Attempto Controlled English for Knowledge Representation

Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn

Department of Informatics & Institute of Computational Linguistics
University of Zurich, Switzerland
{fuchs,kalju,tkuhn}@ifi.uzh.ch
http://attempto.ifi.uzh.ch

Abstract. Attempto Controlled English (ACE) is a controlled natural language, i.e. a precisely defined subset of English that can automatically and unambiguously be translated into first-order logic. ACE may seem to be completely natural, but is actually a formal language, concretely it is a first-order logic language with an English syntax. Thus ACE is human and machine understandable. ACE was originally intended to specify software, but has since been used as a general knowledge representation language in several application domains, most recently for the semantic web. ACE is supported by a number of tools, predominantly by the Attempto Parsing Engine (APE) that translates ACE texts into Discourse Representation Structures (DRS), a variant of first-order logic. Other tools include the Attempto Reasoner RACE, the AceRules system, the ACE View plug-in for the Protégé ontology editor, AceWiki, and the OWL verbaliser.

1 Introduction

Traditionally human knowledge is presented informally, predominantly in natural language. Everybody knows and understands natural language that takes no extra learning effort, is highly expressive, and provides domain-specific words and phrases. Concerning knowledge representation the disadvantages of natural language are its ambiguity, vagueness and potential inconsistency. To represent knowledge in computers people use formal languages. These languages have a well-defined syntax and an unambiguous semantics, and support formal methods, specifically reasoning. However, many domain specialists are unfamiliar or uneasy with formal languages and formal methods. Furthermore, in order to express domain-specific knowledge in a formal language we need to bridge a conceptual distance. Thus there exists a conflict between the wish to use natural languages and the need to use formal languages. Controlled natural languages[1] have been proposed as a way to resolve this conflict.

Attempto Controlled English (ACE) is a controlled English, i.e. a precisely defined, tractable subset of full English that can automatically and unambiguously be translated into first-order logic. ACE seems completely natural, but is

[1] http://www.ics.mq.edu.au/~rolfs/controlled-natural-languages/

actually a formal language defined by a small set of construction and interpretation rules. One could say that ACE is a first-order logic language with the syntax of a subset of English. As a consequence, ACE is understandable both by humans and by machines. Most importantly, ACE texts can be read by anybody who knows English.

Like any formal language ACE has to be learned. Teaching the construction and interpretation rules takes — according to our experience — about two days. Of course, getting fluent with ACE takes longer. This process is supported by ACE tools that provide feedback like paraphrases and reasoning results. To relieve users of having to learn ACE's construction rules, we are developing a predictive editor that guides users in constructing syntactically correct ACE texts.

The Attempto Parsing Engine (APE) that embodies ACE's construction and interpretation rules translates ACE texts unambiguously into discourse representation structures (DRS). DRSs are a variant of first-order logic, and can be easily translated into any formal language equivalent to (a subset of) first-order logic. For the current version 6 of ACE, we developed an extended form of discourse representation structures that allows us to express complex linguistic features, for instance plurals and generalised quantifiers, in first-order logic, and that furthermore facilitates logical deductions on ACE texts.

As a DRS can get a model-theoretic or a proof-theoretic semantics, we can assign the same formal semantics, i.e. unique meaning, to the ACE text from which the DRS was derived. Thus, every ACE sentence is unambiguous, even if people may perceive the sentence as ambiguous in full English.

ACE is supported by a number of tools, predominantly the Attempto Parsing Engine (APE) already mentioned above. Besides translating an ACE text into discourse representation structures, APE also offers translations into various other forms of first-order logic, into OWL, SWRL, and into RuleML. Furthermore, APE can generate ACE paraphrases of DRSs derived from ACE texts.

To support automatic reasoning in ACE, we have developed the Attempto Reasoner (RACE). RACE can prove that one ACE text is the logical consequence of another one, and give a justification for the success or the failure of the proof in ACE.

Recently, ACE has found several applications within the semantic web. Therefore, we have developed translations of ACE into and from semantic web languages. Concretely, there are the translations ACE $\leftrightarrow$ OWL/SWRL and ACE $\rightarrow$ rules. Also, there are various tools that use these translations: AceWiki (uses ACE $\rightarrow$ OWL/SWRL), ACE View (uses ACE $\leftrightarrow$ OWL/SWRL), and AceRules (uses ACE $\rightarrow$ rules). The tool AceWiki combines controlled natural language with the ideas and technologies of the semantic web and with the concepts of wikis. AceWiki also incorporates a predictive editor that enables users to construct syntactically correct ACE sentences from a restricted vocabulary. ACE View is a plug-in for the ontology editor Protégé. Finally, AceRules is a forward chaining rule system that offers three different semantics.

Applications of ACE include software and hardware specifications, data base integrity constraints, agent control, legal and medical regulations, and ontologies. Furthermore, ACE served as natural language interface to semantic web languages like OWL, SWRL, RuleML, Protune, R2ML and as query language for MIT's Process Handbook.

In the following we give an overview of the language ACE, and then briefly present the tools Attempto Parsing Engine (APE), Attempto Reasoner (RACE), ACE View, AceRules, AceWiki, and OWL verbaliser.

2 Overview of ACE

This section is a brief introduction into ACE 6. For a full account please consult the ACE documentation found at the Attempto website[2].

Sections 2.1 to 2.6 describe the syntax of ACE 6, sections 2.8 to 2.10 summarise the handling of ambiguity, and section 2.11 explains anaphoric references.

2.1 Vocabulary

The vocabulary of ACE comprises

– Predefined function words (e.g. determiners, conjunctions, prepositions), and some predefined phrases (*there is/are, it is false that, ...*)
– Content words (nouns, verbs, adjectives, and adverbs)

2.2 Grammar

The grammar of ACE defines and constrains the form and the meaning of ACE sentences and texts. ACE's grammar is expressed as a small set of construction rules[3]. The meaning of ACE texts is defined by a small set of interpretation rules[4].

2.3 ACE Texts

An ACE text is a sequence of declarative sentences that can be anaphorically interrelated. Furthermore, ACE supports questions that allow users to interrogate the contents of an ACE text, and commands for the interactive control of agents.

Declarative sentences are categorised as simple sentences, and composite sentences.

2.4 Simple Sentences

A simple sentence describes that something is the case — a fact, an event, a state.

A customer inserts 2 cards.
The temperature is -2 °C.
At least 3 cards and exactly 2 codes are valid.

