I2-D3

A Parser for Attempto Controlled English

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/I2D3/D/PU
Responsible editor:	Norbert E. Fuchs
Reviewers:	Piero Bonatti, Uta Schwertel
Contributing participants:	University of Zurich
Contributing workpackages:	I2
Contractual date of delivery:	March 31, 2005
Actual date of delivery:	April 10, 2005

Abstract

The Attempto Parsing Engine (APE) translates texts in Attempto Controlled English (ACE) into discourse representation structures, a variant of first-order logic. This report presents version 4 of ACE in a nutshell, gives an overview of discourse representation structures, describes APE, and shows how users can work with APE.

Keyword List

Attempto Controlled English, ACE, controlled natural language, discourse representation structure, DRS, DRT, parser, Attempto Parsing Engine, APE, web-interface, remote procedure call, web service

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

A Parser for Attempto Controlled English

Norbert E. Fuchs, Kaarel Kaljurand, Fabio Rinaldi, Gerold Schneider

Department of Informatics

&

Institute of Computational Linguistics

University of Zurich

Email: {fuchs, kalju, rinaldi, gschneid}@ifi.unizh.ch

April 5, 2005

Abstract

The Attempto Parsing Engine (APE) translates texts in Attempto Controlled English (ACE) into discourse representation structures, a variant of first-order logic. This report presents version 4 of ACE in a nutshell, gives an overview of discourse representation structures, describes APE, and shows how users can work with APE.

Keyword List

Attempto Controlled English, ACE, controlled natural language, discourse representation structure, DRS, DRT, parser, Attempto Parsing Engine, APE, web-interface, remote procedure call, web service

Contents

1. Introduction

> *[…] a truly semantic web is more likely to be based on natural language processing than on annotations in any artificial language.*
>
> *If I had to process web annotations in any artificial language, I would prefer to use controlled English rather than special notations such as RDF. There is no reason why web annotations have to be humanly unreadable in order to be easy to process by a computer.*
>
> John F. Sowa, CG Mailing List, October 19, 2003

Attempto Controlled English (ACE) is a controlled natural language, i.e. a precisely defined subset of full English that can automatically and unambiguously be translated into full first-order logic. Thus ACE is both human and machine understandable.

ACE seems completely natural, but is in fact a formal language. One could say that ACE is a first-order logic language with an English syntax.

While the meaning of a sentence in natural language can vary depending on world-knowledge and on a – possibly only vaguely defined – context, the meaning of an ACE sentence is completely and uniquely defined in the context of the preceding sentences. Thus writers and readers of an ACE text will understand the text in the same way.

ACE combines natural language with formal methods, for instance deduction, and is arguably easier to learn and use than evidently formal languages like RDL or OWL. ACE texts can be translated into any formal language equivalent to (a subset of) first-order logic, and vice versa[*], and thus can either replace or complement these languages.

The attributes of ACE make it a prime candidate for the knowledge representation and query tasks of the semantic web. Since ACE is specifically suited to represent business and policy rules, REWERSE has decided to base its controlled English on an appropriately extended version of ACE.

[*]The translation of first-order logic into ACE will be addressed in future REWERSE deliverables.

2. Attempto Controlled English

The Attempto system, specifically Attempto Controlled English (ACE), is intended for domain specialists – e.g. engineers, economists, physicians – who want to use formal notations and formal methods, but may not be familiar with them. Thus the Attempto system has been designed in a way that allows users to work solely on the level of ACE without having to take recourse to formal notations.

ACE is a subset of standard English that allows users to express technical texts precisely, and in the terms of the respective application domain. ACE texts are computer processable and can be unambiguously translated into discourse representation structures (DRS), a syntactic variant of first-order predicate logic. Though ACE appears completely natural, it is in fact a formal language with the semantics of the underlying logic representation. One could say that ACE is a first-order logic language with the syntax of a subset of English.

ACE is based on Discourse Representation Theory (DRT). The central concern of DRT is to assign meaning to complete texts or discourses, and to account for the context dependence of meaning. While in general the context of a natural language text is only vaguely defined and can vary, the context of an ACE text is completely fixed. Concretely, an ACE text consists of a sequence of interrelated sentences where each sentence can anaphorically refer to noun phrases occurring in previous sentences. Thus, each sentence is interpreted in the context of the preceding sentences. No further context exists.

The Attempto system is not associated with any specific application domain, or with any particular formal method. By itself it does not contain any knowledge of application domains, of formal methods, or of the world in general. Thus the only source of information is the ACE text itself.

