into first-order logic.

Copyright ©2008, Australian Computer Society, Inc. This paper appeared at the Knowledge Representation Ontology Workshop (KROW 2008), Sydney, Australia. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 90. Thomas Meyer and Mehmet A. Orgun, Eds. Reproduction for academic, not-for profit purposes permitted provided this text is included.

There exists an entire stream of research that investigated the usefulness of controlled natural languages for authoring and verbalising description logic-based ontologies. For example, (Schwitter & Tilbrook 2004, 2006) discuss the bidirectional mapping between the controlled natural language *PENG* and various subsets of the web ontology language OWL. This work built the foundation for *Sydney OWL Syntax* – a proposal of a controlled natural language syntax for OWL 1.1 (Cregan et al. 2007). In (Bernardi et al. 2007) a categorial grammar is introduced that translates the controlled natural language *Lite Natural Language* into the description logic DL-Lite, a tractable fragment of OWL 1.1. This fragment is expressive enough to deal with UML diagrams and relational databases. In (Hart et al. 2007) the controlled natural language *Rabbit* is presented which focuses on the knowledge acquisition process and requires that a domain expert and a knowledge engineer work in cooperation to create an OWL ontology. In (Kaljurand 2007) a bidirectional interface to OWL is discussed where a subset of the controlled natural language *Attempto Controlled English* is used for the purpose of authoring and verbalising OWL ontologies. The three controlled natural languages *Sydney OWL Syntax*, *Rabbit*, and *Attempto Controlled English* are compared in (Schwitter et al. 2008) and a number of requirements to an OWL compatible controlled natural language are put forward.

There exists another stream of research that studied the usefulness of controlled natural language and unrestricted natural language as ontology query languages. For example, in (Bernstein et al. 2004) a controlled language-based query interface is introduced that guides the writing process of questions using predictive interface techniques, and in (Bernstein et al. 2005) the authors show that this type of interface performs better than formal query languages. There exist also a number of systems that support the use of unrestricted natural language for answering questions over ontologies (Lopez et al. 2006, Kaufmann et al. 2006, Mithun et al. 2007). However, these unrestricted approaches suffer from similar problems to database systems with natural language interfaces (Copestake & Sparck-Jones 1990, Androutsopoulos et al. 1995, Popescu et al. 2004) because the end-users both over- and undershoot the system's capabilities since they don't know where the system's boundaries are and what kind of queries are supported. It is an ongoing debate which approach is most convenient for end-users (Reichert et al. 2005, Kaufmann et al. 2007), and it is not surprising that the performance of a natural language query interface to a description logic knowledge base depends highly on the quality and choice of the vocabulary in the knowledge base (Kaufmann et al. 2007).

In this paper, we will bring these two research streams closer together and argue that a description

logic knowledge base should be created in a **linguistically motivated** way in order to support question answering in an optimal way. We will show that a controlled natural language can be used for making terminological and assertional statements and that the structure of these statements is related in a systematic way to the controlled natural language used for expressing questions.

The rest of this paper is structured as follows: In Section 2, we will give a brief introduction to description logics and introduce the main constructors that are usually used to build a description logic knowledge base. In Section 3, we will study the relationship between the terms used in existing description logic ontologies and natural language expressions and make some recommendations about how to use linguistic expressions in an optimal way in ontologies. In Section 4, we will show that a description logic knowledge base can be specified in controlled natural language and introduce a notation that uses reification of relations in order to deal with n-ary relations in description logics. In Section 5, we look at the kind of questions that are supported by a state of the art description logic reasoner and suggest controlled natural language renderings for these questions. In Section 6, we will discuss the architecture of a controlled natural language processor that relates the structures of statements and questions written in controlled natural language in a systematic way to each other and translates these sentences into the input format of a description logic reasoner; we will also show that the same language processor can be used to generate answers to questions in controlled natural language. Finally, in Section 7, we will summarise the benefits of our approach and point to a number of further research challenges.

2 Description Logics (DLs)

DLs are a family of knowledge representation languages that can be understood as decidable fragments of first-order logic (Baader et al. 2003). DLs – such as the web ontology language OWL DL (Horrocks et al. 2003) – play an important role in the Semantic Web architecture because they are decidable and allow for a clear separation between the terminological information and the data.

A DL knowledge base usually consists of two components: a terminological component (= TBox) and an assertional component (= ABox). The TBox asserts general facts about concepts (= unary predicates) and roles (= binary relations) through declarations while the ABox asserts specific facts about individuals in an application domain. For example, a TBox might contain concept definitions like (1) and general inclusion axioms like (2):

1. (equivalent academic_staff (or professor senior_lecturer lecturer))
2. (implies student (or graduate_student undergraduate_student))

and the ABox might contain concept assertions like (3) and role assertions like (4):

3. (instance david_miller undergraduate_student)
4. (related david_miller comp101 attend)

Note that we use the Knowledge Representation System Specification (KRSS) syntax (Patel-Schneider & Swartout 1993) here in order to express these and all subsequent axioms in a compact way.

The expressivity of a DL depends on the set of constructors that can be used for defining concept terms from concept names and role names. Some common constructors include logical constructors: *intersection*, *union*, and *complement*; and *quantified role restriction* like (5), and *number restriction* like (6):

5. (instance kylie_jones (some attend unit))
6. (instance comp101 (exactly 22 student))

Other constructors are used to characterise roles and for enhanced reasoning with roles, for example: *inverse*, *transitivity*, and *functionality*. For instance, (7) defines a role with a domain and a range restriction and provides a name for the inverse of the defined role:

7. (define-primitive-role teach :inverse is_taught_by :domain academic_staff :range unit)

In addition to these features, some DLs provide extensions for algebraic reasoning over concrete domains, for example:

8. (instance david_miller (= age 21))

Here the age of an individual is specified with the help of a concrete domain predicate (=) and a concrete domain attribute (age) that is of type cardinal.

