I2-D13

Reasoning, Rules and Semantic Wikis

Project title:	Reasoning on the Web with Rules and Semantics
Project acronym:	REWERSE
Project number:	IST-2004-506779
Project instrument:	EU FP6 Network of Excellence (NoE)
Project thematic priority:	Priority 2: Information Society Technologies (IST)
Document type:	D (deliverable)
Nature of document:	R/P (report and prototype)
Dissemination level:	PU (public)
Document number:	IST506779/Zurich/I2D13/D/PU
Responsible editor:	Norbert E. Fuchs
Reviewers:	Sergey Lukichev (I1), Juri Luca De Coi (I2)
Contributing participants:	University of Zurich
Contributing workpackages:	I2
Contractual date of delivery:	15 September 2007
Actual date of delivery:	15 October 2007

Abstract

This report describes several tracks of research on Attempto Controlled English (ACE). First, we present the extensions of ACE and its tools in versions 5.5 and 6. Second, we elaborate on data structures and operations in ACE 6. Third, we introduce AceWiki. Fourth, we describe the ACE view plug-in for the Protégé OWL editor. Fifth, we present on-going work on the Attempto reasoner RACE. Sixth, we summarise updates on AceRules and on DRACE. Seventh, we list cooperations and visits.

Keyword List

Attempto Controlled English, ACE, AceWiki, ACE view plug-in, Protégé, OWL, Attempto reasoner, RACE, AceRules, DRACE

Project co-founded by the European Commission and the Swiss State Secretariat for Education and Research within the Sixth Framework Programme

Reasoning, Rules and Semantic Wikis

Norbert E. Fuchs, Kaarel Kaljurand, Tobias Kuhn

Department of Informatics

&

Institute of Computational Linguistics

University of Zurich

Email: {fuchs, kalju, tkuhn}@ifi.uzh.ch

15 October 2007

Abstract

This report describes several tracks of research on Attempto Controlled English (ACE). First, we present the extensions of ACE and its tools in versions 5.5 and 6. Second, we elaborate on data structures and operations in ACE 6. Third, we introduce AceWiki. Fourth, we describe the ACE view plug-in for the Protégé OWL editor. Fifth, we present on-going work on the Attempto reasoner RACE. Sixth, we summarise updates on AceRules and on DRACE. Seventh, we list cooperations and visits.

Keyword List

Attempto Controlled English, ACE, AceWiki, ACE view plug-in, Protégé, OWL, Attempto reasoner, RACE, AceRules, DRACE

Contents

1. INTRODUCTION 5

2. EXTENSIONS OF ATTEMPTO CONTROLLED ENGLISH AND ITS TOOLS 6

2.1. FROM ACE 5 TO ACE 5.5 6
2.2. TOWARDS ACE 6 7

3. DATA STRUCTURES AND OPERATIONS 8

3.1. INTEGERS, REALS, AND STRINGS 8
3.2. OPERATORS 8
3.3. FORMULAS 9
3.4. LISTS AND SETS 9

4. ACEWIKI 10

4.1. INTRODUCTION 10
4.2. NATURALNESS 10
4.3. STRICT USER GUIDANCE 11
4.4. SYNTAX BOXES 12
4.5. CONCLUSIONS AND FUTURE WORK 13

5. ACE VIEW PLUG-IN FOR THE PROTÉGÉ OWL EDITOR 14

5.1. INTRODUCTION 14
5.2. PROTÉGÉ AND ACE VIEW 14
5.3. TABS 14
5.3.1. Main 15
5.3.2. Index 15
5.3.3. Paraphrase 16
5.3.4. Inferences 16
5.3.5. Answers 17
5.3.6. Debug 17
5.4. CONCLUSION 18

6. REASONING IN ACE 19

6.1. RACE IN A NUTSHELL 19
6.2. ADAPTATIONS TO ACE 6 19
6.3. CONTROLLING DEDUCTIONS 19
6.4. INCREASING EFFICIENCY 20
6.5. ALTERNATIVE DEDUCTIONS 21

7. UPDATES 22

7.1. ACERULES 22
7.2. DRACE 22

8. COOPERATIONS AND VISITS 23

9. DELIVERABLE I2-D15 24

10. REFERENCES 25

1. Introduction

Research on Attempto Controlled English (ACE) has progressed as planned. In this deliverable we present

- a host of new features in versions 5.5 and 6 of ACE
- data structures and operations on them introduced in ACE 6
- AceWiki
- ACE view plug-in for the Protégé OWL editor
- ACE reasoner RACE
- updates to AceRules and DRACE
- contacts with REWERSE internal and external partners

The task "why not questions" scheduled for this deliverable was postponed to deliverable I2-D15.

2. Extensions of Attempto Controlled English and Its Tools

2.1. From ACE 5 to ACE 5.5

In version 5.5 of Attempto Controlled English (ACE), we introduced the following new language features:

- numbers and strings as general objects, e.g.

```
John's address is "Paris".
The temperature reaches -2.
3.14 badly approximates Pi.
```

- positive, comparative and superlative adjectives, e.g.

```
John is tall.
John is as tall as Mary.
Mary is taller.
Mary is taller than John.
Mary is tallest.
```

- positive, comparative and superlative adverbs, e.g.

```
John runs fast.
Mary runs faster.
Mary runs fastest.
```

- simple form of imperatives, e.g.

`John, enter a card!` (translated as `John enters a card.`)

- generalisation: any noun phrase can now take a relative phrase

Concerning the lexicon, there were the following improvements:

- clean-up of the built-in lexicon
- new lexicon format[1]
- improved error handling for the user lexicon

The Attempto Parsing Engine (APE) gained from

- extended error reporting
- improved error representation in the web-interface

	Type	Sentence	Problem	Suggestion
error	sentence	1	Every dog likes meat .	Add a determiner before: meat

	Type	Sentence	Problem	Suggestion
warning	anaphor	1	Undefined: SonicHedgeHog	Interpreted as neuter proper name. Should this be a proper name or variable, or is a determiner missing?

Changes to the DRS verbalisation component (DRACE) that is used, for instance, as paraphraser of APE:

- DRACE now uses the ACE lexicon
- improved paraphrase of ACE input

Changes of the DRS representation[2]:

- major overhaul leading to simplifications
- typed and untyped representations

[1] http://attempto.ifi.uzh.ch/site/docs/ace_lexicon.html

[2] http://attempto.ifi.unizh.ch/site/pubs/papers/drs_report_55.pdf

- extended representation of adjectives and adverbs

ACE-to-OWL translation:

- added support for OWL 1.1 (data properties partially supported)
- two outputs provided: OWL 1.1 functional-style syntax and OWL 1.1 RDF/XML syntax

2.2. Towards ACE 6

Currently we are preparing the release of ACE 6 that will, among other features, add to Attempto Controlled English

- arithmetic expressions (see section 3)
- formulas (see section 3)
- lists and sets (see section 3)
- generalised quantifier `exactly`

  `Exactly four men wait.`
- `nothing but` that can be followed by a bare mass noun or a bare plural noun – and thus functions as determiner – or by a proper name

  `Every carnivore eats nothing but meat.` (translated as `If there is a carnivore then if the carnivore eats something X then X is some meat.`)
  `John has nothing but apples.` (translated similarly as the preceding sentence)
  `Mary likes nothing but John.` (translated similarly as the preceding sentence)
- `all` & plural

  `All men wait.` (translated as `Every man waits.`)
- `no` & plural

  `No men wait.` (translated as `No man waits.`)
- Saxon genitive for indefinite pronouns and variables, e.g. `somebody's`, `somebody X's`, `X's`

  `Somebody's dog barks.`

The Attempto Parsing Engine (APE) and the DRS representation are adapted accordingly.