Users must explicitly define domain knowledge through ACE sentences like

> A licence is valid.

In this sentence the word *licence* and *valid* are processed by the Attempto system as uninterpreted syntactic elements, i.e. any interpretation of these words is solely performed by the human writer or reader. The meaning of words can be constrained when we add further information, e.g.

> Every licence that is not expired is valid.

As a consequence of its syntactic approach, the Attempto system can only detect explicit logical contradictions. For example, if the text states in one place that a licence is *valid*, and in another place that the same licence is *not valid*, the contradiction can be detected. However, if in one place the text declares a licence as *valid*, and in another place the same licence as *invalid*, the contradiction can only be detected if we explicitly define that *valid* and *invalid* exclude each other, e.g.

> Every licence that is not valid is invalid. Every licence that is invalid is not valid.

Domain-specific ACE definitions, axioms, or ontologies could be predefined, placed in libraries, and imported when required.

2.1. Attempto Controlled English in a Nutshell

The following is intended as a brief introduction into ACE 4. A full account of the language can be found in the report 'Attempto Controlled English (ACE) Language Manual, Version 4.0' [www.ifi.unizh.ch/attempto].

ACE 4 is a major revision and extension of ACE 3 that we presented in previous publications. Working with ACE 3 lead to many – usually small – changes of the language. Most importantly ACE 4 – other than ACE 3 – admits plurals.

Vocabulary

The vocabulary of ACE comprises

- predefined function words (e.g. determiners, conjunctions, prepositions)
- user-defined, domain-specific content words (nouns, verbs, adjectives, adverbs)

The Attempto system provides a basic lexicon of content words. Users can define additional content words with the help of a lexical editor, or can import existing lexica.

Grammar

The grammar of ACE defines and constrains the form and the meaning of ACE texts. ACE's grammar is expressed as a small set of construction rules.

ACE Texts

An ACE text is a sequence of anaphorically interrelated sentences. There are

- simple sentences
- composite sentences

Furthermore, there are query sentences that allow users to interrogate the contents of an ACE text.

Simple Sentences

A simple sentence describes a situation that can be an event or a state.

> A customer inserts 2 cards.
>
> A card is valid.

Simple ACE sentences have the following general structure:

> subject + verb + complements + adjuncts

Complements (direct and indirect objects) are necessary for transitive verbs (*insert something*) and ditransitive verbs (*give something to somebody*), whereas adjuncts (adverbs, prepositional phrases) are optional.

All elements of a simple sentence can be elaborated upon to describe the situation in more detail.

To further specify the nouns *customer* and *card*, we could add adjectives

> A new customer inserts 2 valid cards.

possessive nouns and of-prepositional phrases

> John's customer inserts a card of Mary.

or proper nouns and variables as appositions

> The customer Mr Miller inserts a card A.

Other modifications of nouns are possible through relative sentences

> A customer who is new inserts a card that he owns.

which are described below since they make a sentence composite.

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

> A customer inserts some cards manually.

or equivalently

> A customer manually inserts not more than 2 cards.

or by adding prepositional phrases, e.g.

A customer inserts a card into a slot.

We can combine these elaborations to arrive, for instance, at

John's customer who is new inserts a valid card of Mary manually into a slot A.

Composite Sentences

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 2 cards and the machine checks their codes.

A customer inserts a card and enters a code.

A customer enters a card in the morning and in the evening.

An old and trusted customer enters a card and a code.

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

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

A customer inserts a card or enters a code.

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

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

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

There are two forms of subordination: relative sentences and if-then sentences.

Relative sentences starting with *who*, *which*, *that* allow to add detail to nouns, e.g.

A customer who is new inserts a card that he owns.

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

If a card is valid then a customer inserts it.

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

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

To express that all involved customers insert cards we can write

Every customer inserts a card.

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

There is a card that every customer inserts.

*Exception: Noun phrase coordination in prepositional phrases, e.g. *in the morning and the evening* is not possible. One uses instead a coordination of prepositional phrases, i.e. *in the morning and in the evening*.

ACE does not know the passive voice. To state that every card is inserted by a customer we write somewhat stilted

> For every card there is a customer who inserts it.

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

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

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

> No customer inserts more than 2 cards.

or, equivalently, *there is no*

> There is no customer who inserts a card.

To negate a complete statement one uses sentence negation

> It is not the case that a customer inserts a card.

Query Sentences

Query sentences permit us to interrogate the contents of an ACE text. There are yes/no-queries and as wh-queries.

