

Chapter 6

Controlled English for Reasoning on the Semantic Web

Juri Luca De Coi¹, Norbert E. Fuchs², Kaarel Kaljurand², and Tobias Kuhn²

¹ L3S, University of Hanover

`decoi@l3s.de`

`http://www.l3s.de`

² Department of Informatics and Institute of Computational Linguistics,
University of Zurich

`{fuchs,kalju,tkuhn}@ifi.uzh.ch`

`http://attempo.ifi.uzh.ch`

Abstract. The existing Semantic Web languages have a very technical focus and fail to provide good usability for users with no background in formal methods. We argue that controlled natural languages like Attempto Controlled English (ACE) can solve this problem. ACE is a subset of English that can be translated into various logic based languages, among them the Semantic Web standards OWL and SWRL. ACE is accompanied by a set of tools, namely the parser APE, the Attempto Reasoner RACE, the ACE View ontology and rule editor, the semantic wiki AceWiki, and the Protune policy framework. The applications cover a wide range of Semantic Web scenarios, which shows how broadly ACE can be applied. We conclude that controlled natural languages can make the Semantic Web better understandable and more usable.

6.1 Why Use Controlled Natural Languages for the Semantic Web?

The Semantic Web proves to be quite challenging for its developers: there is the problem of adequately representing domain knowledge, there is the question of the interoperability of heterogeneous knowledge bases, there is the need for reliable and efficient reasoning, and last but not least the Semantic Web requires generally acceptable user interfaces.

Languages like RDF, OWL, SWRL, RuleML, R2ML, SPARQL etc. have been developed to meet the challenges of the Semantic Web. The developers of these languages are predominantly researchers with a strong background in logic. This is reflected in the languages, all of which have syntaxes that conspicuously show their logic descent. Domain experts and end-users, however, often do not have a background in logic. They shy away from logic notations, and prefer to use notations familiar to them — which is usually natural language possibly complemented by diagrams, tables, and formulas.

The developers of Semantic Web languages have tried to overcome the usability problem by suggesting alternative syntaxes, specifically for OWL. However,

even the Manchester OWL Syntax [21], which is advertised by its authors as “easy to read and write”, lacks the features that would bring OWL closer to domain experts. The authors of [33,23] list the problems that users encounter when working with OWL, and as a result of their investigations express the need for a “pedantic but explicit” paraphrase language. Add to this that many knowledge bases require a rule component, often expressed in SWRL. The proposed SWRL syntax, however, is completely different from any of the OWL syntaxes. Query languages for OWL ontologies introduce yet other syntaxes.

The syntactic complexity of Semantic Web languages can be hidden to some extent by front-end tools such as Protégé¹ that provides various graphical means to view and edit knowledge bases. While the subclass hierarchy of named classes can be concisely presented graphically, for more complex expressions users still have to rely on one of the standard syntaxes.

Thus the languages developed for the Semantic Web do not seem to meet all of its challenges. Though by and large they fulfill the requirements of knowledge representation and reasoning, they seem to fail the requirement of providing general and generally acceptable user interfaces.

Concerning user interfaces, natural language excels as the fundamental means of human communication. Natural language is easy to use and to understand by everybody, and — other than formal languages — does not need an extra learning effort. Though for particular domains there are more concise notations, like diagrams and formulas, natural language can be and is used to express any problem: only listen to scientists paraphrasing complex formulas, or to somebody explaining the way to the station. For this reason, we will in the following focus only on natural language, and not discuss complementary notations. Since natural language is highly expressive, and is used in any application domain, some researchers even consider natural language “the ultimate knowledge representation language” [37]. This claim should be taken with reservations since we must not forget that natural language is highly ambiguous and can be very vague.

Thus there seems to be a conflict: on the one side the Semantic Web needs logic-based languages for adequate knowledge representation and reasoning, and on the other side the Semantic Web requires natural language for generally acceptable user interfaces.

This conflict was already encountered before the advent of the Semantic Web, for instance in the fields of requirements engineering and software specification. Their researchers proposed to use controlled natural languages² to solve the conflict — where a controlled natural language is a subset of the respective natural language specifically designed to be translated into first-order logic. This translatability turns controlled natural languages into logic languages and enables them to serve as knowledge representation and reasoning languages, while preserving readability. As existing controlled natural languages show, the ambiguity and vagueness of full natural language can be avoided.

¹ <http://protege.stanford.edu/>

² <http://www.ics.mq.edu.au/~rolfs/controlled-natural-languages/>

Therefore it is only natural that researchers have proposed to use controlled natural language also for the Semantic Web [35]. In fact, several studies have shown that controlled natural languages offer domain experts improved usability over working with OWL [27,14,18].

Controlled natural languages, for instance Attempto Controlled English that we present in the following, can be translated into various Semantic Web languages, thus providing the features of these languages in one and the same user-friendly syntax. In our view, this demonstrates that ACE and similar controlled natural languages have the potential to optimally meet the challenges of the Semantic Web.

This chapter is structured as follows. Section 2 gives an overview of controlled natural languages. In section 3 we present Attempto Controlled English (ACE), and describe how ACE texts can be translated into first-order logic. Section 4 shows how ACE fits into the Semantic Web, concretely how ACE can be translated into OWL and SWRL, how ACE can be used to express rules and policies, and briefly how ACE can be translated into the languages RuleML, R2ML and PQL. Section 5 is dedicated to tools developed for the ACE language, namely the Attempto Reasoner RACE, the ACE View ontology and rule editor, the semantic wiki AceWiki, and the front-end for the Protune policy language. In section 6 we summarize our experiences, and assess the impact of controlled natural languages on the Semantic Web.

6.2 Controlled Natural Languages: State of the Art

Besides Attempto Controlled English (ACE) that we will describe in detail in the next section, there are several other modern controlled natural languages:

PENG [36] is a language that is similar to ACE but follows a more light-weight approach in the sense that it covers a smaller subset of natural English. Its incremental parsing approach makes it possible to parse partial sentences and to look-ahead to find out how the sentence can be continued.

Common Logic Controlled English (CLCE) [38] is another ACE-like language that has been designed as a human interface language for the ISO standard Common Logic³.

Computer Processable Language (CPL) [7] is a controlled English developed at Boeing. Instead of applying a small set of strict interpretation rules, the CPL interpreter resolves various types of ambiguities in a “smart” way that should lead to acceptable results in most cases.

E2V [32] is a fragment of English that corresponds to a decidable two-variable fragment of first-order logic. In contrast to the other languages, E2V has been developed to study the computational properties of certain linguistic structures and not to create a real-world knowledge representation language.

While the languages presented above have no particular focus on the Semantic Web, there are several controlled natural languages that are designed specifically for OWL:

³ <http://cl.tamu.edu/>

Sydney OWL Syntax (SOS) [10] builds upon PENG and provides a syntactically bidirectional mapping to OWL. The syntactic sugar of OWL is carried over one-to-one to SOS. Thus, semantically equivalent OWL statement that use different syntactical constructs are always mapped to different SOS statements.

Rabbit [18] is a controlled English developed and used by Ordnance Survey (Great Britain's national mapping agency). Rabbit is designed for a scenario where a domain expert and an ontology engineer work together to produce ontologies. Using Rabbit is supported by the ROO (Rabbit to OWL Ontology construction) editor [11]. ROO allows entering Rabbit sentences, helps to resolve possible syntax errors, and translates them into OWL.

Lite Natural Language [2] is a controlled natural language that maps to DL-Lite which is one of the tractable fragments of OWL. Lite Natural Language can be seen as a subset ACE.

CLOnE [13] is a very simple language defined by only eleven sentence patterns which roughly correspond to eleven OWL axiom patterns. For that reason, only a very small subset of OWL is covered.

ACE is unique in the sense that it covers both aspects: It is designed as a general-purpose controlled English providing a high degree of expressivity. At the same time, ACE is fully interoperable with the Semantic Web standards, since a defined subset of ACE can bidirectionally be mapped to OWL.

6.3 Attempto Controlled English (ACE)

6.3.1 Overview of Attempto Controlled English

This section contains a brief survey of the syntax of the language Attempto Controlled English (ACE). Furthermore, we summarize ACE's handling of ambiguity, and show how sentences can be interrelated by anaphoric references.

Syntax of ACE. The vocabulary of ACE comprises predefined function words (e.g. determiners, conjunctions), predefined fixed phrases (e.g. 'it is false that', 'for all'), and content words (nouns, proper names, verbs, adjectives, adverbs).

The grammar of ACE — expressed as a set of construction rules and a set of interpretation rules — defines and constrains the form and the meaning of ACE sentences and texts.

An ACE text is a sequence of declarative sentences that can be anaphorically interrelated. Furthermore, ACE supports questions and commands. Declarative sentences can be simple or composite.

Simple ACE sentences can have the following structure:

subject + verb + complements + adjuncts

A customer inserts two cards manually in the morning.

Every sentence of this structure 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'), whereas adjuncts (adverbs, prepositional phrases) that modify the verb are optional.

Alternatively, simple sentences can be built according to the structure:

'there is'/'there are' + noun phrase

There is a customer.

Every sentence of this structure introduces only the object described by the noun phrase.

Elements of a simple sentence can be elaborated upon to describe the situation in more detail. To further specify the nouns, we can add adjectives, possessive nouns and *of*-prepositional phrases, or variables as appositions.