[2] http://attempto.ifi.uzh.ch
[3] http://attempto.ifi.uzh.ch/site/docs/ace_constructionrules.html
[4] http://attempto.ifi.uzh.ch/site/docs/ace_interpretationrules.html

Simple ACE sentences have the following general structure:

subject + verb + complements + adjuncts

Every simple sentence has a subject and a verb. Complements (direct and indirect objects) are necessary for transitive verbs (*insert something*) and ditransitive verbs (*give something to somebody*, *give somebody something*), whereas adjuncts (adverbs, prepositional phrases) are optional.

All elements of a simple sentence can be elaborated upon to describe the situation in more detail. To further specify the nouns *customer* and *card* of the first example sentence, we could add adjectives

A trusted customer inserts two valid cards.

possessive nouns and *of*-prepositional phrases

John's customer inserts 2 cards of Mary.

or variables as appositions

The customer X inserts 2 cards Y.

Other modifications of nouns are possible through relative sentences

A customer who is new inserts 2 cards that he owns.

We can also detail the insertion event, e.g. by adding an adverb

A customer inserts two cards manually.

or equivalently

A customer manually inserts two cards.

or by adding prepositional phrases, e.g.

A customer inserts two cards into a slot.

We can combine all of these elaborations to arrive at

John's customer who is new inserts 2 valid cards of Mary manually into a slot X.

2.5 Composite Sentences

Composite sentences are recursively built from simpler sentences through coordination, subordination, quantification, and negation. Note that ACE composite sentences overlap with what linguists call compound sentences and complex sentences.

Coordination by *and* is possible between sentences and between phrases of the same syntactic type.

A customer inserts a card and the machine checks the code.
There is a customer who inserts a card and who enters a code.
A customer inserts a card and enters a code.
An old and trusted customer enters a card and a code.

Note that the coordination of the noun phrases *a card and a code* represents a plural object.

Coordination by *or* is possible between sentences, relative clauses and verb phrases.

A customer inserts a card or the machine checks the code.
A customer owns a card that is invalid or that is damaged.
A customer inserts a card or enters a code.

Coordination by *and* and *or* is governed by the standard binding order of logic, i.e. *and* binds stronger than *or*. Commas can be used to override the standard binding order. Thus the sentence

A customer inserts a VisaCard or inserts a MasterCard, and inserts a code.

means that the customer inserts a VisaCard and a code or, alternatively a MasterCard and a code.

There are three constructs of subordination: *if-then* sentences, modality, and sentence subordination.

With the help of *if-then* sentences we can specify conditional or hypothetical situations, e.g.

If a card is valid then a customer inserts it.

Note the anaphoric reference via the pronoun *it* in the *then*-part to the noun phrase *a card* in the *if*-part.

Modality allows us to express possibility and necessity.

A trusted customer can insert a card.
A trusted customer must insert a card.
It is possible that a trusted customer inserts a card.
It is necessary that a trusted customer inserts a card.

Sentence subordination comes in various forms.

It is true that a customer inserts a card. (= A customer inserts a card.)
It is false that a customer inserts a card.
It is not provable that a customer inserts a card.
A clerk believes that a customer inserts a card.

Quantification allows us to speak about all objects of a certain class (universal quantification), or to denote explicitly the existence of at least one object of this class (existential quantification). The textual occurrence of a universal or existential quantifier opens its scope that extends to the end of the sentence, or in coordinations to the end of the respective coordinated sentence.

To express that all customers insert cards, we can write

Every customer inserts a card.

Alternatively, with exactly the same meaning

All customers insert a card.

This sentence means that each customer inserts a card that may, or may not, be the same as the one inserted by another customer. To specify that all customers insert the same card — however unrealistic that situation seems — we can write

There is a card that every customer inserts.

or, equivalently

A card is inserted by every customer.

To state that every card is inserted by a customer, we write

Every card is inserted by a customer.

or, somewhat indirectly

For every card there is a customer who inserts it.

Negation allows us to express that something is not the case, e.g.

A customer does not insert a card.
A card is not valid.

To negate something for all objects of a certain class one uses *no*

No customer inserts more than 2 cards.

or, equivalently, *there is no*

There is no customer who inserts more than 2 cards.

To negate a complete statement one uses sentence negation

It is false that a customer inserts a card.

2.6 Query Sentences

Query sentences permit us to interrogate the contents of an ACE text, data bases etc. ACE supports two forms of queries: *yes/no*-queries and *wh*-queries. Note that interrogative sentences need always a question mark at the end.

Yes/no-queries establish the existence or non-existence of a specified situation. If we specified

A customer inserts a card.

then we can ask

Does a customer insert a card?

to get a positive answer.

With the help of *wh*-queries, i.e. queries with query words, we can interrogate a text for details of the specified situation. If we specified

A new customer inserts a valid card manually.

we can ask for each constituent of the sentence with the exception of the verb.

Who inserts a card?
Which customer inserts a card?
What does a customer insert?
How does a customer insert a card?

Queries can also be constructed by a sequence of declarative sentences followed by one interrogative sentence. Here is an example.

A customer uses a card that is valid and that is owned by the customer. The customer has an account that is activated. The card belongs-to the account. What is the code of the card?

2.7 Commands

ACE also supports commands intended to be used in interactive environments. Some examples:

John, go to the bank!
John and Mary, wait!
Every dog, bark!
The brother of John, give the book to Mary!

A command always consists of a noun phrase (the addressee), followed by a comma, followed by an uncoordinated verb phrase. Furthermore, a command has to end with an exclamation mark.

2.8 Constraining Ambiguity

To constrain the ambiguity of full natural language, ACE employs three simple means

– some ambiguous constructs are not part of the language; unambiguous alternatives are available in their place
– all remaining ambiguous constructs are interpreted deterministically on the basis of a small number of interpretation rules
– users can either accept the assigned interpretation — shown for example in the paraphrase generated by APE — or they must rephrase the input to obtain another one

2.9 Avoidance of Ambiguity

Here is an example how ACE replaces ambiguous constructs by unambiguous constructs. In full natural language relative sentences combined with coordinations can introduce ambiguity, e.g.

A customer inserts a card that is valid and opens an account.

In ACE the sentence has the unequivocal meaning that the customer opens an account. This is reflected by the paraphrase

A card is valid. A customer inserts the card. The customer opens an account.

To express the alternative — though not very realistic — meaning that the card opens an account the relative pronoun *that* must be repeated, thus yielding a coordination of relative sentences.