3. Data Structures and Operations

The extensions that are described in this section have been developed for ACE 6 and are not yet public. There might be slight changes until the next release.

3.1. Integers, Reals, and Strings

The next release will bring a simplified representation for integers, reals, and strings. Until now, they have been represented as separate conditions. E.g. the sentence

```
The length of “abc” is 3.
```

was represented as

```
[A, B, C, D]
data(A, 3, integer)
predicate(B, be, C, A)
relation(C, of, D)
data(D, abc, string)
object(C, length, countable, na, eq, 1)
```

With the new representation, no `data/3` predicates are needed anymore. Instead, the numbers and strings are inserted directly at the respective positions in the other predicates:

```
[A, B]
predicate(A, be, B, int(3))
relation(B, of, string(abc))
object(B, length, countable, na, eq, 1)
```

This leads to a decrease of the number of conditions (in this case from 5 to 3) and of the number of discourse referents (in this case from 4 to 2). It makes the remaining conditions only slightly more complex. The numbers and strings are put inside of the functions `int/1`, `real/1`, and `string/1`, respectively. This makes it easier to process the DRSs and to find potential errors. Furthermore, other primitive types can be added in the future without breaking backwards compatibility.

3.2. Operators

ACE defines at the moment five different operators: `+`, `–`, `*`, `/`, and `&`. The four operators `+`, `–`, `*`, and `/` stand for addition, subtraction, multiplication, and division, respectively. Operators are used to construct expressions using integers, reals, variables, proper names, and other expressions. Thus, complex expressions can be built using simpler ones. Such expressions can appear on any valid noun-phrase position of an ACE sentence:

```
5 is 3 + 2.
4 / 2 exceeds 3.5 - 0.6.
A value is 5000 * Discount-Rate.
If there are a number X and a number Y then the average of X and Y is (X
+ Y) / 2.
If a circle's radius is a value R then the circumference of the circle is
2 * Pi * R.
```

Note that "Discount-Rate" and "Pi" are interpreted as proper names, whereas "X", "Y", and "R" are variables.

Parentheses can be used to achieve the intended structure.

```
A value is ( 13 - ( X + 3 ) ) / 2.
```

If used without parentheses `*` and `/` bind stronger than `+` and `–`. Operators of the same priority are processed from left to right (left-associative). Thus

```
A number X is 9 - 4 + 3 * 2.
```

is interpreted as

```
A number X is ( 9 - 4 ) + ( 3 * 2 ).
```

The operator `&` is used for string concatenation and cannot be applied to numbers.

```
John's email-address is "john" & "@" & "mail.com".
```

In the DRS, the operators introduce complex nested terms (using `expr/3`). For example, the sentence

```
The length of "abc" & "123" is 4 + 6 / 3.
```

leads to the DRS

```
[A, B]
predicate(A, be, B, expr(+, int(4), expr(/, int(6), int(3))))-1
relation(B, of, expr(&, string(abc), string(123)))-1
object(B, length, countable, na, eq, 1)-1
```

3.3. Formulas

The boolean connectors = (equal), < (less than), > (greater than), =< (less than or equal), and >= (greater than or equal) can be used to construct a formula using two expressions. Formulas can occur at any sentence position within an ACE text. For example, they can be used as complete ACE sentences or in the if- or then-part of conditional sentences:

```
3 * 4 = 10 + 2.
If there are a number X and a number Y and X + 1 >= Y then Y - 1 =< X.
```

Formulas are represented in the DRS using the predicate `formula/3`. The first example would lead to:

```
formula(expr(*, int(3), int(4)), =, expr(+, int(10), int(2)))
```

3.4. Lists and Sets

We added also support for lists and sets. For lists, square brackets are used; for sets, we use curly brackets. In both cases, the elements are separated by commas. Lists and sets can occur at any valid noun-phrase position.

```
[1, 2, 3] is a list.
{} is empty.
4 is not an element of {1, 2, 3}.
[1, 2, 3, 4, 5] is the concatenation of [1, 2] and [3, 4, 5].
{2, 3} is included by {1, 2, 3, 4}.
```

Lists and sets can contain integers, reals, strings, variables, proper names, and expressions.

```
A value X is an element of {1, 2.3, "abc", X, John}.
[1 + 2, "a" & "bc"] is a list.
```

Lists and sets can also contain nested lists or sets.

```
{[1, 2, 3], {"a", "b", ["c", "d"]}} is a set.
```

Operators cannot be applied on lists and sets. Furthermore, lists and sets cannot be used in formulas. In the DRS, lists and sets are represented with the functions `list/1` and `set/1` respectively.

```
list([int(1), int(2), int(3)])
set([string(a), string(b), string(c)])
```

The "WG I2 Updated Workplan for the Months 37-48: Deliverables Concerning Controlled Natural Language" contains for the deliverable I2-D13 the item

> *Furthermore, we will extend ACE by data structures and operations on them and by procedural attachments.*

To prevent turning ACE into a visibly formal language we decided to express operations on the data structures list and set in natural language (see the above examples). The transformation of these operations into formal notations and the attachment of procedures to execute the operations are left to the tools that process the DRS generated from the input.

Alternatives, that use standard notations for functions and relations, for instance `sin(0.3)` and `member(X,[1,2,3])`, were considered, but for the time being rejected.

4. AceWiki

4.1. Introduction

AceWiki is a prototype demonstrating the use of controlled natural language (i.e. ACE) for semantic wikis [Tazzoli et al. 2004]. Being a semantic wiki, AceWiki combines the ideas and technologies of the Semantic Web with the concepts of wikis. The use of controlled natural language allows common users – who are not familiar with the concepts of logic and ontologies – to understand, modify, and extend the formal semantics of the wiki.

The focus of AceWiki is on usability which means that the architecture is designed on the basis of usability concerns. The usability is not just the top-most layer, but pervades the complete system. This can be illustrated by the fact that ACE is the main internal and external language.

The main goal of AceWiki is to broaden the field of potential users for semantic wikis. It is designed to have a shallow learning curve for both understanding and modification of the semantic content. At the same time, it uses a subset of ACE that is more expressive than most languages used by other semantic wikis. For example, the existing semantic wikis PlatypusWiki [Campanini et al. 2004], Semantic MediaWiki [Völkel et al. 2006] and WikSAR [Aumüller & Auer 2005] support only subject-predicate-object structures.

Naturalness and *strict user guidance* are two major design principles of AceWiki that have been implemented to a great extent. Both of them concern usability. By naturalness we mean that the formal semantics has a direct connection to natural language. Strict user guidance means that a predictive editor ensures that users create only well-formed statements. The next two sections will discuss these two principles and show how they are achieved in AceWiki.