A bank's trusted customer X inserts two valid cards of himself.

Other modifications of nouns are possible through relative clauses

A customer who is trusted inserts two cards that he owns.

Composite sentences are recursively built from simpler sentences through coordination, subordination, quantification, and negation.

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

A customer inserts a card and an automated teller checks the code.

A customer inserts a card and enters a code.

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

A customer inserts a card or an automated teller checks the code.

A customer inserts a card or enters a code.

A customer owns a card that is invalid or that is damaged.

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.

There are three constructs of subordination: *if-then*-sentences, modality, and sentence subordination. With the help of *if-then*-sentences we can specify conditional situations, e.g.

If a card is valid then a customer inserts it.

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 means that a complete sentence is used as an object, e.g.

It is false that a customer inserts a card.

A clerk believes that a customer inserts a card.

Sentences can be existentially or universally quantified. Existential quantification is typically expressed by indefinite determiners ('a man', 'some water', '3 cards'), while universal quantification is typically expressed by the occurrence of 'every' — but see below for the quantification within *if-then*-sentences. In the example

Every customer inserts a card.

the noun phrase 'every customer' is universally quantified, while the noun phrase 'a card' is existentially quantified, i.e. every customer inserts a card that may, or may not, be the same card that another customer inserts. Note that this sentence is logically equivalent to the sentence

If there is a customer then the customer inserts a card.

which shows that noun phrases occurring in the *if*-part of an *if-then*-sentence are universally quantified.

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

A customer does not insert a card.

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

No customer inserts more than 2 cards.

To negate a complete statement one uses sentence negation.

It is false that a customer inserts a card.

ACE supports two forms of queries: *yes/no*-queries and *wh*-queries. *Yes/no*-queries ask for the existence or non-existence of a specified situation.

Does a customer insert a card?

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 trusted customer inserts a valid card manually.

we can ask for each element of the sentence with the exception of the verb, e.g.

Who inserts a card?

Which customer inserts a card?

What does a customer insert?

How does a customer insert a card?

Finally, ACE also supports commands. Some examples:

John, go to the bank!

John and Mary, wait!

Every dog, bark!

A brother of John, give a book to Mary!

Constraining Ambiguity. To constrain the ambiguity of full English 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, or they must rephrase the input to obtain another one

Here is an example how ACE replaces ambiguous constructs by unambiguous constructs. In full English relative clauses 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. To express the alternative meaning that the card opens an account the relative pronoun ‘that’ must be repeated, thus yielding a coordination of relative clauses.

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

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.

Here is an example of an interpretation rule at work. In

A customer inserts a card with a code.

‘with a code’ attaches to the verb ‘inserts’, but not to ‘a card’. To express that the code is associated with the card we can employ the complementary 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.

Anaphoric References. Usually an ACE text consists of more than one sentence, e.g.

A customer enters a card and a code. If a code is valid then an automated teller accepts 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 an automated teller accepts 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.

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, which is the textually closest noun phrase that matches the anaphor ‘the card’, while

The red card is correct.

refers to the first card that is the textually closest noun phrase that matches the anaphor ‘the red card’.

What does “accessible” mean? Like in full English, noun phrases introduced in *if-then*-sentences, universally quantified sentences, negations, modality, and subordinated sentences cannot be referenced anaphorically in subsequent sentences. Thus for each of the sentences

If a customer owns a card then he enters it.

A customer does not enter a card.

we cannot refer to ‘a card’ with

The card is correct.

Anaphoric references are also possible via personal pronouns

A customer enters his own card and its code. If it is valid then an automated teller accepts the card.

or via variables

A customer X enters X’s card Y and Y’s code Z. If Z is valid then an automated teller accepts Y.

Note that proper names always denote the same object.

6.3.2 From Attempto Controlled English to First-Order Logic

ACE texts can be mapped to Discourse Representation Structures (DRS) [24,5]. DRSs use a syntactic variant of the language of standard first-order logic which we extended by some non-standard structures for modality, sentence subordination, and negation as failure. This section gives a brief overview of the DRS representation of ACE texts. Consult [12] for a comprehensive description. DRSs consist of a domain and of a list of conditions, and are usually displayed in a box notation:

Domain
Condition1
...
ConditionN

The domain is a set of discourse referents (i.e. logical variables) and the conditions are a set of first-order logic predicates or nested DRSs. The discourse referents are existentially quantified with the exception of boxes on the left-hand side of an implication where they are universally quantified. We are using a reified (or “flat”) notation for the predicates. For example, the noun ‘a card’ that normally would be represented in first-order logic as

`card(A)`

is represented as

`object(A,card,countable,na,eq,1)`

relegating the predicate ‘card’ to the constant ‘card’ used as an argument in the predefined predicate ‘object’. In that way, we can reduce the potentially large number of predicates to a small number of predefined predicates. This makes the processing of the DRS easier and allows us to include some linguistic information, e.g. whether a unary relation comes from a noun, from an adjective, or from an intransitive verb. Furthermore, reification allows us to quantify over predicates and thus to express general axioms needed for reasoning over ACE text in the Attempto Reasoner RACE that is presented in Section 6.5.1.

Proper names, countable nouns, and mass nouns are represented by the `object`-predicate:

John drives **a car** and buys **2 kg of rice**.

A	B	C	D	E
object(A,'John',named,na,eq,1)				
object(B,car,countable,na,eq,1)				
predicate(C,drive,A,B)				
object(D,rice,mass,kg,eq,2)				
predicate(E,buy,A,D)				

Adjectives introduce `property`-predicates:

A young man is **richer than** Bill.

A	B	C	D
object(A,'Bill',named,na,eq,1)			
object(B,man,countable,na,eq,1)			
property(B,young,pos)			
property(C,rich,comp,than,A)			
predicate(D,be,B,C)			

As shown in the examples above, verbs are represented by `predicate`-predicates. Each verb occurrence gets its own discourse referent which is used to attach modifiers like adverbs (using `modifier_adv`) or prepositional phrases (using `modifier_pp`):

John **carefully** works **in** an office.

A B C
object(A,'John',named,na,eq,1)
object(B,office,countable,na,eq,1)
predicate(C,work,A)
modifier_adv(C,carefully,pos)
modifier_pp(C,in,B)

The **relation**-predicate is used for *of*-constructs, Saxon genitive, and possessive pronouns:

A brother **of** Mary's mother feeds **his own** dog.

A B C D E
object(A,'Mary',named,na,eq,1)
object(B,brother,countable,na,eq,1)
relation(C,of,A)
object(C,mother,countable,na,eq,1)
relation(B,of,C)
relation(D,of,B)
object(D,dog,countable,na,eq,1)
predicate(E,feed,B,D)

There are some more predicates which are not discussed here, but are described in [12]. The examples so far have been simple in the sense that they contained no universally quantified variables and there was no negation, disjunction, or implication. For such more complicated statements, nested DRSs become necessary. In the case of negation, a nested DRS is introduced that is prefixed by a negation sign:

A man **does not** buy a car.

A			
object(A,man,countable,na,eq,1)			
¬			
<table> <tr> <th>B C</th></tr> <tr> <td>object(B,car,countable,na,eq,1)</td></tr> <tr> <td>predicate(C,buy,A,B)</td></tr> </table>	B C	object(B,car,countable,na,eq,1)	predicate(C,buy,A,B)
B C			
object(B,car,countable,na,eq,1)			
predicate(C,buy,A,B)			

Note that 'a man' is not in the scope of the negation. In ACE, scopes are determined on the basis of the textual order of the sentence elements. In the following example, 'a man' is also under negation:

It is false that a man buys a car.

¬				
<table> <tr> <th>A B C</th></tr> <tr> <td>object(A,man,countable,na,eq,1)</td></tr> <tr> <td>object(B,car,countable,na,eq,1)</td></tr> <tr> <td>predicate(C,buy,A,B)</td></tr> </table>	A B C	object(A,man,countable,na,eq,1)	object(B,car,countable,na,eq,1)	predicate(C,buy,A,B)
A B C				
object(A,man,countable,na,eq,1)				
object(B,car,countable,na,eq,1)				
predicate(C,buy,A,B)				

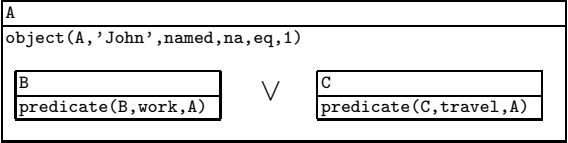
The ACE structures 'every', 'no', and 'if ... then' introduce implications that are denoted by arrows between two nested DRSs.

Every woman owns a house.

<table><tr><th>A</th></tr><tr><td>object(A,woman,countable,na,eq,1)</td></tr></table>	A	object(A,woman,countable,na,eq,1)	⇒	<table><tr><th>B C</th></tr><tr><td>object(B,house,countable,na,eq,1)</td></tr><tr><td>predicate(C,own,A,B)</td></tr></table>	B C	object(B,house,countable,na,eq,1)	predicate(C,own,A,B)
A							
object(A,woman,countable,na,eq,1)							
B C							
object(B,house,countable,na,eq,1)							
predicate(C,own,A,B)							

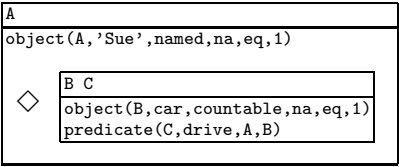
As stated before already, discourse referents that are introduced in a DRS box that is on the left-hand side of an implication are universally quantified. In all other cases, they are existentially quantified. Disjunctions — which are represented in ACE by the coordination ‘or’ — are represented in the DRS by the logical sign for disjunction:

John works **or** travels.