A customer inserts a card that is valid and that opens an account.

with the paraphrase

A card is valid. The card opens an account. A customer inserts the card.

2.10 Interpretation Rules

However, not all ambiguities can be safely removed from ACE without rendering it artificial. To deterministically interpret otherwise syntactically correct ACE sentences we use a small set of interpretation rules. For example, if we write

The customer inserts a card with a code.

then *with a code* attaches to the verb *inserts*, but not to *a card*. However, this is probably not what we meant to say. To express that the code is associated with the card we can employ the interpretation rule that a relative clause always modifies the immediately preceding noun phrase, and rephrase the input as

A customer inserts a card that carries a code.

yielding the paraphrase

A card carries a code. A customer inserts the card.

or — to specify that the customer inserts a card and a code — as

The customer inserts a card and a code.

Another example. Adverbs can precede or follow the verb. To disambiguate the sentence

The customer who inserts a card manually enters a code.

where in full English *manually* could modify *insert* or *enter*, we employ the interpretation rule that the postverbal position has priority. This is exhibited in the paraphrase

There is a customer. The customer enters a code. The customer inserts a card manually.

2.11 Anaphoric References

Usually ACE texts consist of more than one sentence, e.g.

> *A customer enters a card and a code. If a code is valid then SimpleMat accepts a card. If a code is not valid then SimpleMat rejects a card.*

To express that all occurrences of *card* and *code* should mean the same card and the same code, ACE provides anaphoric references via the definite article, i.e.

> *A customer enters a card and a code. If the code is valid then SimpleMat accepts the card. If the code is not valid then SimpleMat rejects the card.*

During the processing of the ACE text, all anaphoric references are replaced by the most recent and most specific accessible noun phrase that agrees in gender and number, yielding the paraphrase

> *There is a customer X1. The customer X1 enters a card X2 and a code X3. If the code X3 is valid then SimpleMat accepts the card X2. If it is false that the code X3 is valid then SimpleMat rejects the card X2.*

where the variables *X1*, *X2*, and *X3* are introduced to clearly show the anaphoric references.

What does "most recent and most specific" mean? Given the sentence

> *A customer enters a red card and a blue card.*

then

> *The card is correct.*

refers to the second card, since it is "most recent" reference to *a card*, while

> *The red card is correct.*

refers to the first card, since it is "most recent" reference to *a red card*.

What does "accessible" mean? In accordance with standard English, noun phrases introduced in *if-then* sentences, universally quantified sentences, negations, modality, and subordinated sentences, and noun phrase preceded by the generalised quantifiers *less than* and *at most* cannot be used anaphorically in subsequent sentences. Thus for each of the sentences

> *If a customer owns a card then he enters it.*
> *Every customer enters a card.*
> *A customer does not enter a card.*
> *A customer can enter a card.*
> *A clerk believes that a customer enters a card.*
> *A customer enters less than 2 cards.*
> *A customer enters at most 2 cards.*

we cannot refer to *a card*, *less than 2 cards* or *at most 2 cards* with, for instance

The card is correct.

Anaphoric references are also possible via personal pronouns

A customer enters a card and a code. If it is valid then SimpleMat accepts the card. If it is not valid then SimpleMat rejects the card.

or via variables

A customer enters a card X and a code Y. If Y is valid then SimpleMat accepts X. If Y is not valid then SimpleMat rejects X.

Anaphoric references via definite articles and variables can be combined.

A customer enters a card X and a code Y. If the code X is valid then SimpleMat accepts the card X. If the code Y is not valid then SimpleMat rejects the card X.

Note that proper names like *SimpleMat* always refer to the same object.

2.12 Other Controlled Natural Languages

Traditionally, controlled natural languages fall into two major types: those that improve readability for human readers, and those that enable reliable automatic semantic analysis of the language. The first type of languages, for example ASD Simplified Technical English[5], Caterpillar Technical English, IBM's Easy English, are used in the industry to increase the quality of technical documentation. The second type of languages have a formal logical basis, i.e. they have a formal syntax and semantics, and can be mapped to an existing formal language, such as first-order logic. Thus, languages of the second type can be used for knowledge representation and reasoning.

Since ACE falls into the second type we will focus here only on other languages of the second type. Expressive and recently developed versions of controlled English include PENG [23, 24, 27, 28], Common Logic Controlled English [29], Boeing's Computer Processable Language [1], and E2V [21]. For an exhaustive list of controlled natural languages see footnote 1 and [20] that lists 32 languages altogether.

Rolf Schwitter's PENG[6] branched out from the research done on ACE. Therefore the designs of the two languages are quite similar. In recent years, more features have been added to ACE, making it both syntactically and semantically more expressive than PENG. Research on PENG, on the other hand, stresses the need for syntax-aware editing tools for controlled natural languages, and has focused on the development of ECOLE [26], a predictive text editor that guides the user in constructing only syntactically acceptable PENG sentences. A similar predictive editor is being developed in the AceWiki system (see section 3.5) for a subset of ACE.

Another ACE-like controlled English is John Sowa's CLCE [29, 30], which has been designed as a "human interface" to the ISO standard Common Logic[7].

[5] `http://www.asd-ste100.org`
[6] `http://www.ics.mq.edu.au/~rolfs/peng/`
[7] `http://cl.tamu.edu/`

However, there is only a partial specification of CLCE available, and a parser for CLCE has not yet been published.

CPL [1] is a controlled English developed at Boeing, and used experimentally for various purposes, e.g. to rephrase texts on chemistry. CPL is closer to everyday English than ACE, PENG, or CLCE, in the sense that it uses fewer strict rules and its interpreter is expected to “smartly” resolve various ambiguities. The interpreter also handles nominalizations, and guesses the word senses of nouns and verbs with the help of the WordNet[8] lexicon. Errors are handled by sophisticated error resolution. The result of parsing is a logical formula in a frame-based knowledge representation language Knowledge Machine [2] on which a reasoner can be applied.

E2V [21] is a fragment of English that corresponds to the decidable two-variable fragment of first-order logic ($\mathcal{L}^2$). Syntactically, E2V is a subset of ACE. However, its treatment of pronominal references is different — pronouns always refer to the closest noun in the syntax tree, and not to the closest noun in the surface order of words. This makes a difference if the preceding noun phrase contains a relative clause. This different treatment seems to be mainly motivated by better reasoning properties. In general, E2V, as well as its extensions have been developed to study the computational properties of certain linguistic structures. The intention of the authors has not been to develop a real-world knowledge representation language by adding features which would increase the usability of the language. Instead, only language features that introduce interesting computational problems have been added.