4.2. Naturalness

AceWiki is natural in the sense that its content is represented in a form that is very close to natural language. First of all, ontological entities (like individuals, concepts, and roles) are represented as natural language words (like proper names, nouns, and verbs). This results in a one-to-one mapping of ontological entities to words in natural language. On this basis, ontological statements can be expressed as ACE sentences. Since every ACE sentence is a valid English sentence, any English speaker can immediately understand those ontological statements.

We believe that ontological terms like *property*, *range*, or *subclass* are unknown or unclear to most potential users of a semantic wiki. Such terms do not comply with our principle of naturalness. For that reason, AceWiki completely avoids such terms. On the other hand, linguistic terms like *noun*, *verb phrase*, or *singular* should be familiar to most users, since they are taught even in elementary schools. AceWiki uses such linguistic terms instead of ontological terms, if necessary. In many cases though, such special terms are superfluous altogether. E.g. instead of saying something like

'man' is a subclass of 'human'

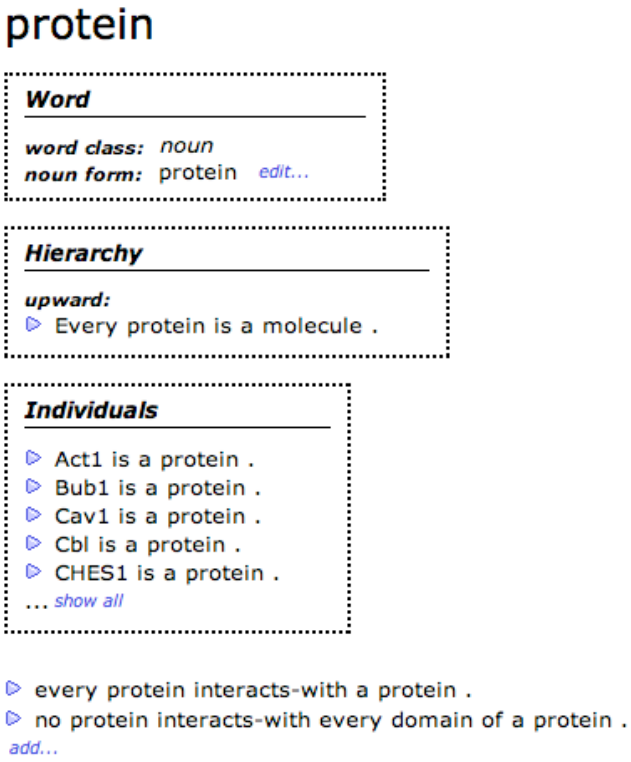
that uses the ontological term *subclass*, we can simply say

```
Every man is a human.
```

which does not use any special term.

A minor problem arises when using controlled natural language. Since informal (uncontrolled) natural language is still needed at some points (e.g. for introductory notes, help pages, labels, etc.), we have to make sure that the user does not confuse informal natural language with ACE. For example, an informal introductory note could be misinterpreted as a formal statement, or a formal statement could be misinterpreted as an informal explanation. In order to overcome this problem, we use a very simple convention: formal statements and terms are printed in normal font, whereas informal statements and terms in uncontrolled language are printed in *italics*. In this way, a user can immediately find out whether a certain statement or term is part of the formal ontology or not.

The picture below shows a screenshot of an example wiki about proteins. All formal representations appear in ACE, and all text that is not ACE is printed in italics.



4.3. Strict User Guidance

Learning a new formal language is normally accompanied by frequent syntax error messages from the parser. Wikis are supposed to enable easy and quick modifications of the content, and syntax errors can certainly be a major hindrance in this respect, especially for new users.

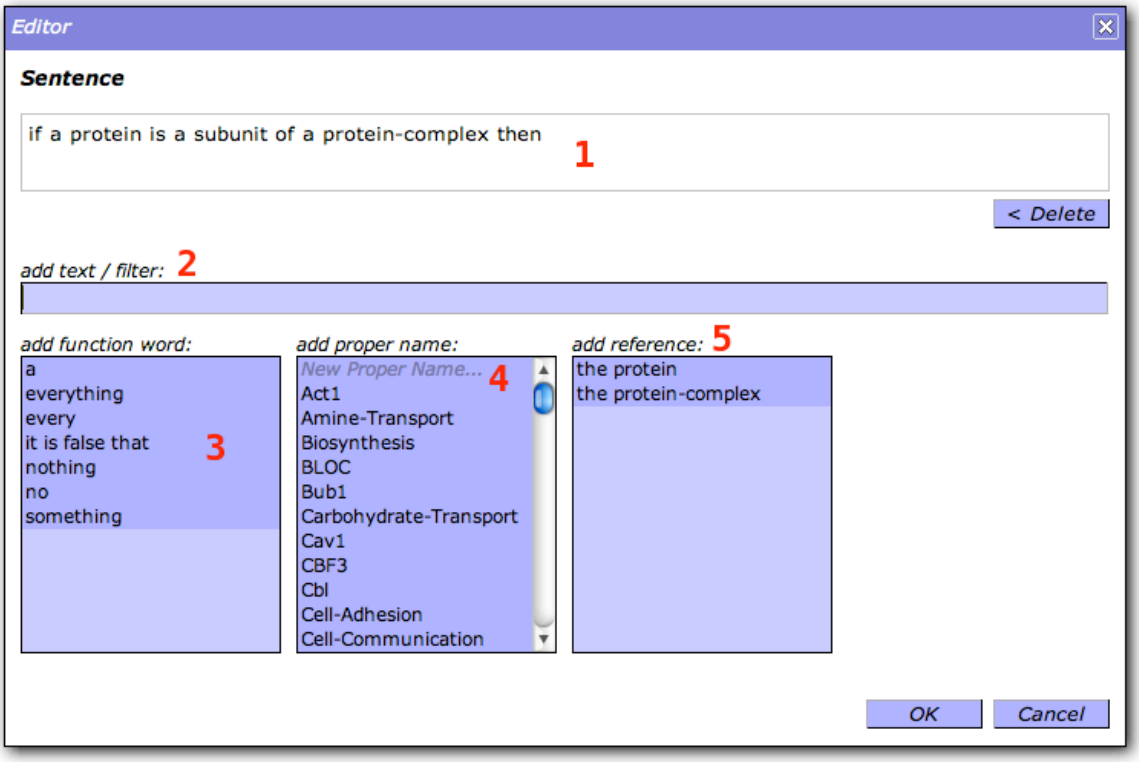
This problem can be solved by guiding the users during the creation of new statements in a strict manner. By strict we mean that the creation of syntactically incorrect sentences is simply made impossible in the first place. This can be achieved by a predictive editor that guides the user step by step and ensures the syntactic correctness.

Syntactic correctness can be subdivided into lexical and grammatical correctness. By lexical correctness we mean that only the words that are defined in a certain lexicon are used. Grammatical correctness on the other hand means that the grammar rules are respected.

To some degree, predictive editors can also take care of the semantic correctness. This is only possible if ontological information is available. If the verb "meets", for example, is defined in the ontology as a relation between humans then the predictive editor can prevent the user from writing sentences like "a man meets a car" (assuming that the ontology says that "car" is not human).