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

> A customer inserts a card.

then we can ask

> Does a customer insert a card?

to get a positive answer.

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

> A new customer inserts a valid card manually.

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

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

Note, however, that we cannot ask for the verb itself.

2.2. Constraining Ambiguity

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

- some ambiguous constructs are not part of the language; unambiguous alternatives are available in their place
- all remaining ambiguous constructs are interpreted deterministically on the basis of a small number of interpretation rules
- users can either accept the assigned interpretation, or they must rephrase the input to obtain another one

Avoidance of Ambiguity

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

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

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

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

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

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

with the interpretation

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

Interpretation rules

However, not all ambiguities can be safely removed from ACE without rendering it evidently artificial. To deterministically interpret otherwise syntactically correct ACE sentences ACE uses less than 20 interpretation rules. Here are some examples. If we write

> The customer inserts a card with a code.

we get the interpretation

> The customer {inserts a card with a code}.

that reflects ACE's interpretation rule that a prepositional phrase always modifies the verb. However, this is probably not what we meant to say here. To express that the code is associated with the card we can employ the interpretation rule that a relative sentence always modifies the immediately preceding noun phrase.

> The customer inserts a card that carries a code.

yielding the interpretation

> The customer inserts {a card that carries a code}.

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

> The customer inserts a card and a code.

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

> The customer who inserts a card manually enters a code.

we employ the interpretation rule that the postverbal position has priority.

> The customer who {inserts a card manually} enters a code.

2.3. Anaphoric References

Usually an ACE text consists of more than one sentence.

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

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

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

During the processing of the ACE text every anaphoric reference is replaced by the most recent and most specific accessible noun phrase that agrees in gender and number, yielding

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

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.

yields

> [The blue card] is correct.

while

> The red card is correct.

yields

> [The red card] is correct.

What does "accessible" mean? According to Discourse Representation Theory noun phrases introduced in if-then sentences, universally quantified sentences or negations are not accessible as antecedents of anaphora. Thus *the card* in

> A customer does not enter a card. The card is correct.

cannot refer to *a card*.

Anaphoric references are also possible via personal pronouns

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

or via variables

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

Anaphoric references via definite articles and variables can be combined.

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

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

3. Discourse Representation Structures

Attempto Controlled English is based on Discourse Representation Theory (DRT) [Kamp & Reyle 1993], a powerful linguistic theory whose central concern is the meaning of texts and discourses. The representation language of DRT are discourse representation structures (DRS), a variant of the language of first-order logic.

For ACE version 4 we developed an extended form of discourse representation structures that

- uses only a small number of predefined predicates
- represents information derived from words as arguments of the predefined predicates
- has eventuality types
- uses a lattice-theoretic representation of objects that allows us to encode plurals in first-order language
- contains quantity information

We will explain extended discourse representation structures by means of an example. The full language of extended discourse representations structures is found in the report 'Discourse Representation Structures in Attempto Controlled English' [www.ifi.unizh.ch/attempto]

The Attempto Parsing Engine (APE) translates the ACE text

> Every company that buys a standard machine gets a discount. A British company buys a standard machine.

unambiguously into the following discourse representation structure

```
drs([A,B,C,D,E],[drs([F,G,H,I,J],[structure(G,atomic),
quantity(G,cardinality,count_unit,F,eq,1),
object(G,company),structure(I,atomic),
quantity(I,cardinality,count_unit,H,eq,1),
property(I,standard),object(I,machine),
predicate(J,event,buy,G,I)])=>drs([K,L,M],[structure(L,atomic),
quantity(L,cardinality,count_unit,K,eq,1),
object(L,discount),predicate(M,event,get,G,L)]),
structure(B,atomic),quantity(B,cardinality,count_unit,A,eq,1),
property(B,'British'),object(B,company),
structure(D,atomic),quantity(D,cardinality,count_unit,C,eq,1),
property(D,standard),object(D,machine),
predicate(E,event,buy,B,D)])
```

The first argument of the discourse representation structure *drs/2* is a list of discourse referents, i.e. quantified variables naming objects of the domain of discourse. In our example the discourse referents *A*, *B*, *C*, *D*, *E*, *K*, *L*, *M* are existentially quantified, and *F*, *G*, *H*, *I*, *J* being introduced in the precondition of an implication are universally quantified. The second argument of *drs/2* is a list of simple and complex conditions for the discourse referents. The list separator ',' stands for logical conjunction. Simple conditions are logical atoms, while complex conditions are built from other discourse representation structures with the help of the logical connectors negation '-', disjunction '*v*', and implication '=>'.