2.13 Decidablility of ACE

ACE texts are translated into a subset of first-order logic. Since first-order logic is not decidable the question arises whether ACE is decidable.

To answer this question, we rely on results of Ian Pratt-Hartmann and Allan Third (see [22]) who investigated the decidability of various fragments of English.

In their paper, Pratt-Hartmann & Third use the following abbreviations to describe English constructs: Cop (singular, existentially/universally quantified nouns, predicative adjectives, copula with and without negation), Rel (relative clauses), TV (transitive verbs without and with negation), DTV (ditransitive verbs without and with negation), RA (reflexive and non-reflexive pronouns as anaphors, resolution of anaphors to closest antecedent noun phrase in phrase structure), GA (reflexive and non-reflexive pronouns as anaphors, resolution of anaphors by coindexing pronouns and antecedent noun phrases).

From the results of Pratt-Hartmann & Third it follows that ACE is not decidable since it is a superset of the fragment Cop + Rel + TV + GA that Pratt-Hartmann & Third proved to be undecidable.

However, Pratt-Hartmann & Third identified 4 decidable fragments that are also fragments of ACE, namely

- Cop + TV + DTV
- Cop + Rel

[8] `http://wordnet.princeton.edu`

- Cop + Rel + TV
- Cop + Rel + TV + DTV

As [14] shows, there is one further decidable ACE fragment that differs from the ones identified by Pratt-Hartmann & Third. This ACE fragment can be translated into OWL 2 and vice versa (see section 3.3). Since OWL 2 is decidable, the respective ACE fragment is also decidable.

Interestingly, none of these decidable ACE fragments contains adverbs or prepositional phrases. Thus verbs cannot be readily modified.

The so-called AE subset of first order logic — that consists of formulas without functions symbols and with no universal quantifier occurring in the scope of an existential one — is decidable [5]. This means that any ACE sentence that can be mapped to an AE formula is also decidable. This defines a further decidable fragment of ACE that consists of simple sentences and *if-then* sentences with rather intricate restrictions on the use of negation and universal quantification.

As can be seen, each of the decidable fragments of ACE introduces some restrictions on the syntactic structures that can be used. It depends on the respective application whether or not this limitation of expressivity matters, and also on the willingness of the users to cope with syntactic restrictions. To relieve users of these considerations, the ACE reasoner RACE (see section 3.2) accepts all of ACE — of course with the exception of negation as failure that is outside of first-order logic — and controls undecidability by introducing a time-out.

3 ACE Tools

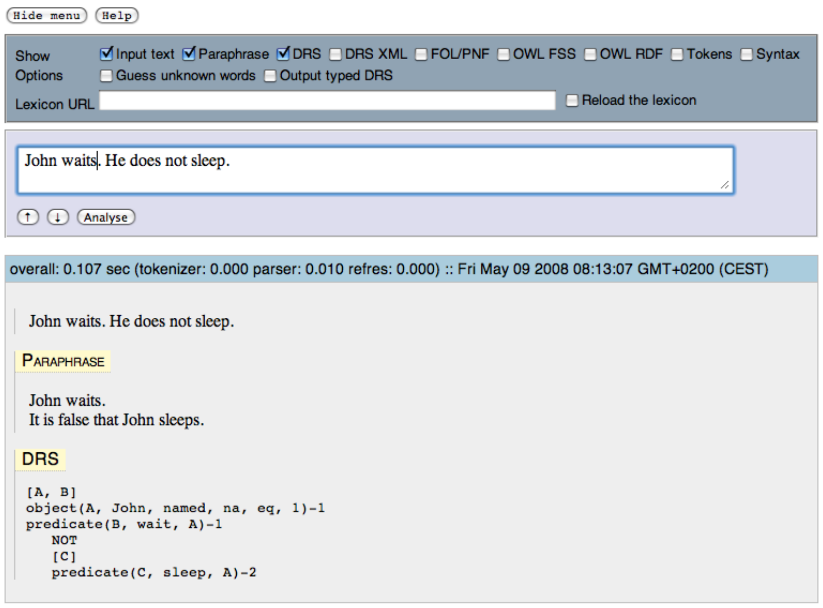
3.1 Attempto Parsing Engine APE

The Attempto Parsing Engine (APE) implements the ACE construction and interpretation rules in Prolog as a Definite Clause Grammar enhanced with feature structures. APE uses a built-in lexicon of function words and a basic lexicon of content words with approximately 100'000 entries. Users can import additional, e.g. domain specific, content word lexicons. Words found in user-imported lexicons take precedence over the same words found in the basic lexicon. Alternatively, users can let the ACE parser guess unknown words, or users can prefix unknown words by their word-class, for instance *n:kitkat, p:Thomas, v:google, a:trusted, a:undeviatingly.*

APE is available as open source under the LGPL license. Alternatively, users can access APE via a web service[9] or via a web client[10] that provides a graphical front-end to the web service.

APE takes an ACE text and optionally a user lexicon as input, and can generate a large number of various outputs: the tokens of the input text, various representations of the syntax tree of the input text, several representations of the DRS of the

[9] `http://attempto.ifi.uzh.ch/site/docs/ape_webservice.html`
[10] `http://attempto.ifi.uzh.ch/ape/`



[✓ Input text] [✓ Paraphrase] [✓ DRS] DRS XML FOL/PNF OWL FSS OWL RDF Tokens Syntax
Guess unknown words Output typed DRS

John waits. He does not sleep.

overall: 0.107 sec (tokenizer: 0.000 parser: 0.010 refres: 0.000) :: Fri May 09 2008 08:13:07 GMT+0200 (CEST)

John waits. He does not sleep.

Paraphrase

John waits.
It is false that John sleeps.

DRS

```
[A, B]
object(A, John, named, na, eq, 1)-1
predicate(B, wait, A)-1
   NOT
   [C]
   predicate(C, sleep, A)-2
```

Fig. 1. Screenshot of the APE web client translating a simple ACE text into a DRS

input text, different paraphrases of the input text derived from its DRS, several first-order representations of the DRS, and two syntaxes of OWL/SWRL derived from the DRS.

If parsing fails, APE generates warning and error messages that identify the cause and the approximate location of the problems encountered, and suggest remedies for the problems. An ACE text can be erroneous because the input contained an unknown word, the input violated the ACE syntax rules, and finally, the input was syntactically correct but anaphoric references could not be resolved.