3 Linguistic Structures in Ontologies

A DL ontology is essentially a logical theory that specifies a conceptualisation of a specific part of the world – in our case of an university. Since the aim of an ontology is to make domain assumptions explicit and to establish a common understanding of the structure of information in a particular domain, it is important that all participants are able to use the same vocabulary in order to make assertions and ask queries in a way that is consistent with the logical theory. It should be possible for a non-specialist to relate the terms used in the ontology to the entities in the application domain. That means the terminology should be easy to use and understand by humans so that humans and machines can cooperate in the best possible way.

It seems intuitive to choose a naming convention for creating an ontology that is close to the linguistic expressions that occur in a particular domain. However, this is not always the case and a wide diversity and fragmentation of naming schemes can be found in real-world ontologies (Schober 2007). This is because knowledge engineers can use in principle any character sequence for naming entities and often make cost/accuracy trade-offs when creating an ontology. For a machine it does not make any difference whether a concept or a role is labeled in a way that can be understood by a human or not. Here is an extreme example:

9. (implies C1 (and C2 (some R (or C3 C4)))) (or C3 C4) (implies C5 (and C1 (some R C3))) (implies C6 (and C1 (some R C4)))

The labels that occur in this excerpt make it difficult for a human reader to understand what the single concept and role names stand for, how they relate to each other, and how they refer to the entities in

the application domain for which the ontology is designed. It is much easier to understand what is going on if the terms in the ontology communicate the intended meaning using natural language-like expressions, for example:

- ```
10. (implies person (and human (some
 has_gender (or female male))))
 (disjoint female male)
 (implies woman (and person (some
 has_gender female)))
 (implies man (and person (some
 has_gender male)))
```

Existing ontologies sometimes combine natural language expressions with non-linguistic artifacts. But there is a clear tendency to use linguistic patterns in ontologies. (Mellish & Sun 2005) collected 882 ontology files encoded in OWL and analysed the linguistic structure of concept and role names. They found that 72% of concept names ended with an English noun, 30% of concept names consisted entirely of noun sequences (various forms of compound nouns), and 14% contained no recognised word. The structure of role names showed a broader variety of patterns but 65% of these patterns started with a verb.

Given these results, we suggest using those patterns that occur most frequently in existing ontologies in order to establish a linguistically motivated naming convention. In particular, we suggest using

- nouns and compound nouns as concept names (e.g. `student` and `undergraduate_student`);
- transitive verbs and auxiliary verb-noun constructions as role names (e.g. `take`, `consist_of`, `is_student_of`, `has_student`);
- normalised forms of proper nouns as identifiers for individuals (e.g. `david_miller`, `comp101`).

Following (Schober 2007), we additionally recommend to use an underscore (`_`) to delimit words in compound terms since this separator is closer to natural language than CamelCase; to replace homonyms by an alternative word form since they create confusion; to resolve abbreviations and acronyms in names and include them as synonyms in the ontology.

#### 4 Controlled Natural Languages (CNL)

In the last section, we have argued that using a linguistically motivated naming convention can improve the readability and usability of an ontology for non-specialists. But also knowledge engineers who have to maintain and extend an ontology for new business needs can benefit from naming conventions since well-chosen naming conventions can enhance clarity, avoid the introduction of faults, and make it easier for subsequent generations of analysts to understand what the ontology is designed for. A linguistically motivated naming convention can directly be applied when creating an ontology with an ontology editor although current ontology authoring tools do not actively enforce naming conventions.

Adopting a common naming convention is a good strategy to improve the quality of an ontology but we can even go a step further towards natural language and express ABox and TBox statements of an ontology completely in a controlled natural language (CNL) and then translate these statements directly into the input language of a DL reasoner. That this can be done has been shown in previous work (Schwitter et al. 2008). Therefore, we will give here only a brief overview in order to discuss the underlying

design principles and illustrate how such CNL statements look like before we focus on the structure of CNL questions and the relationship between CNL assertions and questions.

#### 4.1 Terminological Statements in CNL

TBox statements express the intensional knowledge about a domain in form of a terminology. This terminological information is static, and it is more likely that a knowledge engineer will write and modify such statements than a domain specialist. Nevertheless, domain specialists need to be able to read, understand and validate this information with respect to an application domain. A CNL can help in this respect since it provides a high-level interface to a formal language that is potentially difficult to understand by an end-user. For example, a concept definition like (1) – see Section 2 – can be expressed as (11) in CNL, and a general inclusion axiom like (2) as (12):

- ```
11. Every academic staff is defined as
    a professor or a senior lecturer or
    a lecturer.

12. Every student is a graduate student
    or is an undergraduate student.
```

The TBox statement (11) makes it explicit that this statement is a definition that consists of a set of necessary and sufficient conditions. In contrast to (11), the TBox statement (12) does not speak about a definition since it only states necessary conditions for a primitive concept.

The definition of primitive roles in CNL requires the use of meta-information in order to speak about the features of a role. The use of variables makes it possible to speak about various features of a role in a very compact way in CNL. Note that variables are not a specific characteristic of a formal language; variables are also used in natural language texts as a look into any undergraduate maths textbook proves. Here is the CNL rendering of the primitive role definition (7) that uses two variables to signal the domain and range restrictions of the role:

- ```
13. X teaches Y is defined as Y is taught
 by X and X is an academic staff and Y
 is a unit.
```

Note that there is no need to use additional constructors such as *inverse*, *domain*, and *range* as in (7) since the CNL language processor can figure out the type of these restrictions from the linguistic structure.