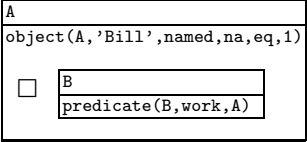


The modal constructs of possibility (‘can’) and necessity (‘must’) are represented by the standard modal operators (see [6] for details):

Sue **can** drive a car.

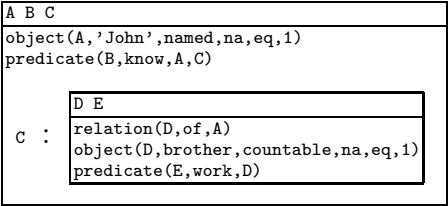


Bill **must** work.



Finally, *that*-subordination can lead to the situation where a discourse referent stands for a whole sub-DRS:

John knows **that** his brother works.



Every ACE sentence can be mapped to exactly one DRS using the introduced DRS elements. DRSs are a convenient and flexible way to represent logical statements.

6.3.3 Attempto Parsing Engine (APE)

The Attempto Parsing Engine (APE) is a tool that translates an input ACE text into a DRS, provides various technical feedback (tokenization and sentence

splitting of the input text, tree-representation of the syntactic structure of the input), and various logical forms and representations derived from the DRS: standard first-order logic form, DRS in XML, OWL/SWRL. An ACE paraphrase of the input text is also offered, by translating (verbalizing) the obtained DRS into a subset of ACE.

If the input text contains syntax errors or unresolvable anaphoric references then the translation into a DRS fails and a message is output that pinpoints the location and the cause of the error. Furthermore, APE tries to suggest how to resolve the problem.

APE implements the ACE syntax in the form of approximately 200 definite clause grammar rules using feature structures. APE comes with a large lexicon containing almost 100'000 English words. User defined lexica can be used in addition or in place of this large lexicon.

APE has been implemented in SWI-Prolog and released under the LGPL open source license. The distribution also includes the DRS verbalizer, translator from ACE to OWL/SWRL, and more⁴. APE has a command-line client and can be also used from Java, or over HTTP as a REST web service⁵ or from its demo client⁶. Figure 1 shows a screenshot of the APE web client.

6.4 Fitting ACE into the Semantic Web

6.4.1 OWL and SWRL

In order to make ACE interoperable with some of the existing Semantic Web languages, mappings have been developed to relate ACE to OWL and SWRL (see a detailed description in [22]). For example, the mapping of ACE to OWL/SWRL translates the ACE text

Every employee that does not own a car owns a bike.
 Every man that owns a car likes the car.
 Which car does John own?

into a combination of OWL axiom, SWRL rule and DL-Query (an OWL class expression).

$$\begin{aligned} employee \sqcap \neg (\exists own\ car) &\sqsubseteq \exists own\ bike \\ man(?x) \wedge own(?x, ?y) \wedge car(?y) &\rightarrow like(?x, ?y) \\ car \sqcap \exists own^- \{John\} \end{aligned}$$

Note that the mapping is performed on the DRS level, meaning that all ACE sentences that share their corresponding DRS are mapped into the same OWL/SWRL form. ACE provides a lot of linguistically motivated syntactic sugar, e.g. the following sentences have the same meaning (because they have the same DRS).

⁴ <http://attempto.ifi.uzh.ch/site/downloads/>

⁵ http://attempto.ifi.uzh.ch/site/docs/ape_webservice.html

⁶ <http://attempto.ifi.uzh.ch/ape/>

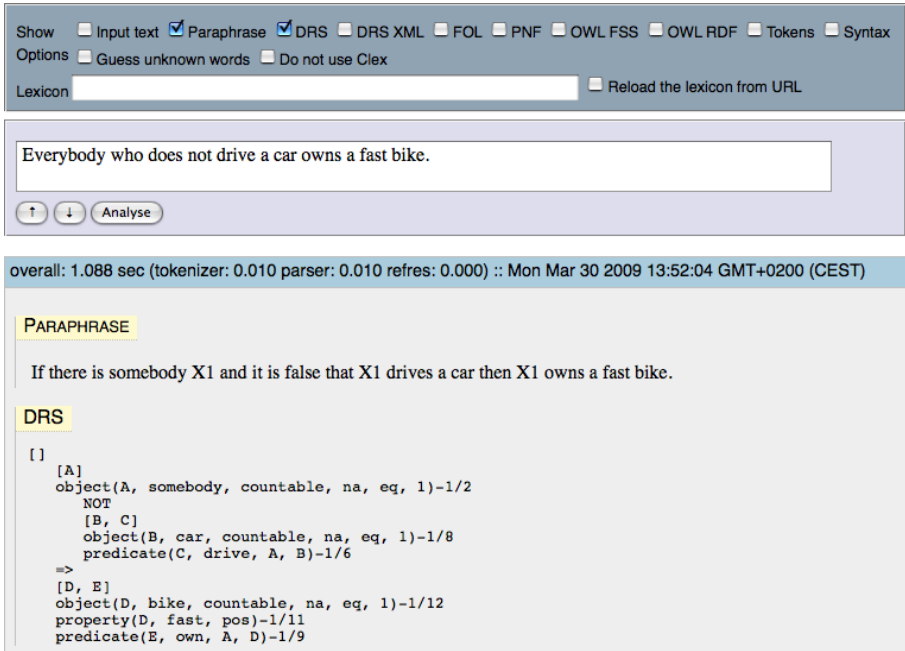


Fig. 1. Screenshot of the APE web client, showing the DRS and the paraphrase of the sentence ‘everybody who does not drive a car owns a fast bike’

John knows every student.
 Every student is known by John.
 If there is a student then John knows the student.
 For every student John knows him/her.

In order to keep the mappings simple and immediately reversible, they currently support only a fragment of ACE. Notably, there is no support for modifiers such as adjectives, adverbs, and prepositional phrases. The covered ACE fragment, however, is so large and syntactically and semantically expressive, that it covers almost all of OWL 2 (some aspects of data properties are not handled) and SWRL (again, data properties are not completely covered). ACE questions that contain exactly one query word (‘what’, ‘which’, ‘whose’, ‘who’) are mapped to DL-Queries.

The OWL→ACE mapping allows us to verbalize existing OWL ontologies as ACE texts. This mapping is not just the reverse of the ACE→OWL as it also covers OWL axiom and expression types that the ACE→OWL mapping does not generate. For example

PropertyDomain(write author)

is verbalized as

Everything that writes something is an author.

Table 1. Examples of verbalizing OWL property and class expressions as ACE verbs and noun phrases (including common nouns and proper names), where R is a named property, $C, C_1, \dots, C_n$ are (possibly complex) class expressions, a is an individual, n is a natural number larger than 0. In the actual verbalizations, the word ‘something’ is often replaced by a noun representing a conjoined named class, e.g. *IntersectionOf(cat ExistsSelf(like))* would be verbalized as ‘cat that likes itself’.

OWL properties and classes	Examples of ACE verbs and noun phrases
<i>Named property</i>	Transitive verb, e.g. ‘like’
<i>InverseProperty(R)</i>	Passive verb, e.g. ‘is liked by’
<i>Named class</i>	Common noun, e.g. ‘man’
owl:Thing	‘something’, ‘thing’
<i>ComplementOf(C)</i>	‘something that is not a car’, ‘something that does not like a cat’
<i>IntersectionOf(C₁...C_n)</i>	something that is a person and that owns a car and that ...
<i>UnionOf(C₁...C_n)</i>	something that is a wild-animal or that is a zoo-animal or that ...
<i>OneOf(a)</i>	Proper name, e.g. ‘John’, ‘something that is John’
<i>SomeValuesFrom(R C)</i>	something that loves a person
<i>ExistsSelf(R)</i>	something that likes itself
<i>MaxCardinality(n R C)</i>	something that has at most 2 spouses

The resulting ACE sentence can be handled by the ACE→OWL mapping by converting it into a general class inclusion axiom with the same semantics as the property domain axiom.

The subset of ACE used in these mappings provides a corresponding ACE content word (proper name, common noun, transitive verb) for each OWL entity, whereas complex OWL class and property expressions map to ACE phrases, and OWL axioms map to ACE sentences. At the entity level, OWL individuals are denoted by ACE proper names, named classes by common nouns, and (object) properties by transitive verbs and relational nouns (e.g. ‘part of’). In OWL, it is possible to build complex class expressions from simpler ones by intersection, union, complementation and property restriction. Similarly, ACE allows building complex noun phrases via relative clauses which can be conjoined (by ‘and that’), disjoined (by ‘or that’), negated (by ‘that is/are/does/do not’) and embedded (by ‘that’). OWL anonymous inverse properties map to ACE passive verbs. This proves that in principle, each OWL structure can be mapped to a corresponding ACE structure.⁷ Table 1 shows some examples of mapping OWL classes and properties.

⁷ Only very complex structures that would require parentheses to denote the scope of their constructors cannot be directly mapped to ACE as ACE does not offer a similar parentheses mechanism for grouping. In order to enable the verbalization in such cases, one can replace part of the complex structure by a named class to simplify the structure.