APE is sufficiently fast and parses approximately 100 ACE sentences per second.

Figure 1 shows a screenshot of the APE web client translating a simple text into a DRS.

3.2 ACE Reasoner RACE

The Attempto Reasoner RACE proves that one ACE text — the theorems — is the logical consequence of another one — the axioms — and gives a justification for the proof in ACE. Variations of the basic deduction mode permit query answering and consistency checking.

RACE is implemented in Prolog. The implementation is based on the model generator Satchmo [18] that tries to find a minimal model of a set of range-restricted clauses. Satchmo had to be extended in various ways to fulfil the requirements of RACE.

Since RACE's input is given in ACE, axioms and theorems are translated by APE into discourse representation structures that are then further translated into Satchmo clauses. To generate output in ACE, RACE keeps track of which axioms are needed to prove the theorems, and reports these axioms as result of the proof.

Satchmo stops once it found that a set of clauses is unsatisfiable. RACE, however, finds all proofs, i.e. all possibilities in which subsets of a set of ACE axioms combined with the negation of a set of ACE theorems is unsatisfiable.

Some proofs require domain-independent linguistic and mathematical knowledge that cannot be expressed in ACE, for instance the relation between plurals and singulars or the ordering of natural numbers. To express this knowledge RACE uses auxiliary first-order and Prolog axioms.

Users can access RACE via its web service[11] or via a web client[12] that provides a graphical front-end to the web service.

RACE offers three deduction modes: consistency checking, proving and query answering. All three modes can be controlled by parameters that allow various deductions from collective plurals, enable RACE to check for additional proofs once the first proof was found, and allow users to display auxiliary axioms used in a deduction.

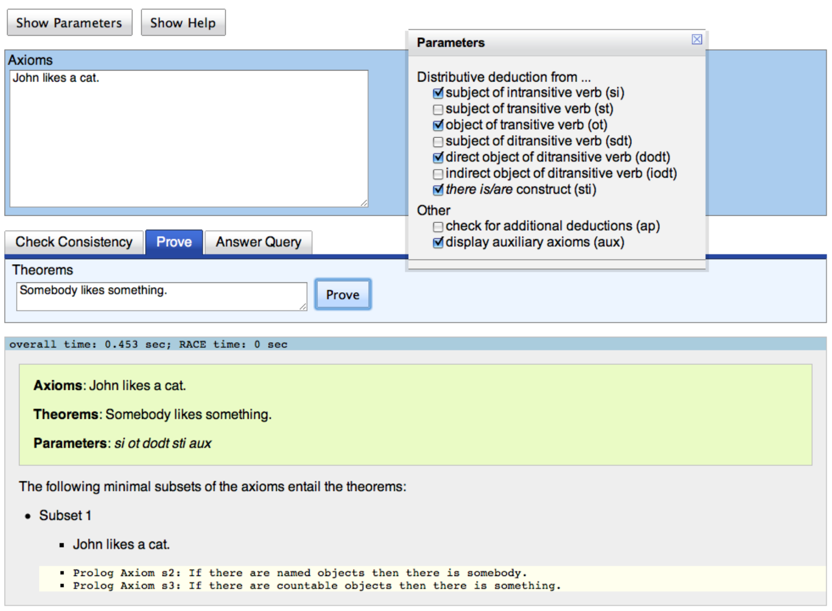
Figure 2 shows a screenshot of the RACE web client proving an ACE theorem from ACE axioms using auxiliary axioms.

RACE applies several means to perform deductions efficiently. Basically, RACE executes its clauses by forward chaining. The worst-case complexity of forward chaining is $O(N^2)$ where N is the number of clauses. Thus an important goal is to reduce the number of clauses that participate in forward chaining. This number was already enormously reduced by simplifications that we introduced in the DRS language. The number of clauses is further reduced by clause compaction, i.e. by a more complex clause form that combines several clauses into one clause. Furthermore, RACE executes rules derived from facts — i.e. rules with the body "true" — only once. Another speed-up is achieved by intelligently selecting the rules that participate in any forward chaining step. Further efficiency is gained by complement splitting of disjunctions — an effective way to prune the search space. Finally, even more efficiency is achieved by expressing auxiliary axioms in Prolog instead of in first-order logic.

3.3 ACE View Protégé Plug-in

ACE View is a novel ontology and rule editor. The goal of ACE View is to simplify the exploration and editing of expressive and syntactically complex OWL 2 [19] ontologies and SWRL [13] rulesets by basing the user interface on ACE. This makes ACE View radically different from current OWL/SWRL editors which are based on standard graphical user interface widgets and formal logic syntaxes, and which are often seen as too complicated and misleading for domain experts

[11] `http://attempto.ifi.uzh.ch/site/docs/race_webservice.html`
[12] `http://attempto.ifi.uzh.ch/race/`



Axioms

John likes a cat.

Parameters

Distributive deduction from ...
✓ subject of intransitive verb (si)
☐ subject of transitive verb (st)
✓ object of transitive verb (ot)
☐ subject of ditransitive verb (sdt)
✓ direct object of ditransitive verb (dodt)
☐ indirect object of ditransitive verb (iodt)
✓ *there is/are* construct (sti)

Other
☐ check for additional deductions (ap)
✓ display auxiliary axioms (aux)

Theorems

Somebody likes something.

```
overall time: 0.453 sec; RACE time: 0 sec
```

Axioms: John likes a cat.

Theorems: Somebody likes something.

Parameters: *si ot dodt sti aux*

The following minimal subsets of the axioms entail the theorems:

- Subset 1
  - John likes a cat.

```
▪ Prolog Axiom s2: If there are named objects then there is somebody.
▪ Prolog Axiom s3: If there are countable objects then there is something.
```