#### 4.2 Assertional Statements in CNL

In contrast to TBox statements, ABox statements are much more dynamic in nature and more likely to be used by non-specialists. ABox statements rely on the vocabulary defined in the TBox. In the simplest case, ABox statements can be expressed via an auxiliary or a transitive verb in CNL, for example, the concept assertion (3) can be expressed as (14) and the role assertion (4) as (15):

- ```
14. David Miller is an undergraduate
    student.

15. David Miller attends COMP101.
```

The following two examples (16) and (17) illustrate how the quantified restriction in (5) and the number restriction in (6) can be expressed in CNL:

- ```
16. Kylie Jones attends some unit.
```

17. COMP101 has exactly 22 students.

The subsequent rendering (18) shows how the concrete domain example introduced in (8) can be expressed in CNL:

18. David Miller's age is 21.

In summary: all these ABox statements are based on the following simple functional structure:

19. Subject Verb Object

As we will see later, coordination is allowed in object position but not in subject position since this would introduce ambiguity. However, nouns that occur in the subject position can be modified as the example in (18) illustrates.

### 4.3 From Binary to N-ary Relations

Most DL languages only support binary relations and at first glance it looks like only very simple CNL statements can be captured by this logical framework. However, it is often necessary (and more natural) to describe relations among more than two individuals in one statement. This can be achieved via reification of binary relations (Noy & Rector 2006). For example, the ABox statement:

20. David Miller takes COMP101 on Monday at 11am in the Lincoln Building.

can be represented in DL via a concept assertion consisting of a new individual that stands for a reified relation (= verbal event) and a set of role assertions that link the individual of the reified relation with its participants:

```
21. (instance e1 take)
 (related e1 david_miller has_agent)
 (related e1 comp101 has_theme)
 (related e1 monday has_day)
 (related e1 1100 has_hour)
 (related e1 lincoln_building
 has_location)
```

Here the new individual `e1` stands for the reified relation and is an instance of the `take` concept. The roles (`has_agent`, `has_theme`, `has_day`, `has_hour`, and `has_location`) link this new individual to their corresponding individuals and values. Of course, this approach requires that the `take` concept is available in the TBox, for example as a defined concept:

```
22. (equivalent take (or attend select))
```

and that all the relevant roles are also defined. For example, the declaration of the `has_theme` role looks as follows in our context:

```
23. (define-primitive-role has_theme
 :domain take :range unit)
```

Note that the concept term `take` occurs in this definition as a domain restriction and the concept term `unit` as a range restriction.

## 5 DL Reasoners

There exist a number of state of the art DL reasoners for querying a DL knowledge base, for example: FaCT<sup>++</sup> (Tsarkov & Horrocks 2006), Pellet (Sirin et al. 2007), and RacerPro (Haarslev & Möller 2003, Wessel & Möller 2006). These reasoners usually implement a tableau-based decision procedure for general TBox reasoning (subsumption, satisfiability, and classification) and offer support for ABox reasoning (retrieval and conjunctive queries). Pellet and RacerPro both support a subset of SPARQL (Prud'hommeaux & Seaborne 2008) for answering conjunctive ABox queries.

### 5.1 RacerPro and nRQL

RacerPro is a knowledge representation system that implements the expressive description logic  $ALCQHIR_R + (D^-)$ . This is the basic description logic  $ALC$  augmented with qualifying number restrictions, role hierarchies, inverse roles, and transitive roles. In addition to these basic features, RacerPro also provides facilities for algebraic reasoning including concrete domains and implements most of the functions specified in the KRSS specification (see (Racer Systems 2007) for details).

The new RacerPro query language (nRQL) is a query language for RacerPro's concept language and supports – among other things – strong negation, negation as failure, and numeric constraints. RacerPro translates SPARQL queries into nRQL queries. nRQL is a more expressive query language than SPARQL. In contrast to SPARQL, nRQL uses description logic reasoning and does not work on the syntactic level of triples but on the level of semantic models (Racer Systems 2007).

In the following, we will first show how the syntactic structure of nRQL queries looks like and discuss then the main types of queries that are supported by RacerPro and provide CNL renderings for these queries.

### 5.2 Queries in nRQL

An nRQL query consists of a query head and a query body. The query head specifies the format of the answer and the query body contains (unary or binary) query atoms that are used to specify retrieval conditions on the bindings of query variables. nRQL queries are either simple or complex. A simple nRQL query consists of a single query atom in the query body. A complex nRQL query consists of two or more query atoms that are combined with the help of query body constructors (e.g. `and`, `union`, `neg`). For example, the simple nRQL query:

```
24. (retrieve (?1) (?1 comp101 take))
```

has a query variable (`?1`) as head and a binary query atom (`?1 comp101 take`) as body. This nRQL query can be expressed as a *wh*-question in CNL:

```
25. Who takes COMP101?
```

Note that the nRQL query in (24) can not be answered over a DL knowledge base that contains reified relations as introduced in (21). In order to answer the CNL question (25) over a DL knowledge base that contains reified relations, the verbal relation of the question needs to be reified too and this results in a complex (conjunctive) nRQL query of the form:

```
26. (retrieve (?2) (and (?1 ?2 has_agent)
 (?1 take) (?1 comp101 has_theme)))
```

The body of this complex query is composed of a query body constructor (`and`) that takes an unary query atom (`(?1 take)`) and two binary query atoms (`(?1 ?2 has_agent)` and `(?1 comp101 has_theme)`) as arguments.

### 5.3 nRQL Queries in CNL

The query language nRQL distinguishes a number of different types of queries that can be answered over a DL knowledge base. In the following, we will discuss CNL renderings for the most important types of these queries.

### 5.3.1 Queries about Concepts

In nRQL, queries about concepts can be used to retrieve all instances of a concept from an ABox. This type of queries can be expressed in CNL via a *wh*-question (27) or as an imperative construction (28):

27. Who is a student?
28. Find all students.

The nRQL query processing engine returns a set of tuples of the form `(((?1 david_miller)) ((?1 eva_barth)))` as answer to these questions. Note that the CNL question (28) makes the expected answer set explicit using the universal quantifier *all* in contrast to (27). A partial answer such as `((?1 eva_barth))` would be an acceptable answer for question (27) but not for question (28).

