I2-D11

Attempto Controlled English 5: Language Extensions and Tools II

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/I2D11/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 April 2007
Actual date of delivery:	19 April 2007

Abstract

This report presents several tracks of research on Attempto Controlled English (ACE). First, we present the imperative mood as a new feature of version 5 of ACE. Second, we introduce an alternative simplified DRS representation of ACE texts. Third, we discuss aspects of the logical embedding of ACE. Fourth, we describe the verbalisation of OWL in ACE. Fifth, we present an enhanced version of AceRules, the implementation of rules expressed in ACE. Sixth, we describe enhancements to the Attempto services. Seventh, we list our cooperations.

Keyword List

Attempto Controlled English, ACE, Discourse Representation Structure, DRS, OWL, AceRules

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

Attempto Controlled English 5: Language Extensions and Tools II

Norbert E. Fuchs, Kaarel Kaljurand, Tobias Kuhn

Department of Informatics

&

Institute of Computational Linguistics

University of Zurich

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

19 April 2007

Abstract

This report presents several tracks of research on Attempto Controlled English (ACE). First, we present the imperative mood as a new feature of version 5 of ACE. Second, we introduce an alternative simplified DRS representation of ACE texts. Third, we discuss aspects of the logical embedding of ACE. Fourth, we describe the verbalisation of OWL in ACE. Fifth, we present an enhanced version of AceRules, the implementation of rules expressed in ACE. Sixth, we describe enhancements to the Attempto services. Seventh, we list our cooperations.

Keyword List

Attempto Controlled English, ACE, Discourse Representation Structure, DRS, OWL, AceRules

Contents

1. Introduction

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

- imperative mood as a new feature of version 5 of ACE
- a new DRS representation of ACE texts
- aspects of the logical embedding of ACE
- the verbalisation of OWL in ACE
- an enhanced version of AceRules
- a dedicated high performance Attempto server
- addenda to the Attempto website
- cooperations with REWERSE internal and external partners

The task "data structures and operations on them and procedural attachments" originally scheduled for this deliverable was postponed to deliverable I2-D13.

2. ACE Extension: Imperative Mood

Using Attempto Controlled English (ACE) in interactive environments, for instance to execute specifications, requires that users can enter commands and directives. Thus we extended ACE 5 by a simple form of imperatives. Here is an example

```
John, enter a card!
```

A proper name is followed by comma, an imperative sentence that contains one unnegated verb, and an exclamation mark.

Imperatives are translated as their declarative equivalent sentence, i.e.

```
John enters a card.
```

with the (pretty-printed) discourse representation structure (DRS)

```
[A, B, C]
named(A, John)-1
object(A, atomic, named_entity, person, cardinality, count_unit, eq, 1)-1
object(B, atomic, card, object, cardinality, count_unit, eq, 1)-1
predicate(C, unspecified, enter, A, B)-1
```

In the context of the interactive environment this DRS is not interpreted as being derived from a declarative sentence but from an imperative.

Experience will show whether this simple form of the imperative suffices, or whether it needs to be extended.

One possible extension would allow negated verbs

```
John, do not enter a card!
```

that could then be used to introduce constraints. Another possible extension generalises the addressees of commands

```
Every man who can fly a helicopter, step forward!
```

3. A Modified DRS Representation

3.1. Introduction

Texts in Attempto Controlled English (ACE) are translated into discourse representation structures (DRS)[1]. Like ACE, over the last years the DRS representations underwent changes, extensions and simplifications.

Here we summarise yet another modification of the DRS representation that takes diverse requirements of the ACE user community into account, and introduces a few extensions and modifications.

The new DRS representation has already been implemented and will be made public in the near future.

3.2. Types

The current DRS representation uses types for nouns, verbs and adverbs. Some users of ACE need these types, or even introduce their own types. Other users of ACE do not need types and remove them before further processing the DRS.

To accommodate both user communities we introduce a type field into the lexical entries of all content words (nouns, verbs, adjectives, adverbs). If not needed the type field can be left empty. During parsing the information of the type field is copied into the respective DRS condition.

Furthermore, we now generate a typed and a untyped DRS representation.

- typed representation: provides type arguments for DRS conditions derived from ACE content words
- untyped representation: is the typed representation without the type arguments

In any case, the Attempto Parsing Engine (APE) generates the typed DRS. That is, lexicons have to provide type arguments, or the placeholder `na` (= not available), for content words. Users of the APE web-service request whether they want the typed or the untyped DRS. If requested, the untyped representation is constructed from the typed one. Only then alternative DRS representations (XML, FOL, ...) are generated.

3.3. Data: Numbers and Strings

Numbers and strings can be used as in full English. Linguistically this means that they behave as complete singular noun phrases, i.e. they do not get a determiner, they cannot be modified by adjectives, saxon genitives, of-prepositional phrases, or relative phrases, and they cannot be conjoined. Furthermore, they cannot be anaphorically referred to. Numbers and strings are represented by a new DRS condition `data/3`.

Strings in apposition to noun phrases are no longer allowed, i.e. the condition `quoted_string/2` is no longer part of the DRS language.

3.4. Other Small Modifications

Here is a list of other small modifications:

- the DRS condition `relation/4` is slightly simplified
- the condition `proper_part_of/2` is renamed to `has_part/2`
- the DRS representation of the query word `how` is simplified
- the query words `where` and `when` are removed from ACE

For details see the forthcoming documentation.

[1] http://attempto.ifi.unizh.ch/site/pubs/papers/drs_report.pdf

4. Logical Embedding of Attempto Controlled English 5

4.1. Three Fragments of Attempto Controlled English

The syntax of Attempto Controlled English 5 (ACE 5) is described in the ACE Construction Rules[2]. To investigate the logical embedding of ACE we split its language constructs into three distinct fragments.