Logical atoms are formed from a small set of predefined predicates like *object/2*, *property/2*, or *predicate/5.* For example, instead of the usual *company(D)*, we reify the relation company, and write *object(D,company)*. This 'flat notation' allows us to quantify over the arguments of the

predefined predicates and thus to express general aspects of relations in first-order axioms that otherwise would require higher-order logic [Hobbs 1985].

The discourse representation structure gets a model-theoretic semantics [Kamp & Reyle 1993] that assigns an unambiguous meaning to the ACE text from which it was derived. Thus the Attempto system treats every ACE sentence as unambiguous, even if people may perceive the same sentence as ambiguous in full English.

4. Attempto Parsing Engine

4.1. Abstract Grammar of Attempto Controlled English

The syntax of ACE is described by an abstract grammar in the report 'The Syntax of Attempto Controlled English: An Abstract Grammar for ACE 4.0' [www.ifi.unizh.ch/attempto] from which the concrete grammar of ACE was derived.

The abstract grammar consists of approximately 100 grammar rules and about 50 definitions of function words.

Here is an excerpt of the abstract grammar describing topicalised sentences.

> A topicalized sentence can start with an existential topic (7) or a universal topic (8). It needs, however, not be topicalized at all but can just be an ordinary composite sentence (9)
>
> (7) TopicalizedSentence --> ExistentialTopic (such that SentenceCoord)
>
> Example: There is a card such that the code of the card is valid [.]
>
> (8) TopicalizedSentence --> UniversalTopic SentenceCoord
>
> Example: For every code there is a card such that the code belongs to it [.]
>
> (9) TopicalizedSentence --> CompositeSentence
>
> (10) ExistentialTopic --> ExistentialGlobalQuantor NPCoord[+NOM,-FORALL,-WH]/
>
> Examples: There is a card [which is valid.] There are a card and a code [such that the code is the code of the card.]
>
> (11) UniversalTopic --> UniversalGlobalQuantor N'[+NOM,-PL]
>
> Examples: For every card [there is a code.] For all money there is a bank.
>
> (12) UniversalTopic --> DistributiveGlobalQuantor NPCoord[+NOM,+PL]/
>
> Examples: For each of the customers [a clerk enters a code.] For each of a customer and a clerk [some code is valid.]

The abstract grammar rules use features to indicate required values, e.g. *+NOM*, or prohibited values, e.g. *-WH*, and thus enforce grammatical constraints.

4.2. Concrete Grammar of Attempto Controlled English: APE

The concrete grammar of ACE 4 is embodied in the Attempto Parsing Engine (APE) that was rewritten from scratch. APE uses a unification based phrase-structure grammar enhanced with feature structures. APE is implemented as a Definite Clause Grammar, i.e. as a Prolog program, and uses ProFIT [Erbach 95] to represent feature structures. The ProFIT preprocessor compiles these feature structures into regular Prolog arguments.

After the tokenisation of the input text, APE generates a syntax tree and the DRS. In a second pass APE resolves anaphoric references.

Unlike previous ACE parsers, APE does not yet generate a paraphrase of the input text, or error messages. As a help to users, warnings about words lacking in the vocabulary are given.

Reflecting the abstract grammar, APE consists of about 160 grammar rules. As an example we present here the concrete grammar rule equivalent to the abstract grammar rule (8) above.

```
%-8----------------------------------------------------------------------------
% TopicalisedSentence --> UniversalTopic SentenceCoord
%----------------------------------------------------------------------------

topicalised_sentence(display!tree![top_s,Topic,SCoord] &
        drs!in!DrsIn &
        drs!out!DrsOut &
        drs!pred!in!PredIn &
        drs!pred!out!PredOut &
        drs!evbef!EvBef &
        drs!id!ID &
        refres!analist!in!AnaListIn &
        refres!analist!out!AnaListOut &
        refres!antelist!in!AnteListIn &
        refres!antelist!out!AnteListOut
        ) -->
universal_topic(display!tree!Topic &
        drs!in!DrsIn &
        drs!out!DrsOut &
        drs!scope!in!ScopeIn &
        drs!scope!out!ScopeOut &
        sem!index!X &
        drs!id!ID &
        refres!analist!in!AnaListIn &
        refres!analist!out!AnaList1 &
        refres!antelist!in!AnteListIn &
        refres!antelist!out!AnteList1
        ),
sentence_coord(display!tree!SCoord &
        drs!in!ScopeIn &
        drs!out!ScopeOut &
        sem!arg1!X &
        drs!id!ID &
        drs!evbef!EvBef &
        drs!pred!in!PredIn &
        drs!pred!out!PredOut &
        refres!analist!in!AnaList1 &
        refres!analist!out!AnaListOut &
        refres!antelist!in!AnteList1 &
        refres!antelist!out!AnteListOut
        ).
```

The example clearly demonstrates the necessity to first introduce an abstract grammar before adding the intricacies of the feature structures and the complications of the resolution of anaphora.