AceWiki has a predictive editor that is used for the creation or modification of ACE sentences. It ensures lexical, grammatical, and (to some degree) semantic correctness of the resulting sentences. In order to be convenient for both, novices and advanced users, the stepwise creation of a sentence can be done either by clicking on lists of proposed word (for novices) or by typing the words in a text field (for advanced users). Both alternatives are supported by a single graphical interface allowing the users to switch from one to the other at any time. The screenshot below shows the predictive editor of AceWiki.

The component with the number (1) is a read-only text field that shows the beginning of an ACE sentence. This beginning has been entered by a user and it has been accepted by the predictive editor as a correct sentence beginning. Thus, there is at least one possible completion that leads to a correct sentence. The button "Delete" can be used to undo the last step. The text field (2) can be used for entering the next words of the sentence. If they are a correct continuation of the sentence then they are moved to the text field (1). The tab key can be used to trigger auto-completion. The text of (2) is also used to filter the entries of the menus (3).



Clicking on the entries of the menu boxes (3) is an alternative way to construct a sentence. There is a menu box for each word class that is allowed at the current position. In this case, only function words, proper names, or references are allowed. The menu box for verbs, for example, is not shown because verbs are not allowed at this position. If a word is not yet known then it can be added on the fly by clicking on the respective menu entry (4). Then a dialog is shown that allows the user to add a new word. Also references can be introduced that point to objects occurring earlier in the sentence (5).

4.4. Syntax Boxes

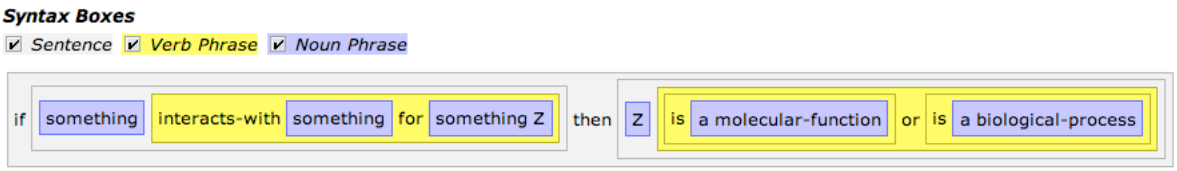
The structure of complex sentences (especially the scoping) is sometimes difficult to figure out. The syntax tree (that is generated by the ACE parser) can clarify the structure, but it is often hard to read as well. For that reason, AceWiki implements a special representation called “syntax boxes”.

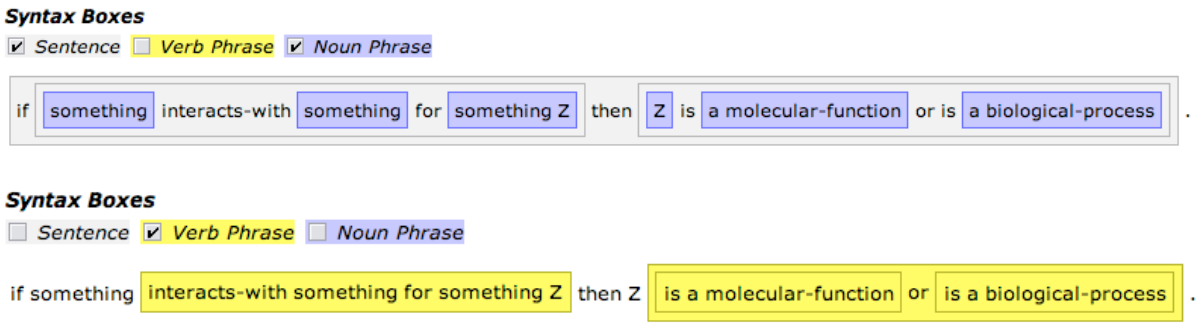
Syntax boxes are an alternative representation of the syntax tree, using nested boxes instead of a tree structure. There are three different kinds of boxes:

- grey boxes stand for sentences,
- yellow boxes stand for verb phrases, and
- blue boxes stand for noun phrases.

Boxes can contain other boxes. Each box directly or indirectly contains the words that belong to the phrase that the box represents. We believe that this representation using coloured boxes is more readable than syntax trees, at least for inexperienced users.

Furthermore, AceWiki allows the user to switch on/off each of the three types of boxes. The following pictures show the syntax boxes for one sentence using different configurations.





4.5. Conclusions and Future Work

Even though AceWiki is still work in progress, we think that it shows nicely the big impact that controlled natural language can achieve concerning usability.

In the near future, we plan to conduct user studies in order to substantiate the benefits of our approach. Furthermore, we plan to implement some kind of integrated reasoning for AceWiki.

5. ACE View Plug-In for the Protégé OWL Editor

5.1. Introduction

This section describes an integration of the ACE→OWL mapping [Kaljurand & Fuchs 2006, Kaljurand & Fuchs 2007] and the OWL→ACE mapping [Kaljurand & Fuchs 2007] into the widely used Protégé OWL editor [Horridge et al. 2004]. This integration is realized as a Protégé plug-in called *ACE View*. Specifically, we use (the alpha version of) Protégé 4[3].

Using Protégé 4 and its underlying OWL API [Horridge et al. 2007] as a platform, gives us access to the OWL reasoners Pellet [Sirin et al. 2007] and FaCT++ [Tsarkov & Horrocks 2006] which can be used to check the consistency of the ontology, entail new axioms on the basis of asserted axioms, and answer DL-Queries. The OWL API also supports the explanation of entailments via a Black Box OWL Debugger [Horridge et al. 2007]. Along with OWL axioms, also SWRL rules can be stored and manipulated using the OWL API.

We have extended the ACE→OWL mapping to support some forms of SWRL rules (without built-ins). Furthermore, some forms of ACE queries are mapped to DL-Queries. Those extensions are not described here. Instead, we concentrate on the ways that natural language based ontology editing can improve the usability of current OWL editors.

5.2. Protégé and ACE View

The standard *Protégé view* to an OWL ontology involves tabs for classes, properties and individuals. Each of those tabs contains several sub windows, e.g. a display of the tree hierarchy of *SubClassOf*-relationships between named classes, a listing of individuals, and lists of complex class descriptions rendered in Manchester Syntax [Horridge et al. 2006]. Protégé also provides several of the so-called *Ontology views*, most of which show various OWL representations of the ontology (RDF/XML, OWL 1.1 XML, etc), or general metrics of the ontology (DL expressivity, counts of various OWL constructs).

The *ACE View* developed by us provides an alternative *Ontology view* – a natural language rendering of the complete logical content of the ontology where for the natural language we use Attempto Controlled English (ACE). In this rendering, ACE sentences correspond to OWL axioms, and all metrics are linguistic, e.g. number of sentences and content words in the ACE text. The ACE view can be edited – sentences can be modified and deleted, and new sentences can be added. A single *Synchronize* button is currently provided to let the user trigger the synchronization of the edited ACE representation with the underlying Protégé representation of the ontology. (In the future, we will try to make the synchronization fully automatic.) A *Preferences* button allows the user to configure the webservices that provide the ACE→OWL and OWL→ACE translators. In addition to those two buttons, six tabs present different views to the ACE text, and thus to the whole ontology.