### 5.3.2 Queries about Roles

In nRQL, queries about roles can be used to retrieve role fillers from an ABox. Since these queries extract information from binary relations, there are in principle three different queries one might want to ask about the arguments of a binary relation. This corresponds to the following three CNL questions:

29. Who takes what?
30. Who takes COMP101?
31. What does David Miller take?

In (29) we are asking for information about the subject as well as the object of a relation, in (30) we are only asking for information about the subject, and in (31) only for information about the object. Note that the structure of (31) is different from (29) and (30). The query word *what* has been moved from the object position to the front of the sentence and is used there together with a *do*-operator.

### 5.3.3 Boolean Queries

In nRQL, simple Boolean queries can be used to check whether or not at least one individual exists for a specific concept, whether or not a specific individual exists for a particular concept, and whether or not two specific individuals stand in a particular relation. Boolean queries return either true or false.

In CNL, we can express Boolean queries via a *yes/no*-question. The following two questions (32) and (33) are equivalent and ask whether or not at least one individual exists that is a student:

32. Is there a student?
33. Does a student exist?

However, these two CNL questions are constructed in a different way: question (32) uses the verb *be* and an existential *there*, and question (33) uses a *do*-operator and the intransitive verb *exist*.

The following CNL question checks whether or not a specific individual belongs to a specific concepts:

34. Is David Miller a student?

and finally the subsequent CNL question (35) checks whether or not two specific individuals stand in a particular relation:

35. Does Lisa Brown teach Kylie Jones?

As we will see later, the answer to these questions is *yes* or *no* but one can also include the focus of the question in the answer, for example: *Yes, Lisa Brown does.*

### 5.3.4 Classical Negated Concepts and Roles

The nRQL query language allows for classical negated concepts and roles. For example, RacerPro can prove that somebody is not a student or that a student cannot teach a unit. Note that the negation of roles is only available in the nRQL query language but **not** in the concept language of RacerPro (in order to guarantee decidability). Here are two examples: in question (36) a concept is negated and in question (37) a role:

36. Who is not a student?
37. Who does not teach a unit?

The first question returns all instances that are not part of a concept and the second question returns all instances that are not a filler of a specific role with an existential restriction.

### 5.3.5 Implied Role Fillers

In nRQL only explicitly modeled role fillers are retrieved by default. However, a DL knowledge base can have logically implied role fillers whose presence is enforced in the logical models of the knowledge base. Let us assume that we asserted at some point that *Anna Grau is a professor* then Anna must have a PhD but this information might not be explicitly present in the DL knowledge base. In nRQL, it is possible to identify those individuals which have a certain role filler that is not explicitly modeled in the knowledge base using a negation as failure (**neg**) and a **project-to** operator. In CNL, we use the specific keyword *show* to trigger the same functionality:

38. Show which professor has a PhD.

This imperative construction returns individuals which have an implicit role filler which is not explicitly modeled in the knowledge base. Note that the answer set of (38) is different from that of question (39):

39. Find all professors who have a PhD.

The answer to (38) returns only the implicit instances while (39) returns all instances.

### 5.3.6 Concrete Domains

In nRQL, the concrete domain part of an ABox can be queried using concrete domain predicates. In CNL, we can use questions such as:

40. Who's age is 18?
41. Who is the oldest student?

for this purpose. Note that the relation between the adjective *oldest* and the concrete domain attribute *age* needs to be handled in the linguistic lexicon of the CNL processor.

### 5.3.7 Synonyms

In nRQL it is possible to check whether two instances are individual synonyms or not. In CNL we can check this using the expression *same as* in a question, for example:

42. Is Kylie Johns the same as Kylie Johns-Pedersen?
43. Is Kylie Johns not the same as Kylie Johns-Pedersen?

That means synonyms need not to be modeled in the linguistic lexicon. As we will see in the next section, only orthographic variants of content words are modeled in the linguistic lexicon but not synonyms.

### 5.3.8 Conjunctive Queries

A conjunctive query is a complex nRQL query that is constructed from unary and binary query atoms with the help of the query body constructor `and`. As soon as we work with a DL knowledge base that relies on reification of relations we will end up with conjunctive queries in the translation. The following CNL questions are complex (conjunctive) questions:

44. On what day does Kylie Johns take COMP101?
45. Where does Kylie Johns take COMP101?
46. Which student who takes COMP101 does not take MATH102?

Note that the processing of the two questions (44) and (45) relies on a DL knowledge base that reifies relations as discussed in Section 4.3 while (46) does not. However, all three CNL questions are complex questions and their translation results in three conjunctive nRQL queries.

## 6 The CNL Processor

The task of the CNL processor is to translate ABox and TBox statements as well as questions written in controlled natural language into the input format of the DL reasoner, and to generate answers in CNL from the output of the reasoner.

The CNL processor uses for this task a bidirectional unification-based grammar and a linguistic lexicon that contains syntactic constraints and information about the mapping between the linguistic expressions and the terms of the DL.

The kernel of the grammar is implemented as a definite clause grammar that can either run independently or be processed by a chart parser (Kay 1980) that stores processed substrings and hypothesizes about substrings. Storing processed substrings reduces redundancies during the parsing process, and the maintained hypotheses can be used to predict the next possible processing steps. As we will see later, the chart parser implements a meta-interpreter that generates lookahead information that can be used to support the writing process of the user in a predictive way.

The CNL processor first translates the input sentences into TPTP notation (Sutcliffe & Suttner 1998) and then further into the DL reasoner's target format. TPTP is a widely used notation for representing problems for automated theorem proving in first-order logic. The use of TPTP as an intermediate representation language has the advantage that we can extend the grammar of the CNL processor later and interface the processor easily with other (first-order) reasoning services. TPTP gives us the necessary flexibility to translate statements and questions into a suitable target format.