- BASE consists of all of ACE with the exception of modality, sentence subordination, and negation as failure; note that BASE contains logical negation (also called classical or strong negation).
- MOD consists of modality (possibility, necessity) and sentence subordination, both described in section 4.5 of the ACE Construction Rules.
- NAF consists of negation as failure (non-provability), as described in section 4.4 of the ACE Construction Rules.

4.2. Translation of ACE into Discourse Representation Structures

All fragments of ACE can be translated into Discourse Representation Structures (DRS) as described in the DRS Report[3].

- BASE can be translated into a variant of the standard DRS language [Blackburn & Bos 1999].
- BASE + MOD can be translated into an extension of the standard DRS language suggested by [Bos 2004]; the translation of the fragment MOD is described in sections 2.5, 2.6, 7.3 and 10 of the DRS Report.
- BASE + NAF can be translated into a further extension of the standard DRS language, modelled after the extension used for MOD; the translation of the fragment NAF is described in sections 2.3 and 9.4 of the DRS Report.

4.3. Verbalising DRSs in ACE

DRSs can be verbalised in ACE.

- All DRSs described in the DRS Report – in particular DRSs generated from ACE texts – can be verbalised in Core ACE, a syntactically simplified subset of ACE [cf. REWERSE I2-D5].
- A subset of DRSs can also be verbalised in NP ACE, a subset of ACE that uses rich noun phrases [cf. REWERSE I2-D9].
- The verbalisation tool DRACE – that translates DRSs into ACE – makes a 'best effort' to provide NP ACE where possible and falls back to Core ACE otherwise.
- DRACE can also serve as a syntax checker for acceptable DRS input.

4.4. Translation of ACE into First-Order Logic

Concerning the translation of the three ACE fragments into the standard language of first-order logic (FOL) we have the following results.

- BASE can be translated into a subset of FOL. This subset is defined by the translation of the respective DRS representations into the standard language of first-order logic [Blackburn & Bos 1999].
- BASE + MOD can be translated into a subset of FOL extended by a binary relation that stands for 'possible worlds' [Bos 2004].
- NAF cannot be translated into FOL

4.5. Verbalising FOL in ACE

Since FOL plays an important role as a representation language, its verbalisation in ACE would be of considerable interest.

---

[2] http://attempto.ifi.unizh.ch/site/docs/ace_constructionrules.html

[3] http://attempto.ifi.unizh.ch/site/pubs/papers/drs_report.pdf

Unfortunately, verbalising a FOL formula in ACE is not possible in general since one cannot translate an arbitrary FOL formula into a proper DRS, that is a DRS that could be verbalised in ACE. DRSs use a small set of predefined predicates in specific combinations. If a FOL formula does not contain the same – or semantically equivalent – predicates in the same combinations then the FOL formula cannot be translated into a proper DRS. Briefly, these constraints amount to the condition that a FOL formula can only be mapped to a DRS if the DRS could be ACE'ified, i.e. that the FOL formula *corresponds* to an ACE sentence.

Relevantly, these constraints are fulfilled for subsets of FOL, for instance the W3C languages OWL DL [Kaljurand & Fuchs 2006] and OWL 1.1 [see section 5 of this deliverable]. OWL ontologies can be verbalised in ACE since they could in principle be expressed in natural language.

4.6. Semantics

The BASE fragment and the combinations BASE + MOD and BASE + NAF can be given formal semantics.

- BASE can be given a standard first-order model-theoretic or proof-theoretic semantics [Schwertel 2003].
- BASE + MOD can be given a possible worlds model-theoretic semantics [Bos 2004]. We are working on a proof-theoretic semantics.
- BASE + NAF can, for instance, be given the semantics of Grosof's courteous logic programming [Grosof 1997], or alternatively a stable model semantics with or without logical negation [Gelfond & Lifschitz 1988, Gelfond & Lifschitz 1990].

4.7. Decidability

Concerning the decidability of ACE, we rely on results of [Pratt-Hartmann & Third 2006] who investigated the decidability of various fragments of natural language (NL).

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

BASE, and a fortiori ACE, is not decidable since it is a superset of the NL fragment `Cop + Rel + TV + GA` that Pratt-Hartmann & Third identified as undecidable.

Pratt-Hartmann & Third identified 4 decidable fragments of NL that are also decidable fragments of ACE:

```
Cop + TV + DTV
Cop + Rel
Cop + Rel + TV
Cop + Rel + TV + DTV
```

There is a draft version of a grammar that translates a ACE fragment into OWL 1.1. Since OWL 1.1 is decidable and the grammar is bi-directional the respective ACE fragment is also decidable.

Whether or not one of the decidable fragments of ACE offers sufficient expressivity, depends on the problem one tries to solve. Note that two major language constructs are missing which severely limits expressivity.

- The absence of anaphors allows only proper names to relate sentences.
- The absence of adverbs and prepositional phrases makes modification of verbs difficult.

If the problem does not require anaphors beyond proper names then one of the more comprehensive decidable subsets can suffice.