Table 2. Examples of verbalizing OWL axioms as ACE sentences, where $R_1, \dots, R_n$, and S are object property expressions; C and D are class expressions; and a, a_1 and a_2 are individuals.

OWL axiom types	Examples of ACE sentences
SubClassOf($C D$)	Every man is a human.
SubPropertyOf(PropertyChain($R_1 \dots R_n$) S)	If X knows Y and Y is an editor of Z then X submits-to Z.
DisjointProperties($R_1 R_2$)	If X is a child of Y then it is false that X is a spouse of Y.
SameIndividual($a_1 a_2$)	Bill is William.
DifferentIndividuals($a_1 a_2$)	Bill is not John.
ClassAssertion($C a$)	Bill is a man that owns at least 2 cars.

OWL axioms are mapped to ACE sentences (see table 2 for some examples). Apart from sentences that are derived from the axioms about individuals (*SameIndividual*, *DifferentIndividuals*, *ClassAssertion*, *PropertyAssertion*), all sentences are *every*-sentences or *if-then*-sentences, meaning that they have a pattern ‘*NounPhrase VerbPhrase*’ or ‘*If X ... then X ... Y*’ where *NounPhrase* starts with ‘every’ or ‘no’. Of course, in the ACE to OWL/SWRL direction one can use *if-then*-sentences instead of *every*-sentences and has also otherwise more flexibility.

In a nutshell, the mappings between ACE and OWL/SWRL provide an alternative syntax for OWL and SWRL. This syntax is readable as standard English, it makes the difference between OWL and SWRL invisible, and provides linguistically motivated syntactic sugar. This syntax is mainly intended for structurally and semantically complex knowledge bases for which visual methods and the official OWL/SWRL syntaxes fail to provide a user-friendly front-end.

6.4.2 AceRules: Rules in ACE

AceRules is a multi-semantics rule engine using ACE as input and output language. AceRules has been introduced in [26] and is designed for forward-chaining interpreters that calculate the complete answer set. The following is a simple exemplary program (we use the term “program” for a set of rules and facts):

John is a customer.
 John is a friend of Mary.
 Mary is an important customer.
 Every customer is a person.
 Every customer who is a friend of Bill gets a discount.
 If a person is important then he/she gets a discount.
 Every friend of Mary is a friend of Bill.

Submitting this program to AceRules, we get the following answer (we use the term “answer” for the set of facts that can be derived from a program):

Mary is important.
Mary is a customer.
John is a customer.
Mary is a person.
John is a person.
John is a friend of Mary.
John is a friend of Bill.
Mary gets a discount.
John gets a discount.

The program and the answer are both represented in ACE and no other formal notation is needed for the user interaction.

AceRules is designed to support various semantics. Depending on the application domain, the characteristics of the available information, and on the reasoning tasks to be performed, different rule semantics are needed. At the moment, AceRules incorporates three different semantics: courteous logic programs [17], stable models [15], and stable models with strong negation [16]. Only little integration effort would be necessary to incorporate other semantics into AceRules.

Negation is a complicated issue in rule systems. In many cases, two kinds of negation [39] are required. Strong negation (also called “classical negation” or “true negation”) indicates that something can be proven to be false. Negation as failure (also called “weak negation” or “default negation”), in contrast, states only that the truth of something cannot be proven.

However, there is no such general distinction in natural language. It depends on the context, what kind of negation is meant. This can be seen with the following two examples in natural English:

1. *If there is no train approaching then the school bus can cross the railway tracks.*
2. *If there is no public transport connection to a customer then John takes the company car.*

In the first example (which is taken from [16]), the negation corresponds to strong negation. The school bus is allowed to cross the railway tracks only if the available information (e.g. the sight of the bus driver) leads to the conclusion that no train is approaching. If there is no evidence whether a train is approaching or not (e.g. because of dense fog) then the bus driver is not allowed to cross the railway tracks.

The negation in the second sentence is most probably to be interpreted as negation as failure. If one cannot conclude that there is a public transport connection to the customer on the basis of the available information (e.g. public transport schedules) then John takes the company car, even if there is a special connection that is just not listed.

As long as only one kind of negation is available, there is no problem to express this in controlled natural language. As soon as two kinds of negation are supported, however, we need to distinguish them somehow. We found a

natural way to represent the two kinds of negation in ACE. Strong negation is represented with the common negation constructs of natural English:

- ‘does not’, ‘is not’ (e.g. ‘John is not a customer’)
- ‘no’ (e.g. ‘no customer is a clerk’)
- ‘nothing’, ‘nobody’ (e.g. ‘nobody knows John’)
- ‘it is false that’ (e.g. ‘it is false that John waits’)

To express negation as failure, we use the following constructs:

- ‘does not provably’, ‘is not provably’ (e.g. ‘a customer is not provably trustworthy’)
- ‘it is not provable that’ (e.g. ‘it is not provable that John has a card’)

This allows us to use both kinds of negation side by side in a natural looking way. The following example shows a rule using strong negation and negation as failure at the same time.

If a customer does not have a credit-card and is not provably a criminal then the customer gets a discount.

This representation is compact and we believe that it is well understandable. Even persons who have never heard of strong negation and negation as failure can understand it to some degree.

The original stable model semantics supports only negation as failure, but it has been extended to support also strong negation. Courteous logic programs are based on stable models with strong negation and support both forms of negation.

None of the two forms of stable models guarantee a unique answer set. Thus, some programs can have more than one answer. In contrast, courteous logic programs generate always exactly one answer. In order to achieve this, priorities are introduced and the programs have to be acyclic. The AceRules system demonstrates how these different rule semantics can be expressed in ACE in a natural way.

6.4.3 The Protune Policy Language

The term “policy” can be generally defined as a “statement specifying the behavior of a system”, i.e., a statement which describes which decision the system should take or which actions it should perform according to specific circumstances.

Some of the application areas where policies have been lately used are security and privacy. A security policy defines security restrictions for a system, organization or any other entity. A privacy policy is a declaration made by an organization regarding its use of customers’ personal information (e.g., whether third parties may have access to customer data and how that data will be used).

The ability of expressing policies in a formal way can be regarded as desirable: the authority defining policies would have to express them in a machine-understandable way whereas all further processing of the policies could take place in an automatic fashion.

For this reason a number of policy languages have been defined in the last years (cf. [9] for an extensive comparison among them). Nevertheless a major hindrance to widespread adoption of policy languages are their shortcomings in terms of usability: in order to be machine-understandable all of them rely on a formal syntax, which common users find unintuitive and hard to grasp.

We think that the use of controlled natural languages can dramatically improve usability of policy languages. This section describes how we exploited (a subset of) ACE in order to express policies and how we developed a mapping between ACE policies and the ones written in the Protune policy language. The Protune policy language is extensively described in Chapter 4. This section only provides a general overview of the Protune policy language and especially focuses on its relevant features w.r.t. the $ACE \rightarrow$ Protune mapping.

Protune is a Logic Programming-based policy language and as such a Protune policy has much in common with a Logic Program. For instance the Protune policy

$$\begin{aligned} A &\leftarrow B_{11}, \dots, B_{1n}. \\ \dots \\ A &\leftarrow B_{m1}, \dots, B_{mn}. \end{aligned}$$

can be read as follows: A holds if one of

- $(B_{11} \text{ and } \dots \text{ and } B_{1n})$
- $\dots$
- $(B_{m1} \text{ and } \dots \text{ and } B_{mn})$

holds. In this overview we only introduce two additional features of Protune policies w.r.t. usual logic programs, namely actions and complex terms.

A policy may require that under some circumstances some actions are performed: a typical scenario in an authorization context requires that access to some resource is allowed only if the requester provides a valid credential. For this reason the Protune language allows to define actions like in the following example.

$$allow(action_1) \leftarrow action_2.$$

The rule above can be read as follows: $action_1$ can be executed if $action_2$ has been executed. Notice the different semantics of the actions according to the side of the rule they appear in: in order to stress this semantic difference we force the policy author to write actions appearing in the left side of a rule into the $allow/1$ predicate.

The evaluation of a policy may require to deal with entities which can be modeled as sets of $(attribute, value)$ pairs. This is the case with the credentials mentioned in the example above. The Protune language allows to refer to $(attribute, value)$ pairs of such an entity by means of the following notation.

$$identifier.attribute : value$$

Only a subset of the ACE language needs to be exploited when defining policies: data (i.e., integers, reals and strings), nouns, adjectives (in positive, comparative

and superlative form), (intransitive, transitive and ditransitive) verbs and prepositional phrases (in particular *of*-constructs) can be used with some restrictions, the most remarkable of which is that plural noun phrases are not allowed. This means that neither expressions like ‘some cards’ or ‘at least two cards’ nor sentences like ‘John and Mary are nice’ are supported. However notice that some of such sentences (although not all) can be rewritten as sets of sentences (e.g., the previous example can be split into ‘John is nice’ and ‘Mary is nice’). The complete set of restrictions can be found in [8].