In our case, the CNL processor translates ABox and TBox statements via TPTP notation into RacerPro's KRSS format and adds the resulting formulas to the DL knowledge base. Questions too are first translated into TPTP notation but then transformed into nRQL syntax, and finally answered over the DL knowledge base using RacerPro's reasoning engine. The TPTP representation of a question is stored by the language processor and used as a template for answering questions. This is possible because the syntactic structure of questions and statements is related in a systematic way in CNL. That means that the same CNL grammar can be used as a processor and as a generator.

## 6.1 The CNL Lexicon

The lexicon of the CNL processor distinguishes between two main categories of words: content words and function words. Content words (nouns, adjectives, verbs, prepositions, and proper nouns) are closely related to concept names, role names, and names for individuals; while function words (conjunction, disjunction, negation, quantifiers, cardinals, and operators) are closely related to DL constructors – provided that we model the DL knowledge base in a linguistically motivated way. Function words are predefined in the lexicon and build the scaffolding of the CNL while content words can be added by the user to the lexicon.

In order to exclude ungrammatical sentences in CNL, the lexicon contains syntactic information that enforces, for example, number agreement between the subject and the verb of a sentence like (47) and the subject and the auxiliary verb of a question like (48):

47. David Miller takes COMP101.
48. When does David Miller take?

Below are the lexical entries for the proper nouns *David Miller* and *COMP101*:

49. `lex(cat:[pn],wf:['David','Miller'],sn:[],syn:[third,sg],sort:[person],fol:X^named(X,john_miller)).`
50. `lex(cat:[pn],wf:['COMP101'],sn:[['COMP',101],['COMP-101'],[comp,101],[comp-101]],syn:[third,sg],sort:[entity],fol:X^named(X,comp101)).`

These two entries contain syntactic information, sortal information and information that is required to generate the TPTP representation. The syntactic information deals with number agreement between the subject and the verb as explained above. There is also orthographic information available that deals with approved variants of the input, for example: *COMP-101* instead of *COMP101*, etc.

In order to resolve the ambiguity of prepositional phrases in CNL, the lexicon includes sortal information for nouns, proper nouns and prepositions so that the correct role name can be derived for a prepositional phrase, for example in:

51. David Miller takes COMP101 on Monday on South Campus.

the preposition *on* is ambiguous on the surface level of the CNL. The sortal information in the lexicon for the proper noun and for the preposition disambiguates the two prepositional phrases *on Monday* and *on South Campus*. This results in two different role names in the DL representation:

52. (related e2 monday has\_day)
53. (related e2 south\_campus has\_location)

Please note that the sortal information is not required, if we do not allow for this kind of prepositional phrases in the CNL – it is in theory possible to construct a reified version of a CNL sentence such as (51) that does not use prepositional phrases but offers the same expressivity, for example in the following way:

54. E1 is a take relation.  
E1 has David Miller as an agent.  
E1 has COMP101 as a theme.  
E1 has Monday as a day.  
E1 has South Campus as a location.

Let us assume that an ontology has already been constructed with the help of an ontology editor in a linguistically motivated way and that we are only interested in building an interface for querying the DL knowledge base. If that is the case, then we can automatically extract all concept names, role names, and names for individuals from the ontology using the following RacerPro functions:

- 55. (all-atomic-concepts)
- 56. (all-roles)
- 57. (all-individuals)

and populate the lexicon semi-automatically.

## 6.2 The CNL Grammar

The CNL grammar distinguishes three different modes: a TBox mode for processing terminological statements, an ABox mode for processing assertional statements and for generating answers to questions, and a query mode for processing questions. The grammar rules of the ABox mode are bidirectional (that means they can be used to analyse and generate statements) and some of these grammar rules can also be used to process questions.

## 6.3 Processing Statements

Below is an example of an ABox grammar rule that gives a high-level overview about how sentences written in CNL are parsed and translated during the parsing process into TPTP formulas:

- ```
58. s0(mode:M,
      fol:LF1,
      gap:G1-G3,
      para:P1-P3,
      ant:A1-A3) -->
n3(mode:M,
   syn:[subj,Per,Num],
   sort:_,
   fol:LF2^LF1,
   gap:[]-G2,
   para:P1-P2,
   ant:A1-A2),
v3(mode:M,
   syn:[fin,Per,Num,_,_,_],
   fol:E^LF2,
   gap:G1-G3,
   para:P2-P3,
   ant:A2-A3).
```

This top-level grammar rule takes part in the translation of the following ABox statement:

- 59. David Miller takes COMP101 on Monday.

The grammar rule (58) breaks this statement into a noun phrase *David Miller* and into a verb phrase *takes COMP101 on Monday* and combines the logical form of the noun phrase with the logical form of the verb phrase in a compositional way. Each word form in this sentence is associated with a partial logical form in the lexicon; these partial logical forms are merged (via unification) during the parsing process into a complete TPTP formula for the input sentence. The grammar rule contains various feature structures that are used for syntactic, semantic, and pragmatic purposes; one of these feature structures (*syn*) guarantees for example that the noun phrase agrees in person and number with the verb phrase; other feature structures are used to deal with the different types of sentences (*mode*), the composition of the logical form

(*fol*), discontinuous constituents in the input string (*gap*), the construction of a paraphrase (*para*) that makes the interpretation of the sentence clear, and the collection of noun phrase antecedents (*ant*) for anaphora resolution.

The output of the grammar is a logical formula in TPTP format, in our case the following one:

- ```
60. input_formula(university,axiom,
 (? [A]: (named(A,david_miller) &
 (? [B]: (property(B,has_agent,A) &
 (event(B,take) & contemp(B,u)) &
 (? [C]: (named(C,comp101) &
 property(B,has_theme,C) &
 (? [D]: (timex(D,monday) &
 property(B,has_day,D)))))))).
```

This logical formula is then further translated by the CNL processor into RacerPro's KRSS notation:

- ```
61. (instance e1 take)
    (related e1 david_miller has_agent)
    (related e1 comp101 has_theme)
    (related e1 monday has_day)
```

Note that the TPTP representation (60) contains additional information (*contemp(B,u)*) about the time when the utterance occurred since the CNL processor can handle tense (e.g. present tense and past tense). This information does not appear in the KRSS representation since DL can not deal with this kind of information and therefore all verb forms must be in present tense in a strict DL setting. However, the information about tense is potentially useful in other logical frameworks.