5. Verbalizing OWL in Attempto Controlled English

5.1. Introduction

The Web Ontology Language OWL has a normative syntax based on RDF and XML, languages that are oriented towards machines and thus inherently difficult to read and write for humans. OWL can be alternatively expressed in other RDF notations that do not use XML, or in dedicated OWL syntaxes like the functional-style OWL Abstract Syntax notation [Patel-Schneider et al. 2004], or in the concise syntax traditionally used for description logics. While easier to read and write for logicians and programmers, these alternative syntaxes lack the features that would bring OWL closer to domain experts who are likely not to be well-trained in formal methods. [Rector et al. 2004] list the problems that users encounter when working with OWL DL and express the need for a "pedantic but explicit" paraphrase language. In order to understand OWL, the users are also encouraged to use front-end tools. Such tools map OWL constructs into graphical user interface widgets (tabs, checkboxes, trees, etc.), but in general they too fail to hide the complexities of OWL.

An alternative and less explored approach is to use natural language as a front-end to OWL. In [Kaljurand & Fuchs 2006], we introduced the idea of a bidirectional mapping between OWL DL ontologies and Attempto Controlled English (ACE) texts. In this paper, we focus on the verbalization direction. Concretely, we discuss the details of verbalizing ontologies expressed in OWL 1.1 [Motik et al. 2006] – the likely successor to the OWL standard – without using data-valued properties and extra-logical constructs. A partial implementation of the verbalization covering the OWL DL subset of OWL 1.1 is publicly available[4]. This verbalization is reversible, i.e. the readers of the resulting ACE text can edit it and then convert it back into the normative OWL representation, and are thus able to communicate with OWL reasoners and other ontology tools.

This chapter is structured in the following way. In section 5.2 we review the related work, in section 5.3 we describe the mapping of OWL 1.1 into ACE, in section 5.4 we discuss the problems that we have encountered, in section 5.5 we discuss the implementation, and finally, in section 5.6 we draw conclusions and describe future work.

5.2. Related work

At the moment, the only way to explore the contents of OWL ontologies is to use OWL ontology editors. Such tools – TopBraid Composer[5], Protégé[6], SWOOP[7] – offer a graphical front-end with forms, trees, wizards, etc. to enable the writing and reading of ontologies. For complex class descriptions, however, they revert to using the syntax of description logics, and thus fail to hide the complexities of OWL. They also restrict the user in various ways, for example names have to be declared before they can be used and entering `SubClassOf`-axioms with a complex left-side is impossible in most tools. [Dzbor et al. 2006] compared TopBraid Composer and Protégé and found several problems that both novices and experts encountered. Recently, some tools have adopted the Manchester OWL Syntax [Horridge et al. 2006] as a means to enter complex class descriptions. Several features, for instance infix notation and English operator names, make the Manchester syntax more palatable than the traditional notation of description logics. However, the lack of determiners and specifically the heavy use of parentheses render it unnatural in comparison to English.

There is also existing work to provide more natural representations of OWL. [Hewlett et al. 2005] verbalize OWL class descriptions and use a part-of-speech tagger to analyze the linguistic nature of class names and then split the names apart to form more readable sentences. [Halaschek-Wiener et al. 2006] extend this work to OWL individuals and their properties. They also validate their approach by experimenting with seven university students, and find that they significantly prefer natural language verbalizations to the syntax of description logics, and even more so to the Abstract Syntax, Turtle and RDF/XML. [Mellish & Sun 2005a] discuss so called natural language directed inference to be applied to the ontology to make the verbalization of the ontology linguistically more acceptable. [Jarrar et al. 2006] verbalize OWL ontologies on the basis of predefined templates (e.g. Mandatory,

[4] http://attempto.ifi.unizh.ch/site/documentation/verbalizing_owl_in_controlled_english.html

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

[6] http://protege.stanford.edu

[7] http://www.mindswap.org/2004/SWOOP/

Exclusion, InterUniqueness). Each template contains canned text for one of a set of supported languages. [Lukichev & Wagner 2006] describe a verbalization of an XML-based rule format R2ML (which can be translated into SWRL [Horrocks et al. 2004]). In a more mixed approach, the ontology editor COE[8] uses natural language labels (such as "isMotherOf must be at least 2") on otherwise graphical representation of ontologies.

The major shortcoming of these approaches is that they lack any formal check that the resulting verbalizations are unambiguous. In this sense, a better approach is based on controlled natural languages that typically have a formal language semantics and come with a parser that could convert the verbalization back into the native OWL representation so that the verbalization is not a dead end, but rather a conversation turn in the machine-human communication. [Schwitter & Tilbrook 2004] discuss a mapping between the controlled English PENG and various OWL subsets (RDFS, Description Logic Programs, etc.). [Schwitter & Tilbrook 2006] extend this work to cover OWL DL (without data-valued properties) via a bidirectional mapping that is implemented as a Definite Clause Grammar. In this mapping, the `SubClassOf`-axiom is always written as an `if-then` sentence with explicit anaphoric references which for simpler axioms is unnecessarily hard to read. [Bernardi et al. 2007] provide a Categorial Grammar for a controlled English (Lite Natural Language) that expresses DL-Lite (a subset of OWL-Lite). Users may find Lite Natural Language somewhat unnatural since restrictions in DL-Lite, for instance that negations cannot occur on the left-hand side of the `SubClassOf`-axiom), are reflected in the syntax of Lite Natural Language.

Other expressive and recently developed versions of controlled English include CLCE [Sowa 2004] , Boeing's Computer Processable Language [Clark et al. 2005], and E2V [Pratt-Hartman 2003] that is shown to correspond to the decidable two-variable fragment of first-order logic. None of these controlled languages has been used for the verbalization of OWL ontologies, although most of them seem to have the required expressivity.

5.3. Verbalizing OWL

Verbalizing OWL ontologies in natural language and presenting the result as plain text has several advantages. There is no need for a dedicated and possibly complex ontology editor since plain text can be viewed and modified in any text editor. Plain text can also be easily stored, compared and searched with existing general tools. Presentation in natural language brings further benefits — natural language is understandable by any speaker of that language, it hides the formal syntax of OWL, it makes it possible to apply existing natural language processing tools such as spell checking and speech synthesis to the result.