ACE provides a number of complex structures to combine simple sentences into larger ones, whereas only few of them (namely negation as failure, possibility and implication) can be exploited in order to express policies. Moreover whilst ACE complex structures can be arbitrarily nested, in ACE policies nesting is allowed only according to given patterns. Roughly speaking (more on this in [8]) ACE policies must have one of the following formats

- If B_1 and $\dots$ and B_n then H .
- H .

where B_i ($1 \leq i \leq n$) may contain a negation-as-failure or possibility construct and H may contain a possibility construct. For example, only the first one of the following sentences is a correct ACE policy: ‘if it is not provable that John has a forged credit-card then John can access “myFile”’ and ‘it is not provable that John has a forged credit-card’.

The restrictions listed above allow to straightforwardly map ACE sentences into Protune rules: it should be easy to figure out that the ACE implication (resp. negation as failure) construct maps to Protune rules (resp. negated literals). On the other hand the ACE possibility construct is meant to convey the semantics of the *allow/1* Protune predicate. Other remarkable mapping rules are accounted for in the following list.

- A programmer asked to formalize the sentence ‘John gives Mary the book’ as a logic program would most likely come up with a rule like *give(john, mary, book)*. Indeed in many cases translating verbs into predicate names can be considered the most linear approach, and we pursued this approach as well. However the arity of a Protune predicate can be arbitrary, whereas intransitive (resp. transitive, ditransitive) verbs can be naturally modeled as predicates with arity one (resp. two, three). For this reason we decided to exploit ACE prepositional phrases (except *of*-constructs) for providing further parameters to a Protune predicate. For instance, sentence ‘John gets “A” in physics.’ translates into ‘*get#in*’(‘John’, ‘A’, *physics*).
- A statement like ‘John is Mary’s brother’ can be seen as asserting some information about the entity “Mary”, namely that the value of her property “brother” is “John”. It should be then intuitive exploiting Protune complex terms to map such ACE sentence to ‘*Mary.brother* : ‘John’.
- When translating noun phrases like ‘a user’ it must be decided if it really matters whether we are speaking about a “user” (in which case the noun phrase could be translated as *user(X)*) or not (in which case the noun phrase

could be translated as a variable X). According to our experience, policy authors tend to use specific concepts even if they actually mean generic entities. For this reason we followed the second approach, according to which the sentence ‘if a user owns a file then the user can access the file’ is translated into the Protune rule: $allow(access(User, File)) \leftarrow own(User, File)$. If it is needed to point out that the one owning a file is a user, the sentence can be rewritten e.g., as follows: ‘if X is a user and X owns a file then X can access the file’ which gives the translation: $allow(access(X, File)) \leftarrow user(X), own(X, File)$.

6.4.4 Other Web Languages

ACE has also been translated into other Semantic Web languages. A translator has been implemented that converts ACE texts into the Rule Markup Language (RuleML) [19]. Another translator has been developed that translates a subset of ACE into the REVERSE Rule Markup Language (R2ML) [30]. R2ML integrates among others the Semantic Web Rule Language (SWRL) and the Rule Markup Language (RuleML) [40]. Furthermore, ACE has been used as a front-end for the Process Query Language (PQL) that allows users to query MIT’s Process Handbook. It has been shown that queries expressed in ACE and automatically translated into PQL provide a more user-friendly interface to the Process Handbook [4,3].

6.5 ACE Tools for the Semantic Web

6.5.1 Attempto Reasoner RACE

The Attempto Reasoner RACE supports automatic reasoning in the first-order subset of ACE that consists of all of ACE with the exception of negation as failure, modality, and sentence subordination. For simplicity, the first-order subset of ACE is simply called ACE in this section.

RACE proves that theorems expressed in ACE are the logical consequence of axioms expressed in ACE, and gives a justification for the proof in ACE. If there is more than one proof, then RACE will find all of them. If a proof fails, then RACE will indicate which parts of the theorems could not be proved. Variations of the basic proof procedure permit query answering and consistency checking.

The current implementation of RACE is based on the model generator Satchmo [31]. The Prolog source code of Satchmo is available — which allows us to easily add modifications and extensions. The two most important extensions are an exhaustive search for proofs and a tracking mechanism.

- exhaustive search: while Satchmo stops once it finds the first inconsistency, RACE will find all inconsistencies
- tracking mechanism: RACE will report for each successful proof which minimal subset of the axioms is needed to prove the theorems.

Currently, we employ RACE only for theorem proving. To better answer *wh*-questions we plan to utilize RACE also as model generator.

RACE works with the clausal form of first-order logic. ACE axioms A and ACE theorems T are translated — via DRSs generated by APE — into their first-order representations FA , respectively FT . Then the conjunction $(FA \wedge \neg FT)$ is translated into clauses, submitted to RACE and checked for consistency. RACE will find all minimal inconsistent subsets of the clauses and present these subsets using the original ACE axioms A and theorems T . If there is no inconsistency, RACE will generate a minimal finite model — if there is one.

RACE is supported by auxiliary axioms expressed in Prolog. Auxiliary axioms implement domain-independent linguistic knowledge that cannot be expressed in ACE since this knowledge depends on the DRS representations of ACE texts. A typical example is the relation between the plural and the singular forms of nouns. Auxiliary axioms can also act as meaning postulates for ACE constructs that are under-represented in the DRS, for example generalized quantifiers and indefinite pronouns. Finally, auxiliary axioms could also be used instead of ACE to represent domain-specific knowledge.

ACE is undecidable. Technically this means that RACE occasionally would not terminate. To prevent this situation, RACE uses a time-out with a time limit that is calculated on the size of the input.

In the spirit of the Attempto project, running RACE should not require any knowledge of theorem proving in general, and of the working of RACE in particular. Nevertheless, RACE offers a number of parameters that let users control the deductions from collective plurals, enable or disable the output of the auxiliary axioms that were used during a proof, and limit the search for proofs. These parameters have default settings that allow the majority of the users to ignore the parameters.

RACE processes clauses by forward-chaining whose worst-case time complexity is $O(n^2)$, where n is the number of clauses. To reduce the run-time of RACE we need to reduce primarily the number of clauses that participate:

- we profit from simplifications introduced in the DRS representation that lead to fewer clauses
- we use clause compaction
- we eliminate after the first round of forward reasoning the clauses with the body *true* that cannot be fired again
- we apply intelligent search for clauses that could be fired in the next round of forward reasoning
- we use complement splitting — given a disjunction $(A \vee B)$, one investigates $(A \wedge \neg B)$, respectively $(\neg A \wedge B)$ — though complement splitting is not guaranteed to increase the efficiency in each case

RACE can be called via a SOAP web service⁸ and can conveniently be accessed via a web client⁹. Figure 2 is a typical screenshot of the RACE web client.

⁸ http://attempto.ifi.uzh.ch/site/docs/race_webservice.html

⁹ see <http://attempto.ifi.uzh.ch/race/> and http://attempto.ifi.uzh.ch/site/docs/race_webclient_help.html

The screenshot displays the RACE web interface. At the top, there are two buttons: "Show Parameters" and "Show Help". Below these, the "Axioms" section contains the text: "Every man is a human. Every woman is a human. Mary is a woman. John is a man." To the right, a "Parameters" dialog box is open, showing a list of distributive deduction rules with checkboxes. The checked rules are: "subject of intransitive verb (si)", "object of transitive verb (ot)", "direct object of ditransitive verb (dodt)", and "there is/are construct (sti)". Below the dialog, there are buttons for "Check Consistency", "Prove", and "Answer Query". The "Query" section contains the text: "Is somebody who is a man a human?" and an "Answer Query" button. Below the query, it shows the overall time (1.553 sec) and RACE time (0.21 sec). The results section, highlighted in green, shows the axioms, query, and parameters used. Below this, it lists the minimal subsets of axioms that answer the query: Subset 1, which includes "1: Every man is a human." and "4: John is a man."

Buttons: Show Parameters, Show Help

Axioms
Every man is a human. Every woman is a human.
Mary is a woman. John is a man.

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 parameters

- ☐ show raw proofs & auxiliary axioms (aux)
- ☐ show first proofs only (fpo)

Buttons: Check Consistency, Prove, Answer Query

Query
Is somebody who is a man a human?

Buttons: Answer Query

overall time: 1.553 sec; RACE time: 0.21 sec

Axioms: Every man is a human. Every woman is a human. Mary is a woman. John is a man.
Query: Is somebody who is a man a human?
Parameters: si ot dodt sti

The following minimal subsets of the axioms answer the query:

- Subset 1
 - 1: Every man is a human.
 - 4: John is a man.

Fig. 2. The web interface of RACE showing how to answer an ACE query from ACE axioms with the default setting of the parameters

6.5.2 ACE View Ontology and Rule Editor

The ACE View ontology and rule editor¹⁰ allows users to develop OWL ontologies and SWRL rulesets in ACE. The ACE View editor lets the user create, browse, and edit an ACE text, and query both its asserted and automatically entailed content.

In the context of ACE View, an ACE text is a set of ACE snippets where each snippet is a sequence of one or more (possibly anaphorically linked) ACE sentences. In general, each snippet corresponds to an OWL or SWRL axiom, but complex ACE sentences that involve sentence conjunction can map to several axioms. When a snippet is added to the text, it is automatically parsed and converted into OWL/SWRL. If the translation fails then the snippet is still accepted, but as it does not have any logical axioms attached, it cannot be considered as part of the text during reasoning. In case the translation succeeds, the snippet is mapped to one or more OWL axioms and SWRL rules which are in turn merged with the underlying knowledge base representation. In case a snippet is deleted from the text, its corresponding axioms (if present) are removed from the knowledge base.