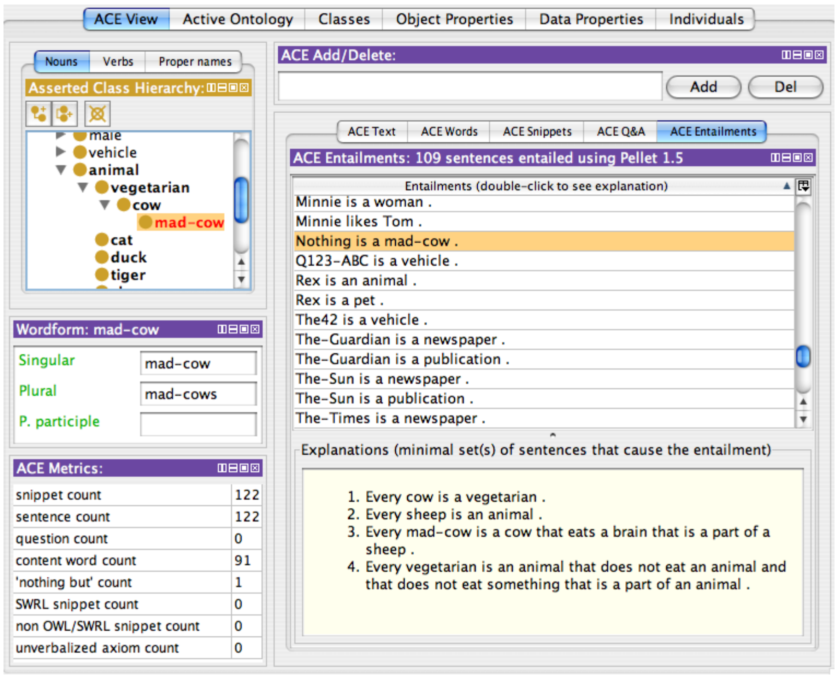
Fig. 2. Screenshot of the RACE web client proving an ACE theorem from ACE axioms using auxiliary axioms

with no background in formal methods. ACE View integrates two mappings, ACE→OWL/SWRL and OWL→ACE (both discussed in detail in [14]), and is implemented as a plug-in for the widely-used Protégé OWL editor [12].

Current OWL editors/viewers (e.g. Protégé[13], OwlSight[14], TopBraid Composer[15]) attempt to hide the complexity resulting from the various ontology, rule, and query formalisms by offering a graphical front-end based on forms, trees, wizards, etc. to enable the entering and viewing of ontologies. This is effective for simple structures like class and property assertions, and the subclass hierarchy between named classes. However, for complex structures like negation, property restrictions, general class inclusion axioms, and SWRL rules, visual methods fail and the tools fall back to one of the native syntaxes of OWL (e.g. Manchester OWL Syntax is used to present OWL class expressions). Thus, in general, current tools fail to hide the complexities of OWL. [4] compared TopBraid Composer and Protégé and found several problems that both novices and experts encountered.

An alternative and less explored approach is to base ontology editing on the use of controlled natural language (CNL) [3, 10, 14]. Several studies have shown that controlled English can offer an improvement in usability over existing approaches for domain experts viewing and editing OWL statements [6, 16, 25]. However, ontology

[13] `http://protege.stanford.edu`
[14] `http://pellet.owldl.com/owlsight/`
[15] `http://www.topbraidcomposer.com`



Wordform: mad-cow

Singular	mad-cow
Plural	mad-cows
P. participle	

ACE Metrics:

snippet count	122
sentence count	122
question count	0
content word count	91
'nothing but' count	1
SWRL snippet count	0
non OWL/SWRL snippet count	0
unverbalized axiom count	0

ACE Entailments: 109 sentences entailed using Pellet 1.5

Entailments (double-click to see explanation)
Minnie is a woman .
Minnie likes Tom .
Nothing is a mad-cow .
Q123-ABC is a vehicle .
Rex is an animal .
Rex is a pet .
The42 is a vehicle .
The-Guardian is a newspaper .
The-Guardian is a publication .
The-Sun is a newspaper .
The-Sun is a publication .
The-Times is a newspaper .

Explanations (minimal set(s) of sentences that cause the entailment)

1. Every cow is a vegetarian .
2. Every sheep is an animal .
3. Every mad-cow is a cow that eats a brain that is a part of a sheep .
4. Every vegetarian is an animal that does not eat an animal and that does not eat something that is a part of an animal .

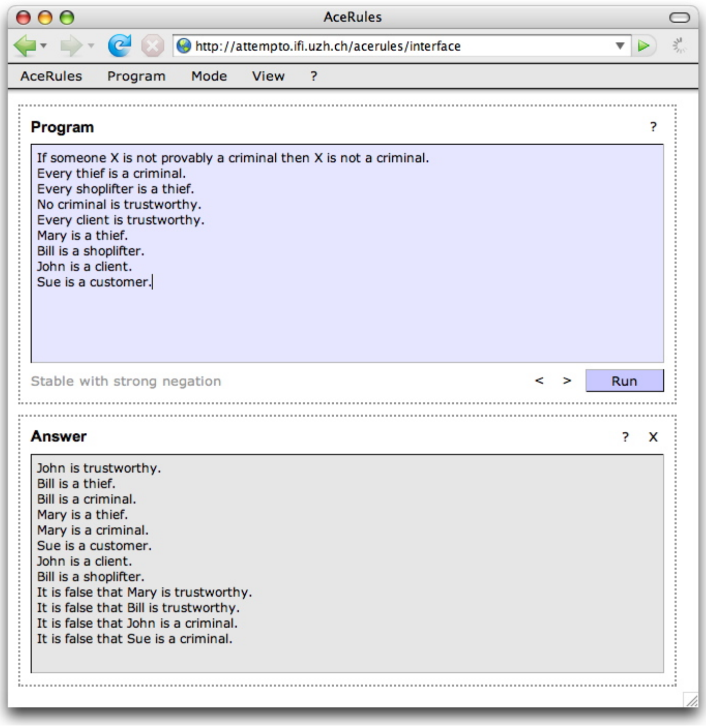
Fig. 3. ACE View offers a controlled English front-end to ontology and rule editing and viewing. The "entailments view" (shown on the screenshot) lists sentences that follow from the entered ACE text. Each of such entailments can be explained by listing a smaller fragment of the text that causes the entailment. The screenshot shows the explanation of why "Nothing is a mad-cow."

editors offering CNL editing as their main component are still in their infancy and their possible architecture has not been agreed upon.

ACE View offers one unified syntax for OWL axioms, SWRL rules and DL queries — axioms and rules are hidden behind English declarative sentences and queries behind English interrogative sentences. ACE View is implemented as an extension to the popular ontology editor Protégé. This greatly simplifies the implementation of our approach as we can leverage the integrated OWL API [11], reasoners, rule and query support that Protégé provides and just concentrate on providing the CNL front-end to these tools. Also, we can easily fall back to the Protégé solutions for e.g. annotation editing, etc. that we do not intend to express in ACE. Being based on Protégé also simplifies the evaluation of our approach, e.g. one sensible way to evaluate ACE View is to let people complete an ontology engineering task and observe for how much of it they want to fall back to Protégé.