When designing the verbalization, our first and most important decision was that the verbalization must be reversible, i.e. the mapping of OWL constructs into ACE constructs must be injective so that the resulting ACE text could be parsed and converted back into OWL, obtaining an ontology that is identical or at least semantically equivalent to the original. This feature makes sure that the output of the verbalization is not ambiguous with regards to the OWL semantics and that communication with OWL reasoners remains possible, which further enforces the user's correct understanding of the ontology.

Secondly, the verbalization must be acceptable and understandable English. This requirement is guaranteed by the ACE design decisions. Furthermore, we use ACE constructs, such as relative clauses, that provide conciseness. In order to increase readability, we also try to avoid anaphoric references. For instance, we prefer "Every man is a human." to the in ACE semantically equivalent "If there is a man then he is a human.".

Third, the OWL to ACE mapping must be compatible with the ACE semantics, for example `SubClassOf(dog animal)` must be mapped to a universally quantified (i.e. `if-then` or `every`) sentence and not to a sentence like "A dog is a kind of an animal." that in ACE is interpreted as having only existential quantification.

Finally, we try to leave the structure of the input ontology as far as possible intact. It must be visible to OWL experts how their constructs where mapped to ACE.

[8] http://cmap.ihmc.us/coe/

Now we describe the steps involved in the verbalization: rewriting some of the class descriptions and axioms via more general constructs, and the generation of ACE noun phrases and sentences[9].

5.3.1. Rewriting OWL constructs

The main intention behind rewriting OWL constructs via more general constructs is to replace constructs like `ObjectPropertyRange` that cannot directly be mapped to ACE. Also, a notion like range cannot be directly verbalized as the ACE word 'range' since this would most probably not confer the intended meaning. According to WordNet[10], the word 'range' has 9 meanings as a noun and 8 meanings as a verb. We therefore replace most OWL constructs with general `SubClassOf`, `SubObjectPropertyOf`, `DisjointObjectProperties`, and `ClassAssertion` axioms (see table 1 for the rewriting rules).

[9] For a detailed description of the ACE subset used in the verbalization, as well as a bidirectional Definite Clause Grammar for this subset, see http://attempto.ifi.unizh.ch/site/documentation/owlace constructionrules.html

[10] http://wordnet.princeton.edu

OWL classes and axioms	Equivalent OWL classes and axioms
owl:Nothing	ObjectComplementOf(owl:Thing)
ObjectOneOf(a_1 ... a_n)	ObjectUnionOf(ObjectOneOf(a_1) ... ObjectOneOf(a_n))
ObjectAllValuesFrom(R C)	ObjectComplementOf(ObjectSomeValuesFrom(R ObjectComplementOf(C)))
ObjectHasValue(R a)	ObjectSomeValuesFrom(R ObjectOneOf(a))
EquivalentClasses(C_1 ... C_n)	SubClassOf(C_1 C_2), SubClassOf(C_2 C_1), ...
DisjointClasses(C_1 ... C_n)	SubClassOf(C_1 ObjectComplementOf(C_2)), ...
DisjointUnion(A C_1 ... C_n)	*Rewriting via* SubClassOf, ObjectComplementOf and ObjectUnionOf.
SubObjectPropertyOf(R S)	SubObjectPropertyOf(SubObjectPropertyChain(R) S)
EquivalentObjectProperties(R_1 ... R_n)	SubObjectPropertyOf(R_1 R_2), SubObjectPropertyOf(R_2 R_1), ...
ObjectPropertyDomain(R C)	SubClassOf(ObjectSomeValuesFrom(R owl:Thing) C)
ObjectPropertyRange(R C)	SubClassOf(ObjectSomeValuesFrom(InverseObjectProperty(R) owl:Thing) C)
InverseObjectProperties(R S)	SubObjectPropertyOf(R InverseObjectProperty(S)), SubObjectPropertyOf(InverseObjectProperty(S) R)
FunctionalObjectProperty(R)	SubClassOf(owl:Thing ObjectMaxCardinality(1 R owl:Thing))
InverseFunctionalObjectProperty(R)	SubClassOf(owl:Thing ObjectMaxCardinality(1 InverseObjectProperty(R) owl:Thing))
ReflexiveObjectProperty(R)	SubClassOf(owl:Thing ObjectExistsSelf(R))
IrreflexiveObjectProperty(R)	SubClassOf(owl:Thing ObjectComplementOf(ObjectExistsSelf(R)))
SymmetricObjectProperty(R)	SubObjectProperty(SubObjectPropertyChain(R) InverseObjectProperty(R))
AntiSymmetricObjectProperty(R)	DisjointObjectProperties(R InverseObjectProperty(R))
TransitiveObjectProperty(R)	SubObjectPropertyOf(SubObjectPropertyChain(R R) R)
ObjectPropertyAssertion(R a b)	ClassAssertion(a ObjectSomeValuesFrom(R ObjectOneOf(b)))
NegativeObjectPropertyAssertion(R a b)	ClassAssertion(a ObjectComplementOf(ObjectSomeValuesFrom(R ObjectOneOf(b))))
SameIndividual(a_1 ... a_n)	ClassAssertion(a_1 ObjectOneOf(a_2)), ...
DifferentIndividuals(a_1 ... a_n)	ClassAssertion(a_1 ObjectComplementOf(ObjectOneOf(a_2))), ...

Table 1: Semantics-preserving rewriting of some OWL constructs.