6.4 Processing Questions

The CNL processor distinguishes between *wh*-questions, *yes/no*-questions, and imperative constructions. It is well-known that the structure of questions in natural language is related in a systematic way to declarative sentences. For example, the declarative sentence (59) has the following functional structure:

- ```
62. Subject: (David Miller)
 Verb: (takes)
 Object: (COMP101)
 Modifier: (on Monday)
```

In the case of *wh*-questions, the query word is related to one of these functional elements. For example in:

- 63. On what day does David Miller take COMP101?

a constituent in modifier position has been replaced by a query expression (*on what day*) and this query expression has been moved to the front of the sentence where it functions as a filler and is used together with a *do*-operator letting a gap behind:

- ```
64. [On what day]_filler does David Miller
    take COMP101 [ ]_gap?
```

When the CNL grammar processes (63), the query expression which acts as a filler for this gap is moved back into its original position. Schematically, the result of this movement process looks as follows:

- ```
65. Subject: (David Miller)
 Verb: (takes)
 Object: (COMP101)
 Modifier: (on what day)
```

The TPTP representation for the question is constructed during the parsing process. That means at the same time when the syntactic movement of the query expression takes place. As already explained in the previous section, each word form is related to a partial logical form in the linguistic lexicon. For example, the processing of the query expression *on what day* triggers the following partial logical form:

```
66. (timex(F,G) & property(C,has_day,F))
```

and the processing of the entire question (63) results in the subsequent conjunctive query which contains the logical form for the query expression as a part:

```
67. input_formula(university,conjunctive_query,(
 (? [A]: (named(A,david_miller) &
 (? [B]: (named(B,comp101) &
 (? [C]: ((property(C,has_agent,A) &
 (event(C,take) &
 (property(C,has_theme,B) &
 contemp(C,u)))) & (timex(F,G) &
 property(C,has_day,F)))))))
=> answer(F))).
```

The CNL processor takes this TPTP formula and transforms it into a conjunctive nRQL query:

```
68. (retrieve (?2) (and (?1 david_miller
 has_agent) (?1 take) (?1 comp101
 has_theme) (?1 ?2 has_day)))
```

Note that *yes/no*-questions are treated in a similar way as *wh*-questions. The CNL processor first generates a normalised TPTP representation that looks similar to the representation of a declarative sentence, and then translates this representation into a Boolean conjunctive query, for example (69) into (70):

```
69. Does David Miller take COMP101?

70. (retrieve nil (and (?1 david_miller
 has_agent) (?1 take) (?1 comp101
 has_theme)))
```

The TPTP representation of a question is stored by the CNL processor and can be used as a starting point to answer questions as we will see in the next section.

## 6.5 Generating Answers

The relationship between declarative sentences and questions can be used for generating answers to questions in CNL. Internally, all TPTP formulas are annotated with syntactic information during the parsing process. For example, the TPTP formula (67) contains additional annotations (#) for all predicates that have been derived from content words, for example:

```
71. event(B,take)#[inf,_,_,pres,no,no]
```

Once the DL reasoner comes back with an answer for a question, the stored TPTP representation of a question is transformed into a TPTP representation for a declarative sentence which contains an update of the relevant syntactic annotations, for example:

```
72. event(B,take)#[fin,3rd,sg,pres,no,no]
```

That means the CNL processor can now generate the correct verbal form (*takes - fin*) for the answer string:

```
73. David Miller takes COMP101 on Monday.
```

Since the DL reasoner deals with subsumption hierarchies and concept definitions, there is no need to encode this ontological information in the linguistic lexicon. We get this terminological information for “free” from the DL reasoner during question answering. For example, the following questions:

```
74. Who studies COMP101?
75. Which person studies COMP101?
76. Which student studies COMP101?
```

return all the same answers because the query word *who* is more general than *person* and *person* subsumes *student*.

## 7 Writing in CNL

The writing of DL statements and questions in controlled natural language can be supported with the help of a predictive text editor that guides the writing process in CNL (see (Thompson et al. 2005, Chintaphally et al. 2007) for an introduction). We have implemented such editing techniques in the past (Schwitter et al. 2003) which have some similarities to editing techniques used in programming language environments. The basic idea here is to process the grammar rules with the help of a chart parser. A chart parser stores information about well-formed substrings as well as information about hypotheses of substrings in a chart (Kay 1980, Gazdar & Mellish 1989) and avoids repetition of work by looking up substrings in the chart instead of recomputing them. Complete and incomplete analyses are stored as so-called edges in the chart. Edges that correspond to partially recognized substrings are said to be active, while inactive edges represent completely recognised substrings. In our case, an active edge is a term of the form:

```
77. edge(Number,Pos1,Pos2,Category1,
 Found,[Category2|Categories]).
```

This term consists of the actual sentence number (**Number**) which serves as an index, a start position (**Pos1**) and an end position (**Pos2**) of a substring, the category (**Category1**) on the left-hand side of the grammar rule, a list of categories that have been found (**Found**) on the right-hand side of the grammar rule, and a list of remaining categories to be found (**[Category2|Categories]**). Chart parsing allows us – after processing of each new word form – to search through the active edges in the chart in order to collect the categories of those word forms that can follow the current input string.