5.3. Tabs

ACE View provides 6 tabs that show the ACE text (and thus the ontology) via different angles.

- The *Main tab* provides a plain text ACE-representation of the complete ontology, allowing the user to modify the text via standard editing commands such as copy and paste.
- The *Index tab* provides a more structured representation of the text, using HTML for rendering and navigation.
- The *Paraphrase tab* provides a paraphrase of the ACE text.
- The *Inferences tab* shows the ACE representation of the axioms that the ontology entails together with their explanation.
- The *Answers tab* lets the user query the knowledge base using ACE questions. The answers are given as lists of ACE words or sentences using those words.

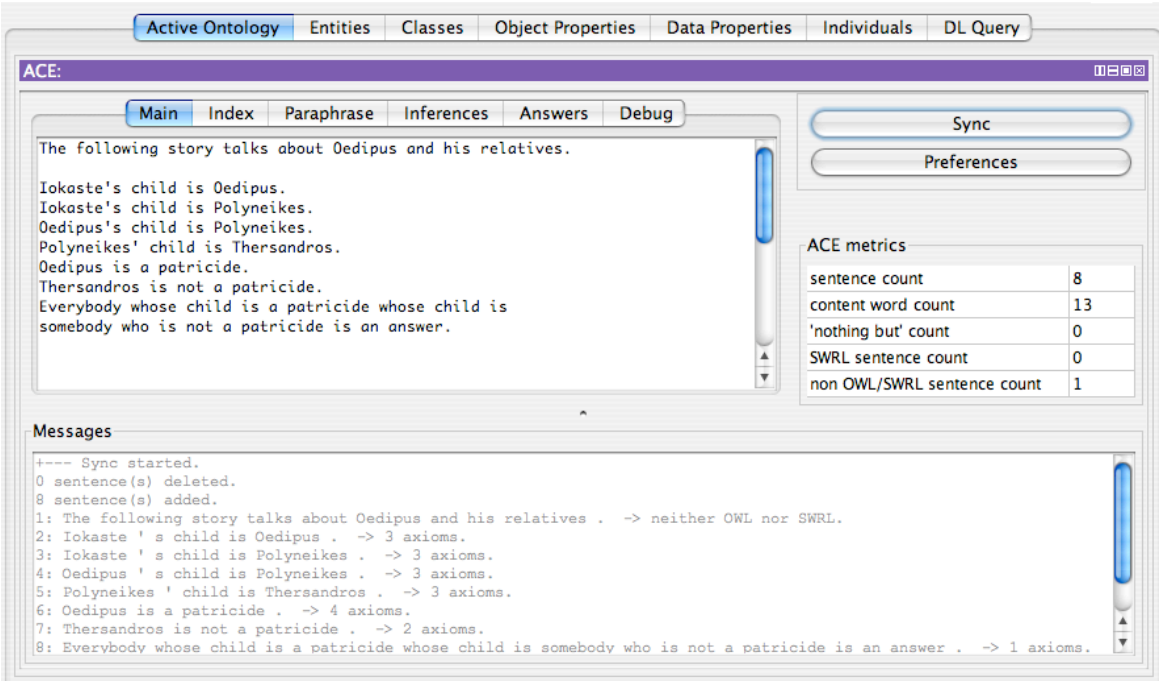
[3] http://www.co-ode.org/downloads/protege-x/

- The *Debug tab* gives a list of all entered ACE sentences along with some technical details about the parsing results for those sentences. In case of parsing failure, error messages are reported that help the user to rephrase the sentence in an ACE-compatible way.

The following sections describe the tabs in more detail and show screenshots of our current implementation. Note that not all aspects of these tabs are fully implemented at the moment. Therefore, we expect some changes to occur in the design and function of these tabs.

5.3.1. Main

In the *Main tab*, the current ACE text is displayed and can be edited. Pressing the *Sync* button, triggers the updates to the text to be parsed and integrated into the ontology. Although the user is expected to enter sentences which can be mapped to OWL, inputting sentences that are not ACE, or that map only to SWRL, is tolerated. Such sentences, however, do not participate in reasoning. The *Debug tab* provides explanations of why a certain sentence could not be parsed. Such a sentence can be modified at any time to comply with ACE, or it can be left around as a "comment".

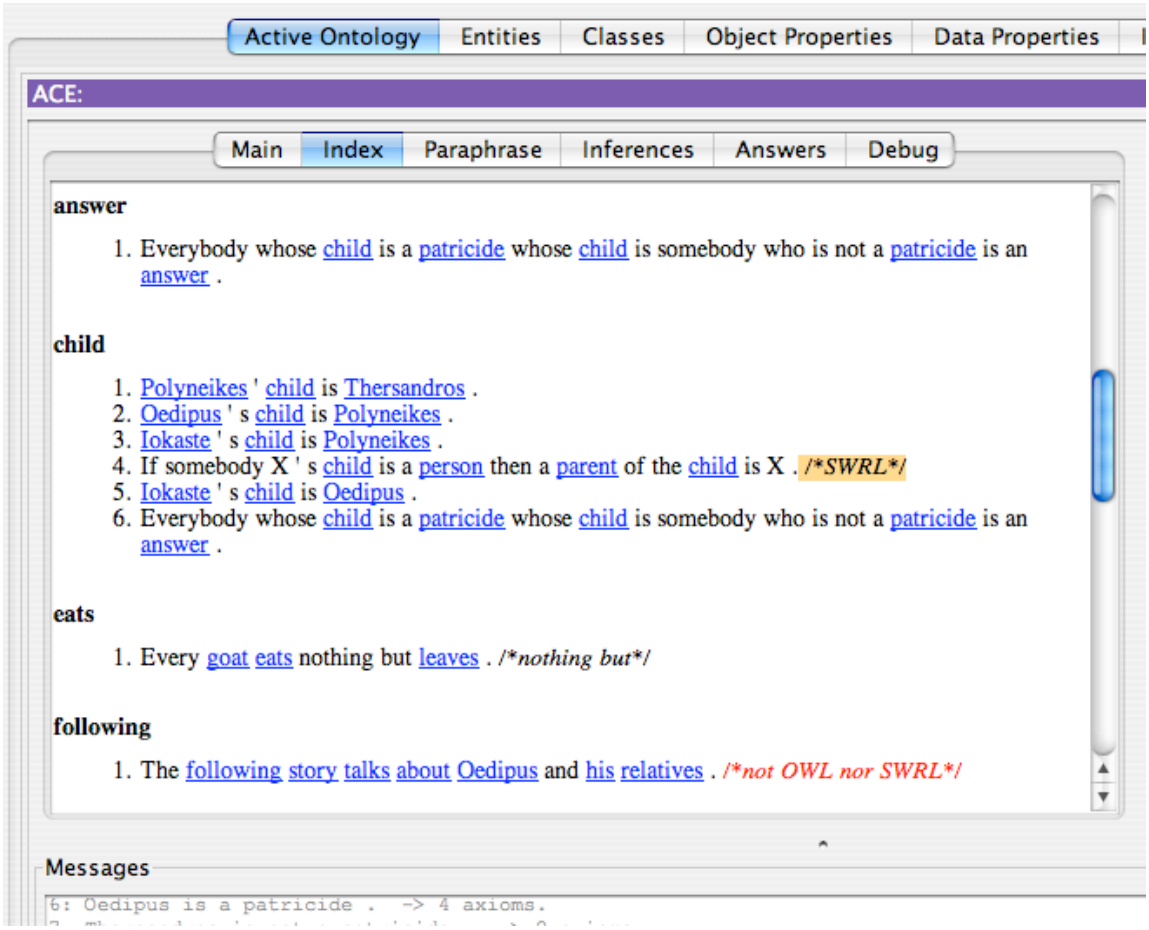


Screenshot of the *Main tab* of the ACE View plug-in for Protégé 4. The complete ontology is displayed in one text area.