5.3.2. Verbalizing OWL classes and properties

After rewriting, the remaining class descriptions map to ACE noun phrases and property descriptions map to active and passive verbs. Note that the class description `ObjectOneOf` maps to a proper name. See table 2.

OWL properties and classes	Examples of corresponding ACE verbs and noun phrases
Named property	*Transitive verb, e.g.* like
InverseObjectProperty(R)	*Passive verb, e.g.* is liked by
Named class	*Common noun, e.g.* cat
owl:Thing	something; thing
ObjectComplementOf(C)	something that is not a car; something that does not like a cat
ObjectIntersectionOf($C_1 \ldots C_n$)	something that is not a cat and that owns a car and that ...
ObjectUnionOf($C_1 \ldots C_n$)	something that is a cat or that is a camel or that ...
ObjectOneOf(a)	*Proper name, e.g.* John; something that is John
ObjectSomeValuesFrom(R C)	something that likes a cat
ObjectExistsSelf(R)	something that likes itself
ObjectMinCardinality(n R C)	something that rides at least 2 cars
ObjectMaxCardinality(n R C)	something that rides at most 2 cars
ObjectExactCardinality(n R C)	something that rides exactly 2 cars

Table 2: Verbalizing OWL property and class expressions as ACE verbs and noun phrases (including common nouns and proper names). Note that in actual verbalizations, the word 'something' is often replaced by a noun representing a conjoined named class.

In OWL, it is possible to build complex class descriptions from simpler ones by intersection, union, complementation and property restriction. Similarly, ACE allows building complex noun phrases via relative clauses that can be conjoined (by 'and that'), disjoined (by 'or that'), negated (by 'that is/does not') and embedded (by 'that'). While the mapping of boolean operators can be found in table 2, embedding allows us to use a relative clause to modify an object of another relative clause. For instance, the OWL class description

```
ObjectIntersectionOf(
   cat
      ObjectComplementOf(
         ObjectSomeValuesFrom(like
            ObjectIntersectionOf(
               dog
               ObjectUnionOf(
                  ObjectSomeValuesFrom(attack mailman)
                  ObjectOneOf(Fido))))))
```

can be verbalized in ACE as

```
something that is a cat and that does not like a dog that attacks a
mailman or that is Fido
```

Class descriptions in OWL can be syntactically arbitrarily complex as one can use parentheses to denote the scope of the expressions. ACE, however, has no support for parentheses. Scope ambiguities are resolved according to a small set of interpretation rules and the users have a choice between disentangling complex sentences, or using syntactic means to enforce the desired scoping.

For instance, the binding order of `and` and `or` favors and, but can be reversed by using a comma in front of and. This approach is natural (as natural language doesn't use parentheses for grouping) but for the verbalization process it poses a problem as very complex class descriptions cannot be mapped to ACE noun phrases.

For example, a relative clause can either modify the object (via 'that') or the subject (via 'and/or that') of a preceding relative clause, but not a more distant noun.

Therefore, complex class descriptions like

```
ObjectIntersectionOf(
   ObjectSomeValuesFrom(R1 ObjectSomeValuesFrom(R2 C1)
   ObjectSomeValuesFrom(R3 ObjectSomeValuesFrom(R4 C2)
)
```

cannot be handled by ACE directly.

The verbalization assumes that all names used in the ontology are English words. Furthermore, that individuals are denoted by singular proper names (preferably capitalized), named classes by singular common nouns, and (object) properties by transitive verbs in their lemma form (i.e. infinitive form). These restrictions are needed because the names will be used in certain syntactic constructions or will undergo certain morphological changes. Proper names are used in the subject and object positions without a determiner, e.g. “Every man knows John.”, “John is a man.”. Common nouns are used in the subject and object positions with determiners ‘every’, ‘a’, ‘at least 2’, etc., and can have a plural ending, e.g. “Every man owns at most 5 cars.”. Transitive verbs are often used in singular, but under negation and in plural will stay in infinitive, e.g. “Every person knows a child that does not own a bike and that has at least 3 friends that own a bike.”. In some cases, most often when verbalizing the ObjectPropertyRange-axiom, the verb will be turned into a past participle in order to construct a passive sentence, e.g. “Everything that is owned by something is a possession.”

In OWL ontologies, the names are URIs. Before morphological synthesis, the URI is truncated to preserve only the part that comes after the #-sign. In case the truncation would cause ambiguity in the names, the names are preserved as URIs.

5.3.3. Sentence planning

OWL axioms are mapped to ACE sentences (see table 3). Apart from sentences that are derived from the `ClassAssertion`-axioms, all sentences are `every`-sentences, i.e. they have a pattern `Noun-Phrase VerbPhrase`, where `NounPhrase` starts with every. If the verb phrase is negated, we move the negation into the noun phrase to obtain a simpler sentence (“Every dog is not a cat.” ⇒ “No dog is a cat.”).

OWL axioms	Examples of corresponding ACE sentences
SubClassOf(C D)	Every man is a human.
SubObjectPropertyOf(SubObjectPropertyChain(R_1 ... R_n) S)	Everything that owns something that contains something owns it.
DisjointObjectProperties(R_1 ... R_n)	Nothing that is-child-of something is-spouse-of it. ...
ClassAssertion(a C)	John is a man that owns at least 2 cars.

Table 3: Verbalizing OWL axioms as ACE sentences. Note that the anaphoric reference ‘it’ is resolved to the most recent noun according to the ACE interpretation rules.