Let us assume that the user is working in the query mode and plans to enter the following question into the text editor:

```
78. Where does David Miller take COMP101?
```

At the beginning, the text editor will display the initial lookahead information for questions. This information is grouped into three categories since the number of possible lexical entries for these categories is already large:

```
79. [wh-question | yes/no-question |
 imperative construction]
```

After clicking on the hypertext link for the *wh-question*, the editor will display the lookahead information that falls under this category:

```
80. [Who | What | Which | Where | ...]
```

The user can type (or select) one of these query words – in our case *where* – that is then sent to the CNL processor. The chart parser of the CNL processor initialises the chart, adds edges into the chart, and activates rules in order to expand the chart. This results in a set of new hypothesis that can be harvested for new lookahead information, in our case the editor will display the following function words:

81. [ *does* | *is* ]

After entering the operator *does*, the editor will display the following lookahead categories:

82. [ *determiner* | *proper noun* ]

At this point, the user has to make a decision whether he or she wants to enter a **determiner** or a **proper noun**. After entering the proper noun *David Miller* (or selecting it from a list of approved proper nouns) more lookahead information is displayed and this process continues until the structure of the question is complete.

Of course, an experienced user can switch off this lookahead facility and rely on the error messages that the parser generates if the user does not stick to the rules of the CNL. These predictive interface techniques guarantee that only sentences are added to the DL knowledge base and that only questions are used that conform 100% to the rules of the CNL.

## 8 Conclusions

This paper argued that a DL knowledge base should be constructed in a linguistically motivated way in order to support question answering in an optimal way. To achieve this the naming conventions used for constructing the ontology should be based on a set of well-defined linguistic patterns and on a terminology that occurs naturally in the application domain.

I showed that such a linguistically motivated naming convention makes it easier to create an ontology on the level of a machine-oriented controlled natural language. The controlled natural language can be used to express ABox and TBox statements as well as a query and feedback language. The presented approach has a number of attractive features: the controlled natural language looks like English and is easy to understand by humans and easy to process by machines; furthermore, the language is so precisely defined that it can be translated unambiguously into DL and thus is in fact a formal language. A logic-based grammar is used to process statements and questions, and the same grammar can run “backwards” taking logical formulas in TPTP notation as input and generate answers to questions in controlled natural language. The user does not need to learn and remember the rules of the controlled natural language since the writing process is supported with the help of predictive interface techniques.

It is important to note that not all parts of a DL knowledge base need to be constructed in controlled natural language. The aim of this paper was to illustrate that this is in principle possible. However, it might be the case that a knowledge engineer feels more comfortable using an ontology editor such as Protégé (Horridge et al. 2004) to develop the terminological knowledge for an application domain. As long as the knowledge engineer develops the TBox of this knowledge base in a linguistically motivated way – as sketched in this paper –, then this terminological knowledge can be combined with assertional knowledge expressed in controlled natural language. For some applications, it might be possible to extract this

assertional knowledge from existing textual information and represent it in controlled natural language so that it can be checked easily and modified by a human (and still be processed by a machine).

I am convinced that controlled natural languages are a promising approach for creating DL knowledge bases and that many applications can benefit from such a high-level interface language. I am currently investigating how controlled natural languages can be used as an interface language to the Semantic Web, as a language for expressing business rules, and as a query and alert language in a decision-support system.

## Acknowledgements

I would like to thank Anne Cregan, Alfredo Galdon, and Kevin Lee of NICTA’s Knowledge Representation and Reasoning program at the Kensington laboratory in Sydney – where I spent my sabbatical – for many fruitful discussions related to this work. I would also like to thank three anonymous reviewers for their constructive and valuable comments.

## References

- Androutsopoulos, I., Ritchie, G., Thanisch, P. (2007), Natural Language Interfaces to Databases – an Introduction, in: Journal of Language Engineering, 1(2), pp. 29–81.
- Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F. (2002), The Description Logic Handbook, Cambridge University Press.
- Bernardi, R., Calvanese, D., Thorne, C. (2007), Lite Natural Language, in: Proceedings of IWCS-7.
- Bernstein, A., Kaufmann, E., Fuchs, N.E. (2004), Talking to the semantic web – a controlled English query interface for ontologies, in: 14th Workshop on Information Technology and Systems, pp. 212–217.
- Bernstein, A., Kaufmann, E., Göhring, A., Kiefer, C. (2005), Querying ontologies: A controlled English interface for end-users, in: Proceedings of the 4th International Semantic Web Conference, pp. 112–126.
- Chintaphally, V.R., Neumeier, K., McFarlane, J., Cothren, J., Thompson, C.W. (2007), Extending a Natural Language Interface with Geospatial Queries, in: IEEE Internet Computing, pp. 82–85.
- Copestake, A., Sparck-Jones, K. (1998), Natural language interfaces to databases, in: Knowledge Engineering Review 5, pp. 225–249.
- Cregan, A., Schwitter, R., Meyer, T. (2007), Sydney OWL Syntax – towards a Controlled Natural Language Syntax for OWL 1.1, in: C. Golbreich, A. Kalyanpur, and B. Parsia (eds.), 3rd OWL Experiences and Directions Workshop (OWLED 2007), Vol. 258, CEUR Proceedings.
- Gazdar, G., Mellish, C. (1989), Natural Language Processing in Prolog. An Introduction to Computational Linguistics, Addison-Wesley.
- Gruber, T. (1993), A translation approach to portable ontologies, in: Knowledge Acquisition, 5(2), pp. 199–220.
- Haarslev, V., Möller, R. (2003), Racer: A Core Inference Engine for the Semantic Web, in: Proceedings of EON2003, pp. 27–36.