5.3.2. Index

In the *Index tab*, the complete ACE text is presented as an index – the set of content words is alphabetically sorted and every content word is listed together with all the sentences that contain the word. Every content word in a sentence is furthermore a hyper-link to the entry of the content word, thus allowing for easy navigation in the index. (Note that there is some similarity between the index view and the *Usage views* of Protégé 4).

Every sentence that was not successfully parsed into OWL or SWRL is marked by a red label /*not OWL nor SWRL*/. Every sentence that was not succesfully parsed into OWL but that could be parsed into SWRL is labelled as /*SWRL*/. The index view does not currently allow for editing. This is future work.



Screenshot of the *Index tab* of the ACE View plug-in for Protégé 4. The complete ACE text is indexed and rendered in HTML.

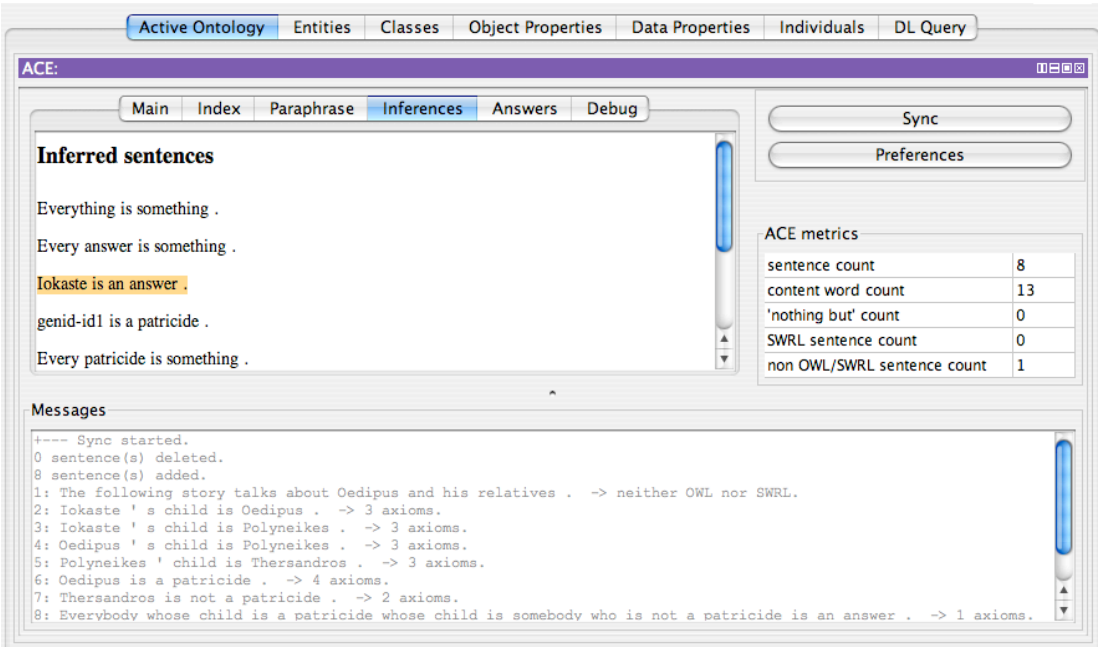
5.3.3. Paraphrase

In the *Paraphrase tab*, a paraphrase of the ACE text is provided. A paraphrase is one way for users to check if their interpretation of the inserted text is accurate. ACE provides many forms of syntactic sugar, thus allowing for paraphrasing, e.g. *every*-sentences can be rephrased via *if-then* sentences, and in many cases vice-versa. At the moment, the paraphrase is a verbalization of the whole ontology via the OWL→ACE mapping. In the future, we will allow the user to select other forms of paraphrasing, e.g. the ones based on Core ACE [Fuchs et al. 2005] or NP ACE [Fuchs et al. 2006].

5.3.4. Inferences

The *Inferences tab* provides a list of ACE sentences that correspond to the entailed axioms of the ontology. Such axioms can be automatically generated by the built-in reasoner. These axioms have a very simple structure, i.e. they are class assertions, property assertions and sub class axioms where the involved individuals, properties, and classes are always named. Thus their natural language verbalization cannot potentially bring significant usability improvement. Nevertheless, the presentation of all entailments as a single list of natural language sentences can provide a good and easily readable overview. Alternatively, an index view to the entailments could be provided.

Protégé also supports entailment explanations. Such an explanation is a sequence of axioms (usually previously asserted, but possibly synthesized) that motivates the entailment. The axioms in this sequence can be of any complexity and thus their natural language verbalization can bring significant improvement in understanding the reason behind the entailment. At the time of writing we have not yet implemented the explanation support.



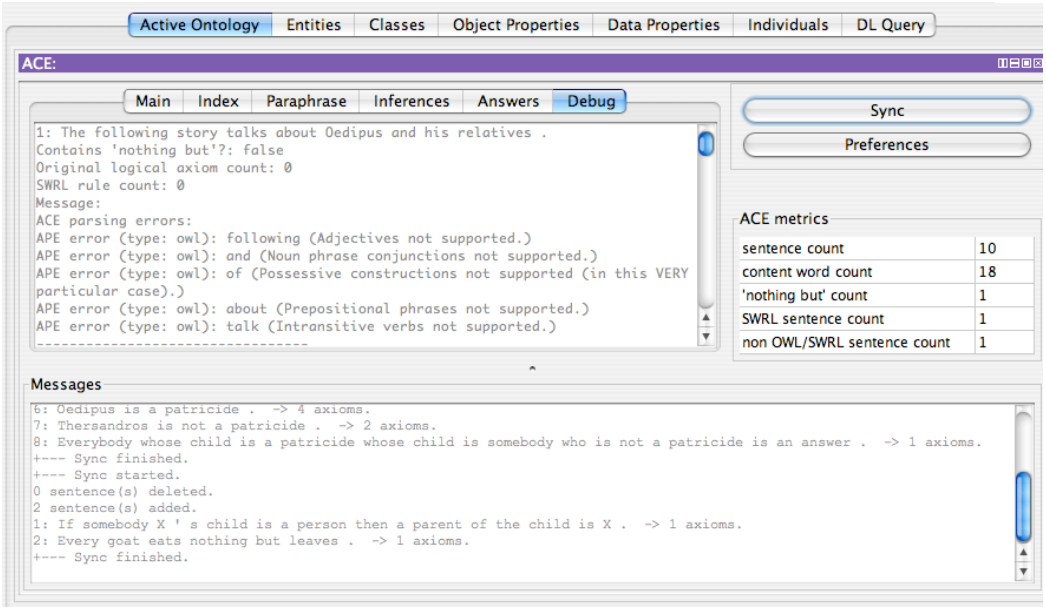
The *Inferences tab* shows a list of inferred axioms as ACE sentences. E.g. the sentence `Iokaste is an answer.` was not explicitly present in the original text. Still, via non-trivial description logic reasoning this sentence can be derived.