In general, we try to keep the structure of the ACE sentence similar to the input axiom, and do not verbalize an axiom as several ACE sentences. Still, for better readability we apply certain modifications to the axioms before verbalizing: we remove negations as much as possible, e.g. “No man owns at most 5 books.” ⇒ “Every man owns at least 6 books.”, and reorder classes in coordination so that simple classes come first, e.g. “Everything that does not own a bike and that is a man and that owns a car . . . ” ⇒ “Every man that owns a car and that does not own a bike . . . ”. In our experience, even simple reordering can increase the readability significantly.

5.3.4. Text planning

Although an XML serialization has an order, there is no notion of order in the editing model of current OWL editors. Also, OWL parsers that build the functional syntax representation of RDF triples do not always keep the order that is originally in the XML/RDF file. Therefore, the planning of the verbalization cannot rely on any existing order and we output the ACE text simply in the order of verbalization of axioms about (1) individuals, (2) classes, and (3) properties.

As any linear order would have some shortcomings (for a particular reader), we have designed an alternative, so called “index view” of the ACE text to overcome the linearity by hypertext. In this view, the ACE text is “visualized” in HTML as an index, i.e. all names occurring in the ontology are ordered alphabetically and each name is “explained” by listing all the sentences that contain the name. Figure 1 shows an example that originates from the “people and pets” ontology[11].

[11] http://protege.cim3.net/file/pub/ontologies/people.pets/people+pets.owl

Figure 1: Index of names used in the verbalized ontology. The "definition" of only one name, has-pet, is presented.

5.4. Problems

5.4.1. OWL naming conventions

As the quality of the verbalization depends on the morphologic and orthographic nature of the names used for individuals, classes and properties in the input ontology, probably the most visible deficiency of the described verbalization is caused by the naming conventions used in OWL ontologies. The class, property and individual names are not under the control of current OWL editing tools and the user is guided only by informal style-guides, which mainly discuss the capitalization of names (see e.g. [Horridge et al. 2004]), or are specific to RDF such as the RoleNoun pattern[12]. Real-world OWL ontologies can contain class names like *Large*, *FifteenMinutes*, *NAMEDArtery*, *Urgent*, *mirrorImaged*; property names like *hasTopping*, *offeredIn*, *isPairedOrUnpaired*, *accountName*, *brotherOf*, *isWrittenBy*; and individual names like *red*, *married*. Such names do not lend themselves well to any verbalization scheme. Still, [Mellish & Sun 2005b] analyze the linguistic nature of class and property names in 882 public OWL ontologies and find that these names fall, in most cases, quite well into the categories of nouns and verbs, respectively, with only a small overlap in linguistic patterns used. Unfortunately, their study does not discuss object properties and data-valued properties separately, and does not analyze the morphological features of names of individuals.

Hopefully, names will become more English-like over time as ontology languages and tools evolve. Encouragingly, OWL 1.1 adds support for anonymous inverse properties (`InverseObject-` Property) and thus does not force the user to invent a new name just to be able to talk about an inverse of an

[12] http://esw.w3.org/topic/RoleNoun

existing property. Also, the practice of attaching nouns (i.e. class names) to property names might disappear in the presence of qualified cardinality restrictions.

5.4.2. Deeply nested class descriptions

Experience with real-world ontologies has shown that class descriptions can occasionally be very deep and branch in complex ways. ACE, on the other hand cannot handle deeply nested structures. We see various solutions to this. First, in an interactive environment a valid solution would be to reject class expressions that cannot be verbalized, so that the user could simplify them. We believe that such a natural restriction would result in more readable ontologies in the end. Secondly, we could add to ACE a support for parentheses. Third, we could use `if-then` sentences which are syntactically more expressive than `every`-sentences but lack conciseness. Fourth, we could try to automatically simplify the ontology by defining new named classes and using them to rewrite the ontology. There are several problems with the last approach – the modified ontology would not be equivalent to the original but would only entail it; the original structure of the class description (which the ontology author has maybe carefully constructed) is destroyed; meaningful names are difficult to construct automatically.

5.4.3. DisjointUnion, ONLYSOME, XOR, etc

OWL 1.1 includes powerful short-hand axioms like `DisjointUnion`, and other forms of syntactic sugar motivated by OWL usage patterns are discussed in the literature [Horridge et al. 2006]. ACE doesn't provide such short-hands and the verbalization will therefore unravel complex constructions. For instance, `DisjointUnion(person male female)` would be verbalized as

```
No male is a female. No female is a male. Every person is a male or is a
female.

Everything that is a male or that is a female is a person.
```

While this is a valid approach that explains the notion of a covering union of pair-wise disjoint classes to a novice OWL user, more experienced OWL users may prefer a more concise verbalization.

Also, the verbalization of a few other OWL constructs would profit if ACE provided a more concise corresponding syntax. E.g. at the moment, ACE doesn't support noun phrase disjunction and a function word such as 'only' or 'nothing but'. OWL's `ObjectOneOf` and `ObjectAllValuesFrom` would be expressed as

```
Every student is John or is Mary or is Bill.

No koala eats something that is not an eucalyptus-leaf.
```

instead of a shorter, but in ACE not allowed

```
*Every student is John, Mary, or Bill.

*Every koala eats nothing but eucalyptus-leaf.
```

5.4.4. Property axioms

The described verbalization can handle most of the OWL 1.1 constructs without using explicit anaphoric references because relative clauses can achieve the same argument sharing effect that anaphoric references would be otherwise needed for. For property axioms (`SubObjectPropertyOf` and `DisjointObjectProperties`) the situation is different. E.g. transitivity would be expressed as

```
Everything that contains something that contains something contains it.
```

where the personal pronoun 'it' refers to the most recent noun phrase (i.e. the last 'something'). Such sentences can be hard to read. Using `if-then` sentences with explicit variables might be most clear in the end.

```
If something X contains something Y and Y contains something Z then X
contains Z.
```

5.5. Implementation

The verbalization of OWL ontologies has been implemented in SWI-Prolog and is publicly available as a REST-webservice[13] which takes OWL RDF/XML as input and produces an ACE verbalization as output (optionally in the "index"-view). The implementation doesn't cover data-valued properties and can currently handle only the OWL DL subset of OWL 1.1. Also, it rejects axioms that contain too complex class descriptions (as described in section 5.4.2).