4.3. Attempto Lexicons

As described above the vocabulary of ACE consists of

- domain-independent predefined function words (e.g. determiners, conjunctions, prepositions, …)
- user-defined, domain-specific content words (nouns, verbs, adjectives, adverbs)

Function words are a closed class. Most of them are represented as entries in APE's lexicon *flex*, while a small number, for instance *if* and *then* is directly embedded in APE's grammar rules.

As a system-internal lexicon *flex* uses a notation with ProFIT feature structures.

Content words form an open class and are often domain-specific APE provides the lexicon *clex* of content words with close to 100'000 entries. The lexicon *clex* can be complemented by the user-specific lexicon *ulex*. If a word occurs both in *ulex* and in *clex* then the definition of *ulex* prevails. While users cannot modify *clex*, they can modify *ulex* using APE's lexical editor or any text editor.

Entries of both *clex* and *ulex* are Prolog facts containing just the necessary linguistic information to represent a word. APE does not access the entries of *clex* and *ulex* directly, but indirectly via an interface that extends the Prolog facts with the feature structures needed by APE.

To temporarily use a content word not found in *clex* or *ulex*, users have the possibility to prefix the word with its word class and use it in this way in an ACE text. Here is an example. While the sentence

> John eats a kitkat.

will not be parsed because the noun *kitkat* is unknown, the sentence

> John eats a n:kitkat.

will be parsed.

5. Working with the Attempto Parsing Engine

There are basically three different ways to work with APE:

- locally, calling APE as Prolog program
- remotely, using APE and its lexical editor via their web-interfaces
- remotely, accessing APE via a remote procedure call

5.1. Calling APE as a Prolog Program

APE was originally implemented in SICStus Prolog but can also be executed by SWI Prolog.

After starting Prolog one compiles APE by executing

```
?- compile(ape).
```

To continue working on the command line one can enter

```
?- ape.
```

to get a prompt for ACE input

```
|:
```

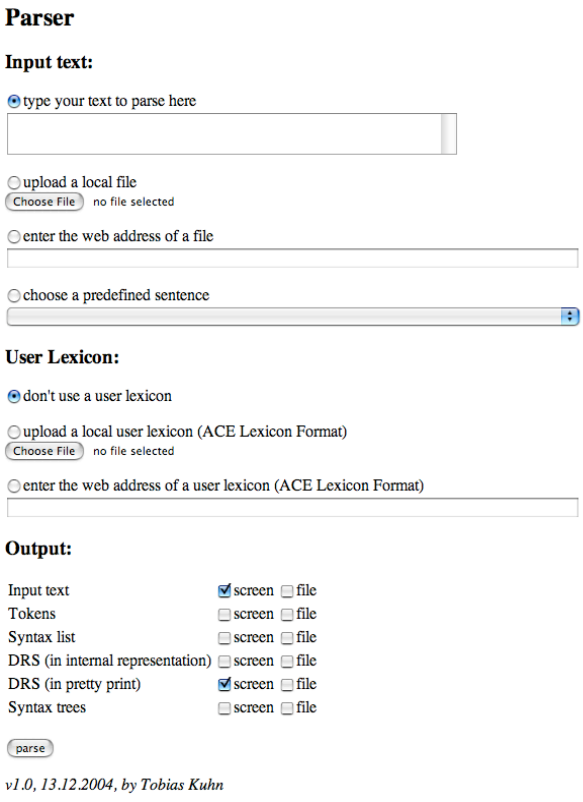
Alternatively, one can call – from the command line or from another Prolog program – one of several entries into APE, for instance

```
acetext_drs(+Text, -SyntaxTreesList, -DRS).
```

Other entries into APE provide additional information, for instance the token list.

5.2. APE Web-Interface

APE and its lexical editor LexEdit have been enhanced by web-interfaces implemented in Prolog using the Pillow library [clip.dia.fi.upm.es/Software/pillow/pillow.html].