3.4 AceRules

AceRules [15] is a multi-semantics rule system prototype using ACE as input and output language. AceRules is designed for forward-chaining interpreters that


AceRules
http://attempto.ifi.uzh.ch/acerules/interface
AceRules Program Mode View ?
Program ?
Stable with strong negation
< > Run
Answer ? X

```
If someone X is not provably a criminal then X is not a criminal.
Every thief is a criminal.
Every shoplifter is a thief.
No criminal is trustworthy.
Every client is trustworthy.
Mary is a thief.
Bill is a shoplifter.
John is a client.
Sue is a customer.
```

```
John is trustworthy.
Bill is a thief.
Bill is a criminal.
Mary is a thief.
Mary is a criminal.
Sue is a customer.
John is a client.
Bill is a shoplifter.
It is false that Mary is trustworthy.
It is false that Bill is trustworthy.
It is false that John is a criminal.
It is false that Sue is a criminal.
```

Fig. 4. Screenshot of the AceRules web interface showing an exemplary program (i.e. rule set) and the answer that is inferred from it

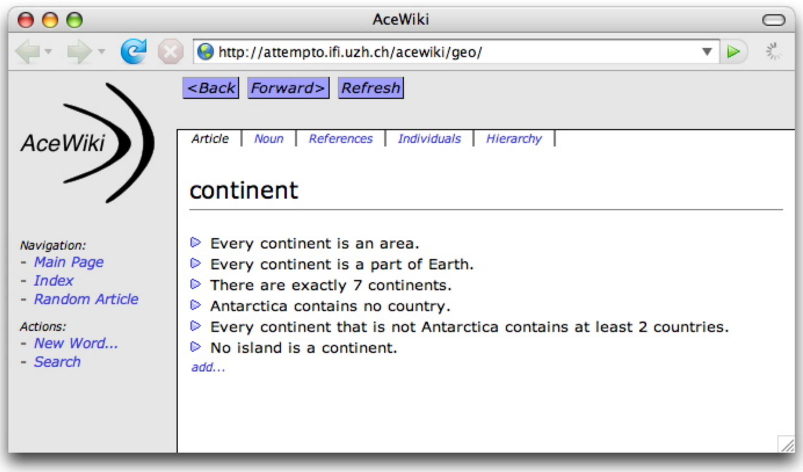
calculate the complete answer set. The general approach of AceRules, however, could easily be adopted for backward-chaining interpreters. Figure 4 shows a screenshot of the AceRules interface. The upper text box is the input component and contains the program to be executed. The result of the program is then displayed in the text box below.

At the moment, AceRules incorporates three different semantics: courteous logic programs [9], stable models [7], and stable models with strong negation [8]. Depending on the semantics, AceRules supports negation as failure, strong negation, labelled rules, and priorities between rules. Furthermore, it has an internal grouping mechanism that transforms certain logical statements that do not have a rule structure into valid rules.

3.5 AceWiki

The goal of AceWiki [17] is to show that semantic wikis can be more natural and at the same time more expressive than existing semantic wikis.

Naturalness is achieved by representing the formal statements in ACE. In order to enable easy creation of ACE sentences, AceWiki provides a predictive editor that shows step-by-step the words and phrases that are syntactically possible at a given position in the sentence. Figure 5 shows a screenshot of the AceWiki interface.



continent

- Every continent is an area.
- Every continent is a part of Earth.
- There are exactly 7 continents.
- Antarctica contains no country.
- Every continent that is not Antarctica contains at least 2 countries.
- No island is a continent.

Fig. 5. Screenshot of the web interface of the AceWiki prototype showing an article about continents

AceWiki makes use of the high expressivity of ACE that goes beyond OWL and SWRL. AceWiki integrates the OWL reasoner Pellet[16] which considers only the sentences that are OWL-compliant. The reasoner is used to ensure that the ontology is always consistent and it calculates the class memberships and class hierarchies which are then displayed in ACE again. Furthermore, the reasoner is used to answer questions formulated in ACE.

We conducted a user experiment [16] that proved that ordinary people with no background in logic are able to deal with AceWiki. The participants — without being instructed how to interact with the interface — were asked to add knowledge to AceWiki. About 80% of the created sentences were correct and sensible. This is remarkable since most of the sentences were quite complex: more than 60% of them contained an implication or a negation or both. Using the predictive editor which the participants had never seen before, they needed on average only five minutes to create their first correct sentence.

3.6 OWL Verbaliser

The OWL verbaliser performs a mapping of the logical content of an OWL ontology into ACE. The OWL verbaliser accepts an OWL ontology in OWL 2 XML serialization as input and produces a plain ACE text as output. The OWL verbaliser is implemented in SWI-Prolog.

The verbalisation of an OWL axiom is done in three steps, two of which perform a set of semantics preserving transformations on the axiom and the third maps the resulting structure directly to ACE. First, the axiom is rewritten into a different form which is either one of *SubClassOf*, *SubPropertyOf*, *DisjointProperties*. This step removes a lot of syntactic sugar (such as axioms dedicated to

[16] `http://pellet.owldl.com`

expressing domains and ranges). Secondly, the structure of *SubClassOf*-axioms is slightly changed, e.g. elements in coordination (*IntersectionOf*, *UnionOf*) are reordered to bring structurally simpler elements to the front. Also, *ComplementOf* is removed from a class description in case it directly embeds (or is directly embedded in) a property restriction. Third, the modified axiom is directly mapped to ACE via a simple Definite Clause Grammar. The purpose of the first two steps is to improve the readability of the eventual ACE representation. The following example demonstrates how an axiom in OWL is via a sequence of transformations translated into a sentence in ACE.

1. ClassAssertion(Sarkozy AllValuesFrom(is-president-of ComplementOf(OneOf(USA))))
2. SubClassOf(OneOf(Sarkozy) AllValuesFrom(is-president-of ComplementOf(OneOf(USA))))
3. SubClassOf(OneOf(Sarkozy) ComplementOf(SomeValuesFrom(is-president-of OneOf(USA))))
4. It is false that Sarkozy is-president-of USA.

4 Conclusions

Attempto Controlled English (ACE) is a language with a dual face — because of its natural language heritage humans can read it easily and because of its logical foundations machines can process it in various ways. The attributes of ACE — specifically its ability to express relations, rules, commands and queries in one and the same language — make it a prime candidate for knowledge representation in almost any application domain including the semantic web. The flexibility and power of ACE becomes apparent when used in tools like APE, RACE, ACE View, AceRules, AceWiki, and OWL verbaliser.