The implementation uses the Thea OWL Library[14] to map OWL RDF/XML into the OWL Abstract Syntax (in Prolog notation). Thea, in turn, uses SWI-Prolog's semweb library[15] to convert RDF/XML into RDF triples. We then apply a relatively straight-forward mapping of the OWL Abstract Syntax to Attempto DRS and use an existing DRS verbalizer that represents DRS implications as `every`-sentences [Fuchs et al. 2006]. Note that going through the DRS might seem like a detour but we are using the DRS as interlingua that allows us to handle also other languages like e.g. SWRL.

5.6. Conclusions and future work

We conclude that OWL can be verbalized in concise and understandable English provided that a certain naming style is adopted for OWL individuals, classes, and properties. In order to be able to compare our approach to existing ontology engineering approaches, we have experimentally integrated the mapping into the Protégé editor[16]. This allows us to perform usability tests with users who are familiar with current ontology editors.

Using an existing ontology editor as a host environment also alleviates some of the problems that we have encountered. For example, the host environment can take care of things that are easier to handle by forms (such as entering data about individuals) and wizards (e.g. entering `DisjointUnion`), and support maintaining a lexicon with all the required linguistic information used in the verbalization (e.g. forms of morphologically exceptional words). The users can thus profit from the synergy resulting from the combination of traditional form-based ontology editing and natural language-based editing.

[13] http://attempto.ifi.unizh.ch/site/documentation/verbalizing_owl_in_controlled_english.html

[14] http://www.semanticweb.gr/TheaOWLLib

[15] http://www.swi-prolog.org/packages/semweb.html

[16] For a demonstration of the integration into Protégé 4.0 alpha, see the screencast

http://attempto.ifi.unizh.ch/site/documentation/screencast_ace_in_protege.mov

6. AceRules: Executing Rules in ACE

6.1. Introduction

The AceRules rule system was introduced in the deliverable I2-D9 [Fuchs et al. 2006]. Since then, AceRules has undergone several changes. The most important one is that it does no longer rely solely on courteous logic programs, but it has now a multi-semantics architecture. Another important improvement of AceRules are the different available interfaces.

The goal of AceRules is to show that controlled natural languages can be used to represent and execute formal rule systems. ACE is used as input and output language. People with no experience of technical notations should be able to understand, write, and execute formal rules.

6.2. The AceRules System

AceRules is a rule system prototype using the controlled natural language ACE as input and output language. AceRules supports currently three different semantics that are implemented as two distinct interpreter modules. AceRules is designed for forward-chaining interpreters that calculate the complete answer set. The general approach of AceRules, however, could easily be adopted for backward-chaining interpreters. AceRules is publicly available as a web service and through two different web interfaces.

To make the point of AceRules clearer, let us consider the following simple program (i.e. rule set):

```
Everyone has a card.
If someone has a card and does not have a code then John gives him/her a
code.
No clerk has a code.
Every customer has a code.
Bill is a clerk.
Sue is a customer.
Mary does not have a code.
```

As we can see, this program looks perfectly like natural English. Obviously, the first four lines stand for rules and the last three lines denote facts. In order to execute this program, AceRules uses first the ACE parser to translate the program into a DRS. Next, the DRS is mapped to a rule structure which is then given to the respective interpreter. The interpreter calculates the answer of the program, i.e. the set of facts that can be deduced by the program. This answer is then backtranslated into an ACE text: (The courteous interpreter is used here.)

```
Sue is a customer.
Bill is a clerk.
John gives Mary a code.
John gives Bill a code.
Sue has a code.
Sue has a card.
Mary has a card.
John has a card.
Bill has a card.
It is false that Bill has a code.
It is false that Mary has a code.
```

Since this answer is written in ACE again, no other formal notations are needed for the user interaction. Even though inexperienced users might not be able to understand how the answer is inferred, they are certainly able to understand input and output and to verify that the output is some kind of conclusion of the input. This is the essential advantage of ACE over other kinds of formal knowledge representation languages.

Existing work to use natural language representations for rule systems is based on the idea of verbalizing rules that already exist in a formal representation [Halpin 04, Jarrar et al. 06, Lukichev & Wagner 06]. In our approach, the controlled natural language is the *main* language that can be translated into a formal representation (parsing) and backwards (verbalizing). It is not necessary that the rules are first formalized in another language.

6.2.1. Semantics

In the beginning, AceRules relied on courteous logic programs [Grosof 1997]. Now, AceRules is redesigned to support various semantics. The decision of which semantics to choose should depend on the application domain, the characteristics of the available information, and on the reasoning tasks to be performed. Thus, this decision does not have to be taken by the end-user but by the system designer. At the moment, AceRules incorporates three different semantics: courteous logic programs, stable models [Gelfond & Lifschitz 1988], and stable models with strong negation [Gelfond & Lifschitz 1991]. The following table shows an overview. The upper part of the table lists properties of the respective theory. The second part shows properties of the integration into AceRules.

	courteous	stable	stable s.n.
Strong negation	X		X
Negation as failure	X	X	X
Priorities	X		
Cycles allowed		X	X
Number of answers	1	1–*	0–*
Trace support in AceRules	X		
Interpreter integration in AceRules	native	external	external

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 allow therefore for both forms of negation.