APE's web-interface allows users

- to enter an ACE text into an edit field
- to upload a local ACE text
- to upload an ACE text via a URL
- to select one of the predefined ACE texts for demo purposes

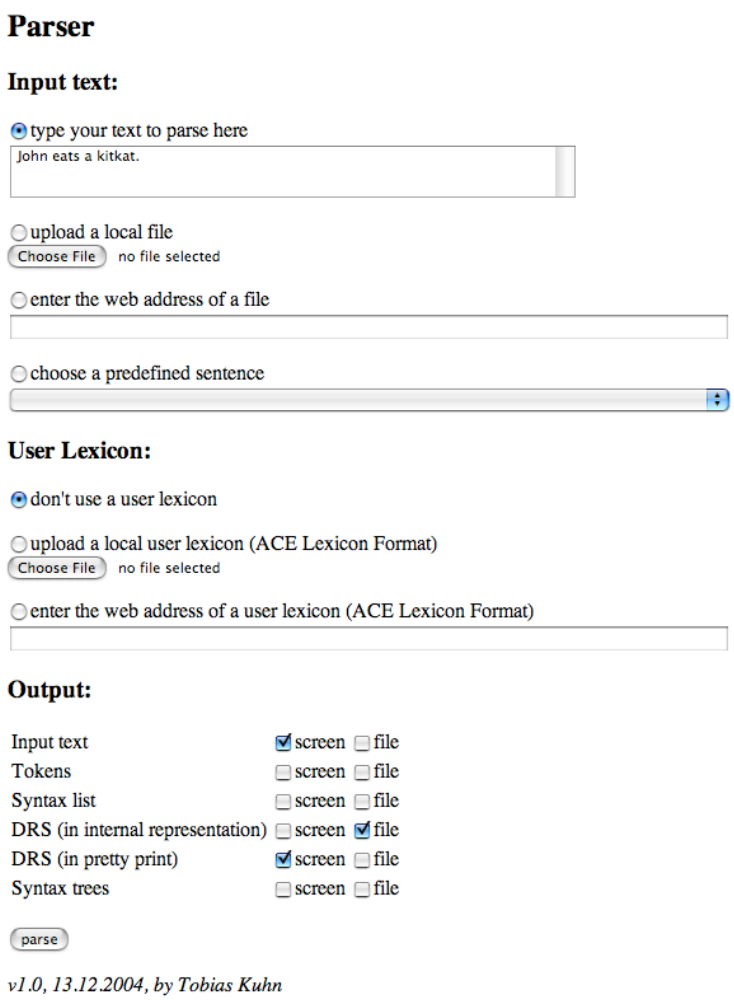
Users can then decide to parse their ACE text

- without a user lexicon
- with a locally uploaded user lexicon
- with a user lexicon uploaded via a URL

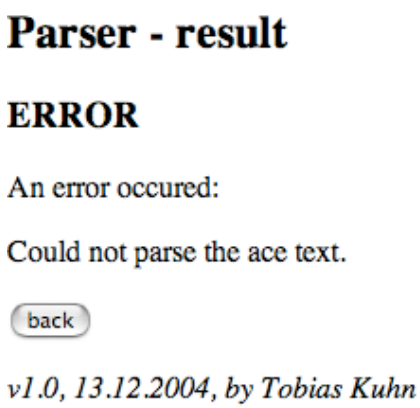
Concerning the output of APE users can view and/or locally download

- the input text
- the token list
- the syntax tree as a list
- the pretty-printed syntax tree
- the DRS as a Prolog term
- the pretty-printed DRS

Here is the result of trying to parse the ACE sentence *John eats a kitkat*.



Parsing fails



Parser - result

ERROR

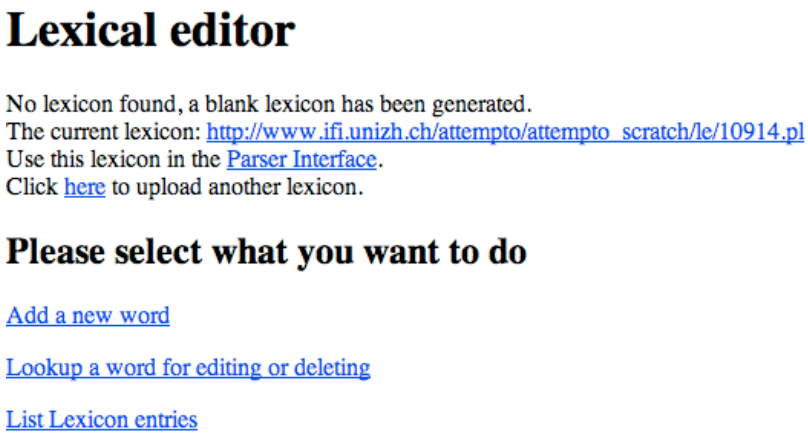
An error occured:

Could not parse the ace text.

back

v1.0, 13.12.2004, by Tobias Kuhn

since the noun *kitkat* is not known. We invoke APE's lexical editor LexEdit to add the word *kitkat* to *ulex*.