5.3.5. Answers

The *Answers tab* – not yet implemented – will allow ACE questions to be entered and answered using the Protégé implementation of DL-Query. A DL-Query is essentially a (complex) class description. Answers to a DL-Query are named individuals (members of the queried class) or named classes (named super and sub classes of the queried class). In ACE, answers are ACE content words – proper names and common nouns. While answers to DL-Queries are representation-wise identical in the ACE view and in the standard Protégé view, the construction of queries is potentially much simpler in the ACE view, as one has to construct a natural language question.

5.3.6. Debug

A *Debug tab* is provided to help users to get an overview of the logical and linguistic properties of the entered sentences. For sentences that fail to map to OWL/SWRL, error messages are provided.



The *Debug tab* lists all entered sentences along with their logical and linguistic properties. The sentence `The following story is about Oedipus and his relatives.` was not mapped to OWL/SWRL and does not participate in the entailments.

5.4. Conclusion

The ACE View plug-in provides a radical simplification of the currently standard ontology editing environment – the complete ontology is displayed in a single text-area, the editing is to be performed as with regular text, supported by well-known operations like copy/paste/cut.

The combination of natural language based ontology editing and the standard form-based editing (which is also used in tools like SWOOP[4] and TopBraid Composer[5]) offers more alternatives for the user and can result in a better usability especially in the case of novice ontology engineers and domain experts.

[4] http://code.google.com/p/swoop/

[5] http://www.topbraidcomposer.com/

6. Reasoning in ACE

6.1. RACE in a Nutshell

The Attempto Reasoner RACE [Fuchs & Schwertel 2003] supports automatic reasoning in ACE. Currently, 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. Variations of the basic proof procedure permit query answering and consistency checking.

RACE is supported by auxiliary axioms expressed in the language of first-order logic or in Prolog. Auxiliary axioms implement domain-independent linguistic and mathematical knowledge that cannot be expressed in ACE since it depends on the DRS representations of ACE texts. Examples are the relation between plurals and singulars and a theory of natural numbers. Auxiliary axioms can also act as meaning postulates for ACE constructs that are under-represented in the DRS, for example generalised quantifiers. Finally, auxiliary axioms can be used to represent domain-specific knowledge that could in principle be expressed in ACE.

The current implementation of RACE is based on the model generator Satchmo [Manthey & Bry 1988]. Satchmo is implemented in Prolog which allows us to add modifications and extensions. Currently, we employ Satchmo only for theorem proving. Improved query answering will utilise Satchmo also as model generator.

Satchmo works with clauses. ACE axioms A and ACE theorems T are translated – via DRSs generated by APE – into their first-order representations FA, respectively FT. The auxiliary first-order axioms are conjoined to the formula FOL. Then the conjunction (FA $\wedge$ FOL $\wedge \neg$ FT) is translated into clauses, submitted to Satchmo and checked for consistency. Satchmo 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, Satchmo will generate a minimal finite model – if there is one.

6.2. Adaptations to ACE 6

The original auxiliary axioms were developed for the DRS language of ACE 4. Since then we changed and extended the DRS language several times. Some changes were rather radical. As a consequence, the auxiliary axioms could no longer be adapted to the new DRS language, but had to be completely redesigned and rewritten. This work is still going on. Specifically, an effective and efficient handling of the modality introduced in ACE 5 is still missing. Here we have the alternative of the standard translation of modal logic into possible-worlds first-order logic, or ad hoc defined auxiliary axioms.

6.3. Controlling Deductions

RACE uses several methods to control, i.e. to enable or disable, deductions. In the following we briefly present some of those methods.

Put simply, a theorem can be proved if the conjunction of its logical atoms can be unified with the model derived from the axioms. Thus we can enable or disable deductions by suitable DRS representations. However, this can conflict with our intention to under-represent critical ACE constructs, for instance generalised quantifiers.

One can enable otherwise impossible deductions by an auxiliary FOL axiom that generates the logical atoms missing in the model of the axioms. To derive

```
A rich man waits.
```

with the DRS

```
[A, B]
object(A, man, countable, na, eq, 1)
property(A, rich, pos)
predicate(B, wait, A)
```

from

```
A man is rich and waits.
```

with the DRS

```
[A, B, C, D]
object(A, man, countable, na, eq, 1)
property(B, rich, pos)
predicate(C, be, A, B)
predicate(D, wait, A)
```

we label the copula `be` of the second DRS as `be_ADJ`, and then use the auxiliary FOL axiom

```
forall([I, A1, A2, B1, EQ1, N1, T1, Adjective, Degree],
predicate(I, be_ADJ, A1, A2) & object(A1, B1, T1, na, EQ1, N1) &
property(A2, Adjective, Degree)
=>
property(A1, Adjective, Degree))
```

that provides the missing logical atom `property(A,rich,pos)`.

On the other hand, sometimes one wants to block a deduction that would otherwise be possible. For instance we do not want to derive the general statement

```
A man is busy.
```

from the restrictive statement

```
A man is busy in the morning.
```

There is a very simple and effective solution to this problem. During the generation of the Satchmo clauses we differently label the copulas `be` of the axiom and of the theorem, thus preventing their unification.

Satchmo processes clauses by forward reasoning. There are, however, cases, where backward reasoning is required. To derive

```
Four men wait.
```

from

```
Five men wait.
```

we have to generate the number `4` from the number `5` which cannot be done in general by forward reasoning with first-order axioms. Instead we apply backward reasoning with the Prolog axiom

```
object(A, B, C, na, eq, N) :-
  number(N),
  asserted_atom(object(A, B, C, na, eq, M), _Indices),
  number(M),
  N=<M.
```

First-order axioms can be used to enable and – with some care to avoid unwanted inconsistencies with the ACE axioms – to disable deductions. Prolog axioms can both enable and – thanks to negation-as-failure – disable deductions.

6.4. Increasing Efficiency

The run-time of RACE depends foremost on the number of clauses that are used for forward reasoning. To reduce the run-time we have investigated four approaches

- trying to keep the number of clauses small: by simplifying the DRS representation, by clause compaction, by keeping the number of first-order axioms small, by replacing first-order axioms by Prolog axioms
- eliminating after the first round of forward reasoning those clauses that cannot be called again, concretely the *fact clauses* with the body `true`
- applying an intelligent search for clauses that could be "fired" in the next round of forward reasoning, concretely those clauses that contain in their body an atom that was added to the working memory in the round before; we will alternatively investigate the use of the Rete algorithm
- using 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

Altogether these means turned out to be very effective.