¹⁰ <http://attempto.ifi.uzh.ch/aceview/>

ACE View is implemented as an extension to the Protégé editor¹¹. Therefore, the ACE View user can alternatively switch to one of the default Protégé views to perform an ontology editing task. In case an axiom is added in a Protégé view, then the axiom is automatically verbalized into an ACE snippet which in turn is merged into the ACE text. If the verbalization fails (e.g. the verbalizer does not support the *FunctionalProperty*-axiom with data properties) then an error message is stored and the axiom is preserved in the ACE text in Manchester OWL Syntax. In case an axiom is deleted, then its corresponding snippet is deleted as well.

The ACE text (and thus the ontology) can be viewed and edited at several levels — word, snippet, vocabulary, text.

- Word level provides an access to OWL entities in the ontology and allows one to specify how the entities should appear in ACE sentences, i.e. what are the surface forms (e.g. singular and plural forms) of the corresponding words.
- Snippets can be categorized as asserted declarative snippets, asserted interrogative snippets (i.e. questions) and entailed (declarative) snippets. Asserted snippets are editable and provide access to their details (parsing results such as error messages or syntax trees/syntax aware layout, corresponding axioms/rules, ACE paraphrase). Questions provide additionally answers. Entailed snippets are not editable but can be explored to find out the reasons that cause the entailment.
- Vocabulary is a set of ACE content words. It can be sorted alphabetically or by frequency of usage. As content words correspond to OWL entities, standard Protégé views offer even more presentation options, e.g. the “back-bone hierarchy” of subclass and “part of” relations; separation of the vocabulary into classes, properties, individuals. The vocabulary level provides a quick access to the word level, each selected/searched word (entity) can be automatically shown in the word level, or its corresponding snippets in the text level.
- An ACE text is a set of ACE snippets. This set can be filtered, sorted, and searched. Reasoning can be performed on the whole text to find out about its (in)consistency. A new text can be generated by filling it with snippets that the asserted text entails.

The ACE View user interface comprises several “views” that allow for browsing and editing of the ACE text at all the described levels (see figures 3 and 4). In the “Lexicon view” and “Words view”, the complete content word vocabulary of the ACE text is presented, sorted either alphabetically or by frequency of usage. The “Lexicon view” allows the user to edit the surface forms (singular, plural, past participle) of words and make sure that they all correspond to the same OWL entity. When a new entity is generated in the standard Protégé views, the surface forms of its corresponding content word are automatically

¹¹ <http://protege.stanford.edu/>

The screenshot displays the ACE View editor interface, which is divided into four main panels:

- ACE Snippet Editor:** Shows a selected snippet: "Every territory that is surrounded by a country is an enclave." It includes buttons for "Add as new", "Update", "Delete", "Annotate", and "Why?".
- ACE Feedback:** Displays feedback for the selected snippet, including messages (0 errors, 0 warnings), paraphrases (1), annotations (3), and corresponding logical axioms (OWL: 1, SWRL: 0). It also lists similar snippets (1).
- ACE Q&A:** Shows a table of 29 snippets. The selected snippet is "Which country is an enclave?". The table has columns for Question, Individual, Subclass, and Superclass. Below the table, it lists 2 named individuals: San_Marino and Vatican_City.
- ACE Explanation:** Provides a justification for the answer "Vatican_City is a country that is an enclave". It includes two explanations: Explanation 2 (snippet count: 7) and Explanation 1 (snippet count: 7). Explanation 1 is expanded, showing a chain of reasoning: "Every country is a territory. Every European-country is a country that Europe contains. Italy is a NATO-country. Vatican_City is a European-country. Every territory that is surrounded by a country is an enclave. Every NATO-country is a country. Vatican_City is surrounded by Italy."

Fig. 3. One possible layout of the ACE View editor. Several views are shown: ACE Snippet Editor shows the currently selected snippet; ACE Feedback shows its paraphrase, annotations, the corresponding OWL axiom, and a list of syntactically similar snippets; Q&A view shows all the entered questions, and the answers to the question ‘Which country is a an enclave?’; ACE Explanation shows the justification for the answer ‘Vatican.City is a country that is an enclave’. The justification contains two sets of snippets (i.e. different explanations), one of which is expanded.

generated based on rules of English morphology. The user can override these forms if needed.

The “Snippets view” organizes all the asserted snippets in a table. With each snippet a set of its features are presented: snippet length (in content words), creation time, number of annotations, etc. The table rows can be highlighted and filtered based on the selected word, presenting only the snippets that contain the word. The “Snippet Editor” lets the user to edit an existing snippet, or create a new one. The “Feedback view” shows the logical and linguistic properties of the selected snippet, and meta information such as annotations for the snippet. For sentences that fail to map to OWL/SWRL, error messages are provided. Error messages point to the location of the error and explain how to deal with the problem.

The “Q&A view” lists ACE questions and answers to them. These questions correspond to DL-Queries which are essentially (possibly complex) class expressions. The answers to a DL-Query are named individuals (members of the queried class) or named classes (named super and subclasses of the queried class). In ACE terms, the answers are ACE content words — proper names and common nouns. While the answers to DL-Queries are representation-wise

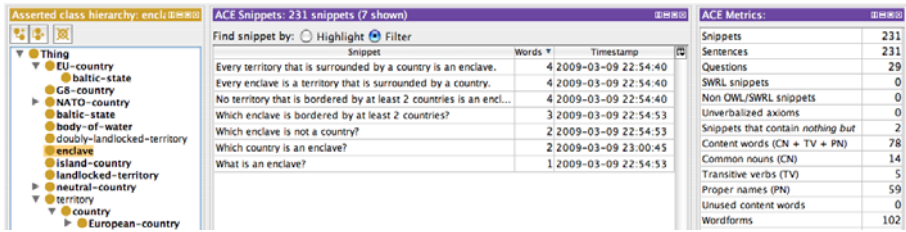


Fig. 4. Another possible layout of the ACE View editor. Several views are shown: the standard Protégé tree view shows the subclass hierarchy of named classes; ACE Snippets view shows the snippets that reference the selected entity ‘enclave’, the number of content words and the creation time is shown for each snippet; Metrics view shows various (mostly linguistic) metrics of the ACE text.

identical in the ACE view and in the standard Protégé view, the construction of the query is potentially much simpler in the ACE view, as one has to construct a natural language question.

The “Entailments view” provides a list of ACE sentences that follow logically from the ACE text, i.e. these sentences correspond to the entailed axioms of the ontology. Such axioms are generated by the integrated reasoner on the event of classification. The “Explanation view” provides an “explanation” for a selected entailed snippet. Such an explanation is a (minimal) sequence of asserted snippets that justify the entailment. Presenting a tiny fragment of the ontology which at the same time is sufficient to cause the entailment greatly improves the understanding of the reason behind the entailment.

ACE View is implemented as a plug-in for Protégé 4 and relies heavily on the OWL API [20] that provides a connection to reasoners, entailment explanation support, storage of OWL axioms and SWRL rules in the same knowledge base, etc. The main task of the ACE View plug-in, translating to and from OWL/SWRL, is performed by two external translators — APE web service (see section 6.3.3) and OWL verbalizer¹². The entity surface forms are automatically generated using SimpleNLG¹³.

6.5.3 AceWiki: ACE in a Semantic Wiki

AceWiki¹⁴ is a semantic wiki that uses ACE to represent its content. Figure 5 shows a screenshot of the AceWiki interface. Semantic wikis combine the philosophy of wikis (i.e. quick and easy editing of textual content in a collaborative way over the Web) with the concepts and techniques of the Semantic Web (i.e. giving information well-defined meaning in order to enable computers and people to work in cooperation). The general goal of semantic wikis is to manage formal representations within a wiki environment.

¹² http://attempto.ifi.uzh.ch/site/docs/owl_to_ace.html

¹³ <http://www.csd.abdn.ac.uk/~ereiter/simplenlg/>

¹⁴ See [27], [28], and <http://attempto.ifi.uzh.ch/acewiki>

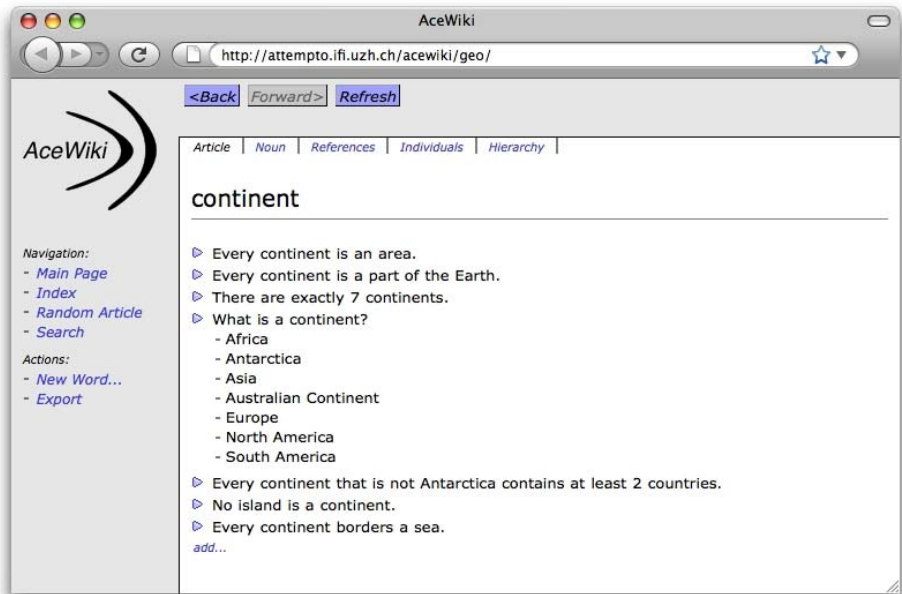


Fig. 5. This screenshot shows an AceWiki article about the concept ‘continent’. The content of the article is written in ACE.