Lexical editor

No lexicon found, a blank lexicon has been generated.
The current lexicon: http://www.ifi.unizh.ch/attempto/attempto_scratch/le/10914.pl
Use this lexicon in the Parser Interface.
Click here to upload another lexicon.

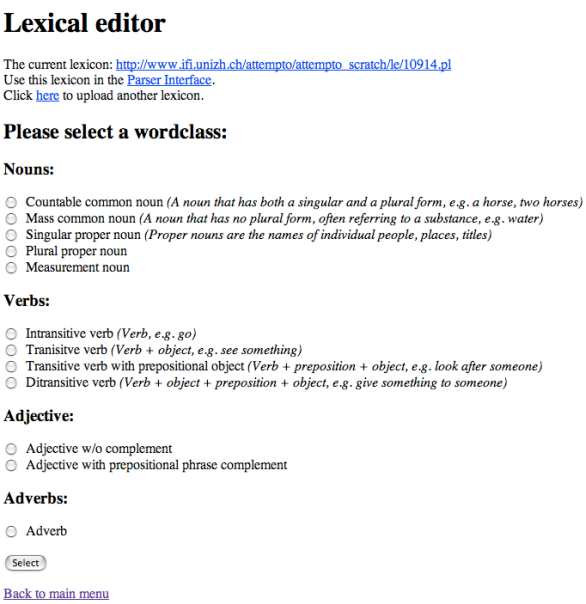
Please select what you want to do

Add a new word

Lookup a word for editing or deleting

List Lexicon entries

and add the missing word.



Lexical editor

The current lexicon: http://www.ifi.unizh.ch/attempto/attempto_scratch/le/10914.pl
Use this lexicon in the Parser Interface.
Click here to upload another lexicon.

Please select a wordclass:

Nouns:

- Countable common noun *(A noun that has both a singular and a plural form, e.g. a horse, two horses)*
- Mass common noun *(A noun that has no plural form, often referring to a substance, e.g. water)*
- Singular proper noun *(Proper nouns are the names of individual people, places, titles)*
- Plural proper noun
- Measurement noun

Verbs:

- Intransitive verb *(Verb, e.g. go)*
- Tranisitve verb *(Verb + object, e.g. see something)*
- Transitive verb with prepositional object *(Verb + preposition + object, e.g. look after someone)*
- Ditransitive verb *(Verb + object + preposition + object, e.g. give something to someone)*

Adjective:

- Adjective w/o complement
- Adjective with prepositional phrase complement

Adverbs:

- Adverb

Select

Back to main menu

Lexical editor

The current lexicon: http://www.ifi.unizh.ch/attempto/attempto_scratch/le/10914.pl
Use this lexicon in the Parser Interface.
Click here to upload another lexicon.

Countable common noun

Fields written in BOLD font MUST BE FILLED!!!
Please use "," to separate aliases (no space around the ","!)

Singular:	kitkat
Plural:	kitkats
Singular aliases:	
Plural aliases:	
Type:	○ person ◉ object ○ time
Gender:	◉ neuter ○ masculine/feminine ○ masculine ○ feminine
Collective Noun:	○ yes ◉ no
Comment:	

Submit

Back to wordclass selection
Back to main menu

Now we reparse the sentence using the newly created lexical entry.

Parser

Input text:

◉ type your text to parse here

John eats a kitkat.

○ upload a local file

Choose File no file selected

○ enter the web address of a file

○ choose a predefined sentence

User Lexicon:

○ don't use a user lexicon

○ upload a local user lexicon (ACE Lexicon Format)