6.5. Alternative Deductions

The "WG I2 Updated Workplan for the Months 37-48: Deliverables Concerning Controlled Natural Language" contains for the deliverable I2-D13 the item

> *We will extend the ACE reasoner RACE to give explanations to "why/why not" queries and to investigate hypothetical "what if" queries.*

While the current version of RACE effectively answers "why" questions, we postponed "why not" questions to the next deliverable I2-D15 since they amount to abduction – one of the topics of I2-D15.

Hypothetical "what if" questions can easily be implemented with the current version of RACE by simply feeding RACE in addition to the axioms and theorems a further set of "hypothetical" axioms.

7. Updates

7.1. AceRules

The AceRules web service has been updated. It is now fully compliant with SOAP 1.1 [SOAP] and WSDL [WSDL]. [AceRules-Webservice] describes the details of the web service. As a new feature, it allows now the users to load their own lexica.

7.2. DRACE

The DRS verbalizer DRACE (which we currently use to paraphrase ACE texts) was updated to support the new ACE features:

- arithmetic expressions
- more expressive adjectives
- limited forms of sentence subordination

Also, the morphological synthesis of surface wordforms from lemmas (e.g. `man→men`, `like→liked`) is now based on the APE lexicon.

8. Cooperations and Visits

We are pleased to report that we are cooperating with several REWERSE internal and external groups who use Attempto Controlled English. Here is a summary:

- Piero Bonatti, Juri Luca De Coi and Luigi Sauro of REWERSE I2 visited the Attempto group from 30 July to 3 August 2007 to work on ACE as input language for the Protune system; code developed for AceRules could be reused for this purpose.
- We are discussing a cooperation with REWERSE A2 concerning the use of ACE as query language for GoPubMed.
- Nelly Schuster and Olaf Zimmermann of IBM Research (Rüschlikon, Switzerland) visited the Attempto group on 10 August 2007 to discuss the application of ACE to software architecture; further contacts are planned.
- Catherine Dolbear and Glen Hart of Ordnance Survey Research (Southampton, UK) visited the Attempto group on 4-5 October 2007 to discuss a controlled English syntax of OWL 1.1.
- Bettina Bauer-Messmer, Rolf Gruetter, and Martin Haegeli of ETH WSL (Birmensdorf, Switzerland) visited the Attempto group on 5 October 2007 to discuss the potential of ACE for their work on geographical data processing.

9. Deliverable I2-D15

In deliverable I2-D15 we will demonstrate the consistent and non-redundant assimilation of ACE sentences to existing ACE texts, and apply knowledge assimilation to the AceWiki system. The ACE reasoner RACE will be extended by abduction to answer queries of the form “under which conditions does . . . occur”, and by "why not " questions originally scheduled for deliverable I2-D13. Since this will be the last REWERSE deliverable, we will also try to collect and to evaluate user feedback on the usability and acceptability of ACE.

10. References

[AceRules-Webservice] AceRules Webservice. Attempto Documentation, 20 August 2007, http://attempto.ifi.uzh.ch/site/docs/acerules_webservice.html

[Aumüller & Auer 2005] D. Aumüller and Sören Auer. Towards a Semantic Wiki Experience – Desktop Integration and Interactivity in WikSAR. In Proceedings of 1st Workshop on The Semantic Desktop, Galway, Ireland, 2005.

[Campanini et al. 2004] Stefano Emilio Campanini, Paolo Castagna, Roberto Tazzoli. Platypus Wiki: a Semantic Wiki Wiki Web. In Semantic Web Applications and Perspectives, Proceedings of 1st Italian Semantic Web Workshop, Dec 2004.

[Fuchs & Schwertel 2003] N. E. Fuchs, U. Schwertel. Reasoning in Attempto Controlled English, in: F. Bry, N. Henze and J. Maluszynski (eds.): *Principles and Practice of Semantic Web Reasoning*, International Workshop PPSWR 2003, Mumbai, India, December 2003. Lecture Notes in Computer Science 2901, Springer Verlag, 2003.

[Fuchs et al. 2005] Norbert E. Fuchs, Kaarel Kaljurand, and Gerold Schneider. Deliverable I2-D5. Verbalising Formal Languages in Attempto Controlled English I. Technical report, REWERSE, 2005. http://rewerse.net/deliverables.html.

[Fuchs et al. 2006] Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. Deliverable I2-D9. Attempto Controlled English 5: Language Extensions and Tools I. Technical report, REWERSE, 2006. http://rewerse.net/deliverables.html.

[Horridge et al. 2007] Matthew Horridge, Sean Bechhofer, Olaf Noppens. Igniting the OWL 1.1 Touch Paper: The OWL API. In 3rd OWL Experiences and Directions Workshop (OWLED 2007), 2007.

[Horridge et al. 2006] Matthew Horridge, Nick Drummond, John Goodwin, Alan Rector, Robert Stevens, and Hai H Wang. The Manchester OWL Syntax. In 2nd OWL Experiences and Directions Workshop (OWLED 2006), 2006.

[Horridge et al. 2004] Matthew Horridge, Holger Knublauch, Alan Rector, Robert Stevens, and Chris Wroe. A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools. Edition 1.0. Technical report, The University Of Manchester, 2004. http://www.co-ode.org/resources/tutorials/.

[Kaljurand 2007] Kaarel Kaljurand. Writing OWL ontologies in ACE. Technical report, Attempto project, 2007. http://attempto.ifi.uzh.ch/site/docs/writing_owl_ in_ace.html.

[Kaljurand & Fuchs 2006] Kaarel Kaljurand, Norbert E. Fuchs. Bidirectional mapping between OWL DL and Attempto Controlled English. In Fourth Workshop on Principles and Practice of Semantic Web Reasoning, Budva, Montenegro, 2006.

[Kaljurand & Fuchs 2007] Kaarel Kaljurand, Norbert Fuchs. Verbalizing OWL in Attempto Controlled English. In 3rd OWL Experiences and Directions Workshop (OWLED 2007), 2007.

[Manthey & Bry 1988] Rainer Manthey, François Bry. SATCHMO: A Theorem Prover Implemented in Prolog. Proc. 9th International Conference on Automated Deduction. LNCS 310, pp. 415-434

[Sirin et al. 2007] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A Practical OWL-DL Reasoner. Web Semantics: Science, Services and Agents on the World Wide Web, 5(2):51–53, 2007.

[SOAP] Simple Object Access Protocol (SOAP) 1.1. W3C Note, 8 May 2000, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[Tazzoli et al. 2004] Roberto Tazzoli, Paolo Castagna, Stefano Emilio Campanini. Towards a Semantic Wiki Wiki Web. 3rd International Semantic Web Conference (ISWC2004), 7-11 November 2004, Hiroshima, Japan.

[Tsarkov & Horrocks 2006] Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In International Joint Conference on Automated Reasoning (IJCAR 2006), volume 4130 of Lecture Notes in Artificial Intelligence, pages 292–297. Springer, 2006.

[Völkel et al. 2006] Max Völkel, Markus Krötzsch, Denny Vrandecic, Heiko Haller, Rudi Studer. Semantic Wikipedia. Proceedings of the 15th international conference on World Wide Web, WWW2006, Edinburgh, Scotland, May 23-26, 2006

[WSDL] Web Services Description Language (WSDL) 1.1. W3C Note, 15 March 2001, http://www.w3.org/TR/2001/NOTE-wsdl-20010315