There exist many different semantic wiki systems. Semantic MediaWiki [25], IkeWiki [34], and OntoWiki [1] belong to the most mature existing semantic wiki engines. Unfortunately, none of the existing semantic wikis supports expressive ontology languages in a general way. For example, none of them allows the users to define general concept inclusion axioms like ‘every country that borders no sea is a landlocked country’. Furthermore, most of the existing semantic wikis fail to hide the technical aspects and are hard to understand for people who are not familiar with the technical terms.

AceWiki tries to solve these problems by using controlled natural language. Ordinary people who have no background in logic should be able to understand, modify, and extend the formal content of a wiki. Instead of enriching informal content with semantic annotations (as many other semantic wikis do), AceWiki treats the formal statements as the primary content of the wiki articles. The use of controlled natural language allows us to express also complex axioms in a natural way.

The goal of AceWiki 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. Since ACE is a subset of natural English, every English speaker can immediately read and understand the content of the wiki. In order to enable easy creation of ACE sentences, the users are supported by an intelligent predictive text editor [29] that is able to

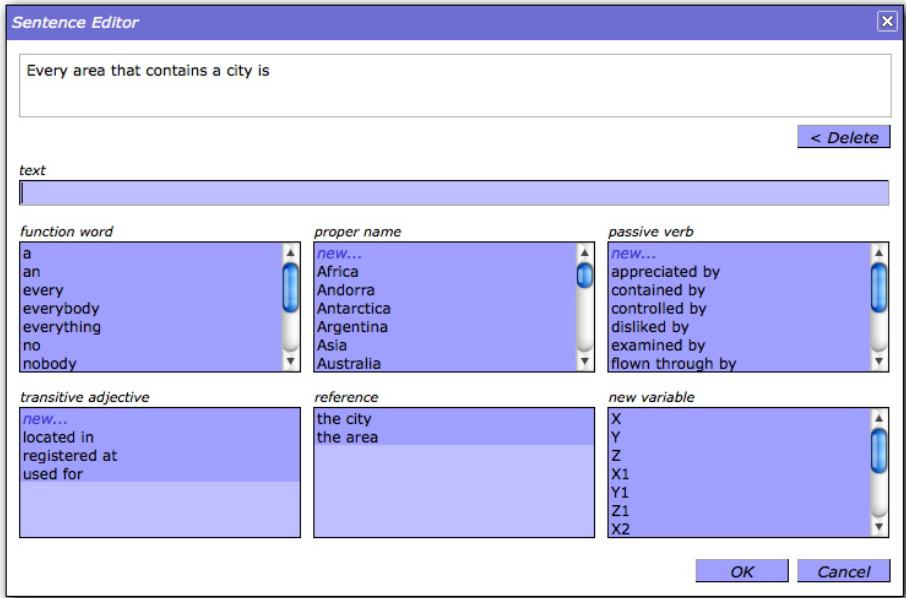


Fig. 6. A screenshot of the predictive editor of AceWiki. The partial sentence ‘Every area that contains a city is ...’ has already been entered and now the editor shows all possibilities to continue the sentence. The possible words are arranged by their type in different menu boxes.

look ahead and to show the possible words to continue the sentence. Figure 6 shows a screenshot of this editor.

In AceWiki, words have to be defined before they can be used. At the moment, five types of words are supported: proper names, nouns, transitive verbs, *of*-constructs (i.e. nouns that have to be used with *of*-phrases), and transitive adjectives (i.e. adjectives that require an object). Figure 7 shows the lexical editor of AceWiki that helps the users in creating and modifying word forms in an appropriate way.

Most sentence that can be expressed in AceWiki can be translated into OWL. Some examples are shown here:

- ▶ **Every country that borders no sea is a landlocked-country.**
- ▶ **Switzerland borders exactly 5 countries.**
- ▶ **No city that is located in Europe is controlled by the USA.**
- ▶ **If X borders Y then Y borders X.**
- ▶ **Every moon orbits a planet or orbits a dwarf-planet.**

AceWiki relies on the ACE→OWL translation that has been introduced in Section 6.4.1. The OWL reasoner Pellet¹⁵ is seamlessly integrated into AceWiki,

¹⁵ <http://clarkparsia.com/pellet/>

Word Creator

Proper Name Noun Of-Construct **Verb** Transitive Adjective

 Every verb represents a certain relation between things. For example, the verb "owns" relates persons to their possessions. Every verb has a third singular form, a bare infinitive form, and an optional past participle form.

third singular *

examples: owns, applies to, touches

bare infinitive *

examples: own, apply to, touch

past participle

examples: owned, applied to, touched

* required field

OK Cancel

Fig. 7. The lexical editor of AceWiki helps the users to define the word forms. The example shows how a new transitive verb — “organize” in this case — is created.

so that reasoning can be done within the wiki environment. Since only OWL compliant sentences can be considered for reasoning, the sentences that are outside of OWL are marked with a red triangle:

- ▶ No ocean borders every continent.
- ▶ Every person that has a car owns the car or leases the car.
- ▶ If Berlin is a capital then Germany is a stable country.
- ▶ Every trip that starts at X and that ends at X is a round trip.

In this way, it is easy to explain to the users that only the statements that are marked by a blue triangle are considered when the reasoner is used. We plan to provide an interface that allows skilled users to export the formal content of the wiki and to use it within an external reasoner or rule-engine. Thus, even though the statements that are marked by a red triangle cannot be interpreted by the built-in reasoner they can still be useful.

Consistency checking plays a crucial role because any other reasoning task requires a consistent ontology in order to return useful results. In order to ensure that the ontology is always consistent, AceWiki checks every new sentence — immediately after its creation — whether it is consistent with the current ontology. Otherwise, the sentence is not included in the ontology:

- ▶ Every country is a part of exactly 1 continent.
- ▶ Every country that borders Switzerland is a part of Europe.
- ▶ Germany borders Switzerland.
- ▶ Germany is a part of Asia.

After the user created the last sentence of this example, AceWiki detected that it contradicts the current ontology. The sentence is included in the wiki article but the red font indicates that it is not included in the ontology. The user can remove this sentence, or keep it and try to reassert it later when the rest of the ontology has changed.

Not only asserted but also inferred knowledge can be represented in ACE. At the moment, AceWiki can show inferred class hierarchies and class memberships. Furthermore, AceWiki supports queries that are formulated in ACE and evaluated by the reasoner:

- ▶ Which cities are located in a country that borders Switzerland?
 - Berlin
 - Milano
 - Paris
 - Rome
 - Vienna

Thus, ACE is used not only as an ontology- and rule-language, but also as a query-language.

A usability experiment [27] showed that people with no background in formal methods are able to work with AceWiki and its predictive editor. The participants — without receiving instruction on how to use the interface — were asked to add general and verifiable knowledge to AceWiki. About 80% of the resulting sentences were semantically correct and sensible statements (in respect of the real world). More than 60% of those correct sentences were complex in the sense that they contained an implication or a negation.

6.5.4 Protune

Chapter 4 describes the Protune framework in detail. Here we simply provide a general overview of the Protune framework, and especially focus on the role of ACE in this framework by building on the concepts introduced in Section 6.4.3.

The Protune framework aims at providing a complete solution for all aspects of policy definition and policy enforcement. Special attention has been given to the interaction with users, be they policy authors or end-users whose requests have been accepted or rejected. For policy authors a set of tools is available to ease the task of defining policies. For end-users a number of facilities are provided to explain why a request was accepted or rejected.

In the following we describe the tools provided by the Protune framework for policy authors, namely

Protune editor: It allows advanced users to exploit the full power of the Protune language by directly providing Protune code

ACE front-end for Protune: It enables users familiar with ACE but not with Protune to define Protune policies

Predictive editor: It provides a user interface which guides non-expert users toward the definition of syntactically correct policies (under development)

Advanced users can exploit the Protune editor for policy authoring. The editor helps them to avoid annoying syntactical errors, and provides facilities like syntax highlighting, visualization of error/warning/todo messages, automatic completion, outlining, as well as other facilities that come for free with a rich client platform. A demo of the Protune editor can be found online¹⁶.

The Protune editor is intended for users who already have some knowledge of the Protune policy language. For others users an ACE front-end for Protune has been developed that allows them to define policies by means of the subset of ACE described in Section 6.4.3. Such natural language policies can then be automatically translated into semantically equivalent Protune policies, and enforced by the Protune framework. The ACE→Protune compiler provides a command-line interface that translates an input ACE policy into the corresponding Protune policy, or if an error occurs, shows error messages like

Within the scope of negation-as-failure only one single predicate is allowed.

or

Only “be” can be used as relation in the “then”-part of an implication.

Messages like these are shown if a syntactically correct ACE sentence cannot be translated into a valid policy. For incorrect ACE sentences the error messages provided by APE (cf. 6.3.3) are returned to the user.