- Hart, G., Dolbear, C., Goodwin, J. (2007), Lege Felicitater: Using Structured English to represent a Topographic Hydrology Ontology, in: C. Golbreich, A. Kalyanpur and B. Parsia (eds), 3rd OWL Experiences and Directions Workshop (OWLED 2007), Vol. 258, CEUR Proceedings.
- Horridge, M., Knublauch, H., Rector, A., Stevens, R., Wroe, C. (2004), A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools. Edition 1.0, The University of Manchester, <http://www.co-ode.org/resources/tutorials>.
- Horrocks, I., Patel-Schneider, P.F., van Harmelen, F. (2003), From *SHIQ* and RDF to OWL: The Making of a Web Ontology Language, in: Journal of Web Semantics, 1(1), pp. 7–26.
- Kaljurand, K. (2007), Attempto Controlled English as a Semantic Web Language, PhD, Faculty of Mathematics and Computer Science, University of Tartu.
- Kaufmann, E., Bernstein, A., Zumstein, R. (2006), Querix: A Natural Language Interface to Query Ontologies Based on Clarification Dialogs, in: Proceedings of the 5th International Semantic Web Conference (ISWC 2006), Athens, GA, pp. 980–981.
- Kaufmann, E., Bernstein, A., Fischer, L. (2007), NLP-Reduce: A “naïve” but Domain-independent Natural Language Interface for Querying Ontologies, 4th European Semantic Web Conference (ESWC 2007), Innsbruck, Austria, pp. 1–2.
- Kaufmann, E., Bernstein, A. (2007), How Useful are Natural Language Interfaces to the Semantic Web for Casual End-users?, in: Proceedings of the 6th International Semantic Web Conference (ISWC 2007), Busan, Korea, pp. 281–294.
- Kay, M. (1980), Algorithm Schemata and Data Structures in Syntactic Processing, CSL-80-12, Xerox Parc, Palo Alto, California.
- Lopez, V., Motta, E., Uren, V. (2006), PowerAqua: Fishing the semantic web, in: The Semantic Web: Research and Applications, Lecture Notes in Computer Science, pp. 393–410.
- Mellish, C., Sun, X. (2005), The Semantic Web as a Linguistic Resource, in: 26th SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence, Peterhouse College, Cambridge, UK, December 12-14th.
- Mithun, S., Kosseim, L., Haarslev, V. (2007), Resolving Quantifier and Number Restriction to Question OWL Ontologies, in: Proceedings of the 3rd International Conference on Semantic, Knowledge and Grid, IEEE Computer Society, pp. 218–223.
- Noy, N., Rector, A. (2006), Defining N-ary Relations on the Semantic Web, W3C Working Group Note 12 April 2006, <http://www.w3.org/TR/swbp-n-aryRelations/>.
- Patel-Schneider, P.F., Swartout, B. (1993), Description-Logic Knowledge Representation System Specification from the KRSS Group of the ARPA Knowledge Sharing Effort, 1 November.
- Popescu, A.-M., Armanasu, A., Etzioni, O., Ko, D., Yates, A. (2004), Modern Natural Language Interfaces to Databases: Composing Statistical Parsing with Semantic Tractability, in: Proceedings of the 20th International Conference on Computational Linguistics (COLING), Article No. 141.
- Prud’hommeaux, E., Seaborne, A. (2008), SPARQL Query Language for RDF, W3C Recommendation 15 January 2008, <http://www.w3.org/TR/rdf-sparql-query/>.
- Racer Systems (2007), RacerPro User’s Guide, Version 1.9.2, Technical report, <http://www.racer-systems.com>.
- Reichert, M., Linckels, S., Meinel, C., Engel, T. (2004), Student’s Perception of a Semantic Search Engine, in: Proceedings of IEEE IADIS International Conference on Cognition and Exploratory Learning in Digital Age (CELDA), Porto, Portugal, pp. 139–147.
- Schober, D., Kusnierczyk, W., Lewis, S.E., Lomax, J., Members of the MSL, PSI Ontology Working Groups, Mungall, C., Rocca-Serra, P., Smith, B., Sansone, S.-A. (2007), Towards naming conventions for use in controlled vocabulary and ontology engineering, in: Proceedings of Bio-Ontologies SIG Workshop 2007.
- Schwitler, R., Ljungberg, A., Hood, D. (2003), ECOLE – A Look-ahead Editor for a Controlled Language, in: Proceedings of EAMT-CLAW03, May 15-17, Dublin City University, Ireland, pp. 141–150.
- Schwitler, R., Tilbrook, M. (2004), Controlled Natural Language meets the Semantic Web, in: S. Wan, A. Asudeh, C. Paris (eds.), Australasian Language Technology Workshop 2004, pp. 55–62.
- Schwitler, R., Tilbrook, M. (2006), Let’s Talk in Description Logic via Controlled Natural Language, in: Logic and Engineering of Natural Language Semantics 2006, (LENLS2006), Tokyo, Japan, June 5-6th.
- Schwitler, R., Kaljurand K., Cregan, A., Dolbear, C., Hart, G. (2008), A Comparison of three Controlled Natural Languages for OWL 1.1, 4th OWL Experiences and Directions Workshop (OWLED 2008 DC), Washington, 1-2 April.
- Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y. (2007), Pellet: A practical OWL-DL reasoner, in: Journal of Web Semantics, 5(2), pp. 51–53.
- Sutcliffe, G., Suttner, C.B. (1998), The TPTP Problem Library: CNF Release v1.2.1., in: Journal of Automated Reasoning, 21(2), pp. 177–203.
- Thompson, C.W., Pazandak, P., Tennant, H.R. (2005), Talk to Your Semantic Web, in: IEEE Internet Computing, 9(6), pp. 75–79.
- Tsarkov, D., Horrocks, I. (2006), FaCT++ Description Logic Reasoner: System Description, in: Proceedings of IJCAR 2006, LNAI 4130, Springer, pp. 292–297.
- Wang, C., Xiong, M., Zhou, Q., Yu, Y. (2007), PANTO: A Portable Natural Language Interface to Ontologies, in: Proceedings of the 4th European Semantic Web Conference (ESWC 2007), Innsbruck, Austria, pp. 473–487.
- Wessel, M., Möller, R. (2006), A Flexible DL-based Architecture for Deductive Information Systems, in: Proceedings of the FLoC’06 Workshop on Empirically Successful Computerized Reasoning, Seattle, USA, August 22, pp. 92–111.