Choose File no file selected

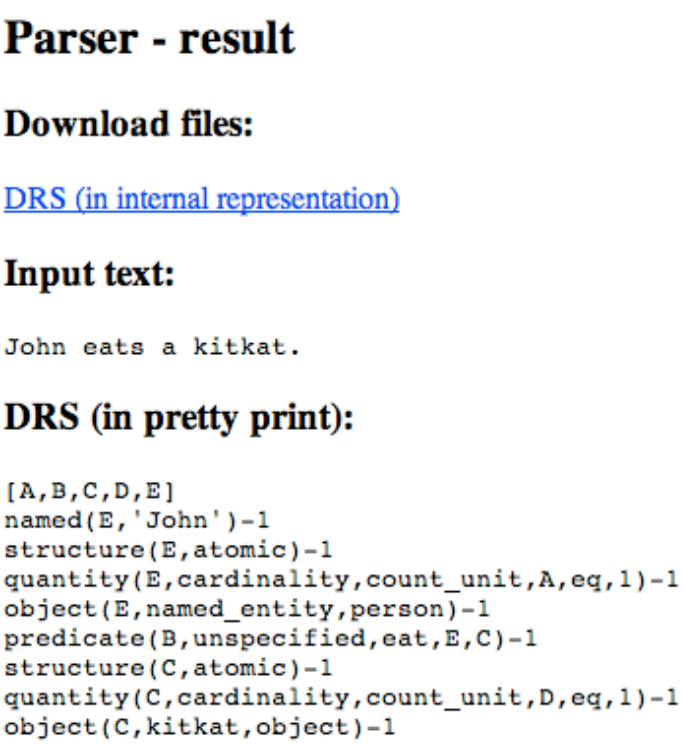
◉ enter the web address of a user lexicon (ACE Lexicon Format)

http://www.ifi.unizh.ch/attempto/attempto_scratch/le/10914.pl

Output:

Input text	☑ screen	☐ file
Tokens	☐ screen	☐ file
Syntax list	☐ screen	☐ file
DRS (in internal representation)	☐ screen	☑ file
DRS (in pretty print)	☑ screen	☐ file
Syntax trees	☐ screen	☐ file

to get



Parser - result

Download files:

DRS (in internal representation)

Input text:

```
John eats a kitkat.
```

DRS (in pretty print):

```
[A,B,C,D,E]
named(E,'John')-1
structure(E,atomic)-1
quantity(E,cardinality,count_unit,A,eq,1)-1
object(E,named_entity,person)-1
predicate(B,unspecified,eat,E,C)-1
structure(C,atomic)-1
quantity(C,cardinality,count_unit,D,eq,1)-1
object(C,kitkat,object)-1
```

Note that the index *-1* attached to the conditions of the DRS indicates that the conditions were derived from the first sentence of the ACE text. This information is needed by the ACE reasoner RACE, but is not relevant in this context.

We had asked for a version of the DRS that we can locally download. After clicking on 'DRS (in internal representation)' we get

drs([A,B,C,D,E],[named(E,'John')-1,structure(E,atomic)-1,quantity(E,cardinality,count_unit,A,eq,1)-1,
object(E,named_entity,person)-1,predicate(B,unspecified,eat,E,C)-1,structure(C,atomic)-1,
quantity(C,cardinality,count_unit,D,eq,1)-1,object(C,kitkat,object)-1]).

that we can download.

5.3. XMLHttpRequest

While the APE web-interface described in the previous section relies on the standard separation of work between CGI client and server, there is a more efficient and more flexible approach using XMLHttpRequest that allows connecting an HTML presentation directly to XML data for interim updates without reloading the page.

We have reimplemented part of the APE web-interface using XMLHttpRequest.

5.4. Remote Procedure Calls

Once APE runs on a server we can access it via remote procedure calls.

Using LWP – the WWW library for Perl – we have implemented a Perl script that provides a simple access mechanism to APE via HTTP POST.

Similar solutions are possible with the help of other programming languages such as Java, C#, Prolog, PHP.

Remote procedure calls to APE can form the basis of an APE web service.

6. Conclusion

In this report we presented the Attempto Parsing Engine (APE) that translates texts in Attempto Controlled English (ACE) into discourse representation structures, a variant of first-order logic.

The development of APE is not yet completed, though. To better support users the next version of APE will generate paraphrases of translated ACE texts, error messages and warnings. Furthermore, there is still room to improve the performance of APE. Last but not least, ACE will be extended which requires appropriate extensions of APE.

7. References

[www.ifi.unizh.ch/attempto] Attempto website containing a description of the Attempto project, a list of the people involved, demos, publications and various other information

[Erbach 95] G. Erbach, ProFIT: Prolog with Features, Inheritance and Templates, Research Report, Saarland University, Saarbrücken, 1995.

[Hobbs 1985] J. R. Hobbs, Ontological Promiscuity, in: Proceedings of the 23rd Annual Meeting of the ACL, University of Chicago, 1985

[Kamp & Reyle 1993] H. Kamp & U. Reyle, From Discourse to Logic, Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory, Kluwer, 1993