The command-line interface that we just described assumes that the user is already familiar with ACE. For unexperienced users a predictive editor like the one described in Section 6.5.3 would be more advisable. A predictive editor for the subset of ACE defined in Section 6.4.3 is in development.

Although the facilities described above have been designed in order to target different categories of users, they can benefit any user. Expert users might want to exploit the ACE front-end for Protune in order to define policies in a more intuitive way and maybe fine-tune the automatically generated Protune policies later. On the other hand, novice users might want to switch from the predictive editor to the command-line interface as soon as they get sufficiently familiar with the ACE language.

6.6 Conclusions

We showed how controlled natural languages in general and ACE in particular can bridge the usability gap between the complicated Semantic Web machinery

¹⁶ <http://policy.13s.uni-hannover.de:9080/policyFramework/protune/>

and potential end users with no experience in formal methods. Many tools have been developed around ACE in order to use it as a knowledge representation and reasoning language for the Semantic Web, and for other applications.

The ACE parser is the most important tool. It translates ACE texts into different forms of logic, including the Semantic Web standards OWL and SWRL. AceRules shows how ACE can be used as a practical rule language. We presented RACE that is a reasoner specifically designed for reasoning in ACE. AceWiki demonstrates how controlled natural language can make semantic wikis at the same time expressive and very easy to use. We showed how ACE can help in defining policies by providing a front-end for the Protune policy language. Last but not least, ACE View is an ontology editor that shows how ontologies can be managed conveniently in ACE. The large number of existing tools exhibits the maturity of our language.

Evaluation of the AceWiki system showed that ACE is understandable and usable even for completely untrained people. More user studies are planned for the future.

If the vision of the Semantic Web should become a reality then we have to provide user-friendly interfaces. The formal nature of controlled natural languages enables to use them as knowledge representation languages, while preserving readability. Our results show how controlled natural language can bring the Semantic Web closer to its potential end users.

References

1. Auer, S., Dietzold, S., Riechert, T.: OntoWiki — A Tool for Social, Semantic Collaboration. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 736–749. Springer, Heidelberg (2006)
2. Bernardi, R., Calvanese, D., Thorne, C.: Lite Natural Language. In: IWCS-7 (2007)
3. Bernstein, A., Kaufmann, E., Fuchs, N.E.: Talking to the Semantic Web — A Controlled English Query Interface for Ontologies. AIS SIGSEMIS Bulletin 2(1), 42–47 (2005)
4. Bernstein, A., Kaufmann, E., Fuchs, N.E., von Bonin, J.: Talking to the Semantic Web — A Controlled English Query Interface for Ontologies. In: 14th Workshop on Information Technology and Systems, December 2004, pp. 212–217 (2004)
5. Blackburn, P., Bos, J.: Working with Discourse Representation Structures. In: Representation and Inference for Natural Language: A First Course in Computational Linguistics, vol. 2 (September 1999)
6. Bos, J.: Computational Semantics in Discourse: Underspecification, Resolution, and Inference. *Journal of Logic, Language and Information* 13(2), 139–157 (2004)
7. 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)
8. De Coi, J.L.: Notes for a possible ACE $\rightarrow$ Protune mapping. Technical report, Forschungszentrum L3S, Appelstr. 9a, 30167 Hannover (D) (July 2008)
9. De Coi, J.L., Olmedilla, D.: A Review of Trust Management, Security and Privacy Policy Languages. In: Proceedings of the 3rd International Conference on Security and Cryptography (SECRYPT 2008). Springer, Heidelberg (2008)

10. 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.) 3rd OWL Experiences and Directions Workshop (OWLED 2007). CEUR Proceedings, vol. 258 (2007)
11. Dimitrova, V., Denaux, R., Hart, G., Dolbear, C., Holt, I., Cohn, A.: Involving Domain Experts in Authoring OWL Ontologies. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 1–16. Springer, Heidelberg (2008)
12. Fuchs, N.E., Kaljurand, K., Kuhn, T.: Discourse Representation Structures for ACE 6.0. Technical Report ifi-2008.02, Department of Informatics, University of Zurich, Zurich, Switzerland (2008)
13. Funk, A., Davis, B., Tablan, V., Bontcheva, K., Cunningham, H.: D2.2.2 Report: Controlled Language IE Components version 2. Technical report, University of Sheffield (2007)
14. 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.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ISWC 2007. LNCS, vol. 4825, pp. 142–155. Springer, Heidelberg (2007)
15. 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)
16. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9, 365–385 (1990)
17. Grosz, B.N.: Courteous logic programs: Prioritized conflict handling for rules. Technical Report RC 20836, IBM Research, IBM T.J. Watson Research Center (December 1997)
18. Hart, G., Johnson, M., Dolbear, C.: Rabbit: Developing a Controlled Natural Language for Authoring Ontologies. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 348–360. Springer, Heidelberg (2008)
19. Hirtle, D.: TRANSLATOR: A TRANSLator from LAnguage TO Rules. In: Canadian Symposium on Text Analysis (CaSTA), Fredericton, Canada (October 2006)
20. Horridge, M., Bechhofer, S., Noppens, O.: Igniting the OWL 1.1 Touch Paper: The OWL API. In: Golbreich, C., Kalyanpur, A., Parsia, B. (eds.) 3rd OWL Experiences and Directions Workshop (OWLED 2007). CEUR Proceedings, vol. 258 (2007)
21. Horridge, M., Drummond, N., Goodwin, J., Rector, A., Stevens, R., Wang, H.H.: The Manchester OWL Syntax. In: 2nd OWL Experiences and Directions Workshop (OWLED 2006) (2006)
22. Kaljurand, K.: Attempto Controlled English as a Semantic Web Language. PhD thesis, Faculty of Mathematics and Computer Science, University of Tartu (2007)
23. Kalyanpur, A., Parsia, B., Sirin, E., Grau, B.C.: Repairing unsatisfiable concepts in OWL ontologies. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 170–184. Springer, Heidelberg (2006)
24. Kamp, H., Reyle, U.: From Discourse to Logic. Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory. Kluwer Academic Publishers, Dordrecht (1993)
25. Krötzsch, M., Vrandečić, D., Völkel, M., Haller, H., Studer, R.: Semantic Wikipedia. *Web Semantics: Science, Services and Agents on the World Wide Web* 5(4), 251–261 (2007)

26. Kuhn, T.: AceRules: Executing Rules in Controlled Natural Language. In: Marchiori, M., Pan, J.Z., de Sainte Marie, C. (eds.) RR 2007. LNCS, vol. 4524, pp. 299–308. Springer, Heidelberg (2007)
27. Kuhn, T.: AceWiki: A Natural and Expressive Semantic Wiki. In: Semantic Web User Interaction at CHI 2008: Exploring HCI Challenges (2008)
28. Kuhn, T.: AceWiki: Collaborative Ontology Management in Controlled Natural Language. In: Proceedings of the 3rd Semantic Wiki Workshop. CEUR Proceedings, vol. 360 (2008)
29. Kuhn, T., Schwitter, R.: Writing Support for Controlled Natural Languages. In: Proceedings of the Australasian Language Technology Workshop (ALTA 2008) (2008)
30. Lukichev, S., Wagner, G., Fuchs, N.E.: Deliverable I1-D11. Tool Improvements and Extensions 2. Technical report, REWERSE (2007), <http://rewerse.net/deliverables.html>
31. Manthey, R., Bry, F.: SATCHMO: A Theorem Prover Implemented in Prolog. In: Lusk, E., Overbeek, R. (eds.) CADE 1988. LNCS, vol. 310, pp. 415–434. Springer, Heidelberg (1988)
32. Pratt-Hartmann, I.: A two-variable fragment of English. *Journal of Logic, Language and Information* 12(1), 13–45 (2003)
33. Rector, A.L., Drummond, N., Horridge, M., Rogers, J., Knublauch, H., Stevens, R., Wang, H., Wroe, C.: OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns. In: Motta, E., Shadbolt, N.R., Stutt, A., Gibbins, N. (eds.) EKAW 2004. LNCS, vol. 3257, pp. 63–81. Springer, Heidelberg (2004)
34. Schaffert, S.: IkeWiki: A Semantic Wiki for Collaborative Knowledge Management. In: Proceedings of the First International Workshop on Semantic Technologies in Collaborative Applications (STICA 2006), pp. 388–396 (2006)
35. 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 2008), DC, Washington, April 1-2 (2008)
36. 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)
37. Sowa, J.F.: Knowledge Representation: Logical, Philosophical, and Computational Foundations. Brooks Cole Publishing Co., Pacific Grove (2000)
38. Sowa, J.F.: Common Logic Controlled English. Technical report (2007), <http://www.jfsowa.com/clce/clce07.htm> (Draft, March 15, 2007)
39. Wagner, G.: Web Rules Need Two Kinds of Negation. In: Bry, F., Henze, N., Małuszyński, J. (eds.) PPSWR 2003. LNCS, vol. 2901, pp. 33–50. Springer, Heidelberg (2003)
40. Wagner, G., Giurca, A., Lukichev, S.: A Usable Interchange Format for Rich Syntax Rules Integrating OCL, RuleML and SWRL. In: Hitzler, P., Wache, H., Eiter, T. (eds.) RoW 2006 Reasoning on the Web Workshop at WWW 2006 (2006)