References

1. Clark, P., Harrison, P., Jenkins, T., Thompson, J., Wojcik, R.H.: Acquiring and Using World Knowledge Using a Restricted Subset of English. In: FLAIRS 2005, pp. 506–511 (2005)
2. Clark, P., Porter, B.: KM — The Knowledge Machine 2.0: Users Manual. Technical report (2004), `http://www.cs.utexas.edu/users/mfkb/km/userman.pdf`
3. Cregan, A., Schwitter, R., Meyer, T.: Sydney OWL Syntax — towards a Controlled Natural Language Syntax for OWL 1.1. In: Golbreich, C., Kalyanpur, A., Parsia, B. (eds.) CEUR Proceedings of 3rd OWL Experiences and Directions Workshop (OWLED 2007), vol. 258 (2007)
4. Dzbor, M., Motta, E., Buil, C., Gomez, J., Görlitz, O., Lewen, H.: Developing ontologies in OWL: An observational study. In: 2nd OWL Experiences and Directions Workshop (OWLED 2006) (2006)
5. Fitting, M.: First-Order Logic and Automated Theorem Proving. Springer, Heidelberg (1996)

6. Funk, A., Tablan, V., Bontcheva, K., Cunningham, H., Davis, B., Handschuh, S.: CLOnE: Controlled Language for Ontology Editing. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, Springer, Heidelberg (2007)
7. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Proceedings of the 5th International Conference on Logic Programming, pp. 1070–1080. MIT Press, Cambridge (1988)
8. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing 9, 365–385 (1990)
9. Grosof, B.N.: Prioritized conflict handling for rules. Technical Report RC 20836, IBM Research, IBM T.J. Watson Research Center (December 1997)
10. Hart, G., Dolbear, C., Goodwin, J.: Lege Feliciter: Using Structured English to represent a Topographic Hydrology Ontology. In: Golbreich, C., Kalyanpur, A., Parsia, B. (eds.) CEUR Proceedings of 3rd OWL Experiences and Directions Workshop (OWLED 2007), vol. 258 (2007)
11. Horridge, M., Bechhofer, S., Noppens, O.: Igniting the OWL 1.1 Touch Paper: The OWL API. In: Golbreich, C., Kalyanpur, A., Parsia, B. (eds.) CEUR Proceedings of 3rd OWL Experiences and Directions Workshop (OWLED 2007), vol. 258 (2007)
12. Horridge, M., Jupp, S., Moulton, G., Rector, A., Stevens, R., Wroe, C.: A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools. The University Of Manchester (2007), `http://www.co-ode.org/resources/tutorials/`
13. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission May 21, 2004. Technical report, W3C (2004), `http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/`
14. Kaljurand, K.: Attempto Controlled English as a Semantic Web Language. PhD thesis, Faculty of Mathematics and Computer Science, University of Tartu (2007)
15. Kuhn, T.: AceRules: Executing Rules in Controlled Natural Language. In: Marchiori, M., Pan, J.Z., d Marie, C.S. (eds.) RR 2007. LNCS, vol. 4524, pp. 299–308. Springer, Heidelberg (2007)
16. Kuhn, T.: AceWiki: A Natural and Expressive Semantic Wiki. In: Semantic Web User Interaction at CHI 2008: Exploring HCI Challenges (2008)
17. Kuhn, T.: AceWiki: Collaborative Ontology Management in Controlled Natural Language. In: SemWiki 2008 —The Wiki Way of Semantics (2008)
18. Manthey, R., Bry, F.: SATCHMO: A Theorem Prover Implemented in Prolog. In: Lusk, E.L., Overbeek, R.A. (eds.) CADE 1988. LNCS, vol. 310, pp. 415–434. Springer, Heidelberg (1988)
19. Motik, B., Patel-Schneider, P.F., Horrocks, I.: OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax. Technical report, W3C (2008), `http://www.w3.org/TR/2008/WD-owl2-syntax-20080411/`
20. Pool, J.: Can Controlled Languages Scale to the Web? In: 5th International Workshop on Controlled Language Applications (2006)
21. Pratt-Hartmann, I.: A two-variable fragment of English. Journal of Logic, Language and Information 12(1), 13–45 (2003)
22. Pratt-Hartmann, I., Third, A.: More fragments of language: the case of ditransitive verbs. Notre Dame Journal of Formal Logic 47(2), 151–177 (2006)
23. Schwitter, R.: A Controlled Natural Language Layer for the Semantic Web. In: Zhang, S., Jarvis, R. (eds.) AI 2005. LNCS (LNAI), vol. 3809, pp. 425–434. Springer, Heidelberg (2005)

24. Schwitter, R.: Controlled Natural Language as Interface Language to the Semantic Web. In: 2nd Indian International Conference on Artificial Intelligence (IICAI 2005), Pune, India, December 20–22 (2005)
25. Schwitter, R., Kaljurand, K., Cregan, A., Dolbear, C., Hart, G.: A Comparison of three Controlled Natural Languages for OWL 1.1. In: 4th OWL Experiences and Directions Workshop (OWLED, DC), Washington, April 1–2, 2008, p. 10 (2008)
26. Schwitter, R., Ljungberg, A., Hood, D.: ECOLE — A Look-ahead Editor for a Controlled Language. In: Controlled Translation, Proceedings of EAMT-CLAW 2003, Joint Conference combining the 8th International Workshop of the European Association for Machine Translation and the 4th Controlled Language Application Workshop, Dublin City University, Ireland, May 15–17, 2003, pp. 141–150 (2003)
27. Schwitter, R., Tilbrook, M.: Controlled Natural Language meets the Semantic Web. In: Wan, S., Asudeh, A., Paris, C. (eds.) Australasian Language Technology Workshop 2004, Macquarie University, December 2004, pp. 55–62 (2004)
28. Schwitter, R., Tilbrook, M.: Let's Talk in Description Logic via Controlled Natural Language. In: Logic and Engineering of Natural Language Semantics 2006 (LENLS 2006), Tokyo, Japan, June 5–6 (2006)
29. Sowa, J.F.: Common Logic Controlled English. Technical report, 2004. Draft, 24 (February 2004), `http://www.jfsowa.com/clce/specs.htm`
30. Sowa, J.F.: Common Logic Controlled English. Technical report, 2007. Draft (March 15, 2007), `http://www.jfsowa.com/clce/clce07.htm`