None of the two forms of stable models guarantee a unique answer set. Thus, a certain program can lead to several answers. Courteous logic programs eliminate this downside by the introduction of priorities and by the restriction to acyclic programs [Apt & Bezem 1990]. The deliverable I2-D9 [Fuchs et al. 2006] showed how the priorities are represented. We will not discuss here the acyclic-restriction, apart from saying that it can be a heavy restriction in some situations. The advantage of courteous semantics is that every program delivers exactly one answer.

On the basis of these properties, a system designer should decide which semantics to choose. Since we do not want to restrict AceRules to a certain application or domain, we decided to make the semantics exchangeable.

The three semantics are implemented in AceRules as two interpreter modules. The first interpreter module handles courteous logic programs and is implemented natively. (The implementation of the courteous interpreter is based on [Doerflinger 2005]). For the stable model semantics with and without strong negation there is a second interpreter module that wraps the external tools *Smodels* [Niemela & Simons 1997] and *Lparse* [Syrjanen 2000].

There are various other semantics that could be supported, e.g. defeasible logic programs [Nute 1994, Antoniou et al. 2000] or disjunctive stable models [Przymusinski 1991]. Only little integration effort would be necessary to incorporate these semantics into AceRules.

6.2.2. Two Kinds of Negation

The handling of two kinds of negation (negation as failure and strong negation) was already explained in the deliverable I2-D9 [Fuchs et al. 2006]. As a new feature, we support now another syntactic variant for negation as failure. Instead of

```
there is a customer and it is not provable that the customer waits
```

we can now also write

```
a customer does not provably wait
```

This new construct “is/does not provably” allows us to express some statements in a more concise way. Note that the two examples above are translated into the same DRS.

We think that these representations are well understandable. Even for persons that have never heard of negation as failure, they *makes some sense*, even though they are probably not able to grasp all its semantic properties.

6.2.3. Intelligent Grouping

Rules expressed in ACE are potentially more complex than the rules allowed in a certain rule theory. Thus, some rules have to be rejected by AceRules because they can not be handled by the respective interpreter. Nevertheless, there are situations where the logical representation (that is generated by the ACE parser) is not compliant with the rule theory, but can be translated in a consistent way into a correct rule representation. This translation we call "intelligent grouping".

To make this point clear, we present some examples using stable model semantics with strong negation. The described procedure can be used in the same way for the other semantics. Rules of the stable model semantics with strong negation have the form

```
L0 <- L1, ... Lm, ~Lm+1, ... ~Ln
```

with 0 =< m =< n and each Li being a literal. A literal is an atomic proposition (Ai) or its strong negation (-Ai). Negations are allowed to be applied only to atomic propositions or – in the case of negation as failure (~) – to literals. Furthermore the heads of rules must contain nothing but a single literal. These restrictions we have to keep in mind when we translate an ACE text into a rule representation. As a first example, let us consider the following AceRules program:

```
John owns a car.
Bill does not own a car.
If someone does not own a car then he/she owns a house.
```

The ACE parser transforms this text into its logical representation (simplified):

```
owns(john, X)
car(X)
- ( owns(bill, Y), car(Y) )
- ( owns(A, B), car(B) ) -> ( owns(A, C), house(C) )
```

which is not yet compliant with the rule theory. It contains complex terms inside of a negation and in the head of a rule. But considering the initial text, we would expect this example to work. In fact, it was just formalized in an inappropriate way. This is the point where the intelligent grouping is applied. If we aggregate some of the predicates then we end up with a simpler representation that has a correct rule structure:

```
owns_car(john)
- owns_car(bill)
- owns_car(X) -> owns_house(X)
```

This transformation is based on a set of group patterns that are collected in a first step, and then these patterns are used to perform the actual grouping. For our example, the following two patterns have been used:

```
owns(X1, I1), car(I1) ==> owns_car(X1)
owns(X2, I2), house(I2) ==> owns_house(X2)
```

In such patterns, there can be two kinds of placeholders: Each Xi stands for any variable or atom, and each Ii stands for a variable that does not occur outside of the group. This allows us to omit the variables Ii after the transformation. From a more intuitive point of view, we can say that the phrases "owns a car" and "owns a house" are considered atomic. This means that the car and the house do not have an independent existence, and thus references to these objects are not allowed. If this restriction is violated then a consistent transformation into a valid rule structure is not possible. For example, the program

```
Bill does not own a car.
John owns a car X.
Mary sees the car X.
```

that leads to the logical representation

```
- (owns(bill, A), car(A))
owns(john, B)
car(B)
sees(mary,B)
```

cannot be translated into a valid rule structure. An error message has to be raised in such cases informing the user that the program has an invalid structure. It has still to be evaluated how hard it is for normal users to follow this restriction and how often such situations actually occur.

Concerning the grouping step, one might think that the text was just translated into a too complex representation in the first place and that the parser should directly create a grouped representation. The following program shows that this is not the case:

```
John owns a car.
The car contains a suitcase.
If someone owns something that contains something X then he/she owns X.
```

It is transformed by the ACE parser into:

```
owns(john,H)
car(H)
contains(H,S)
suitcase(S)
owns(Z,X), contains(X,Y) -> owns(Z,Y)
```

In this case, we need the more fine-grained representation and no grouping has to be done. In other words, the pattern collection step returns an empty set of patterns because the program is already in a compliant form. This and the first example start both with the sentence "John owns a car", but in the end it has to be represented differently. Thus, the grouping is intelligent in the sense that it must consider the whole program to find out which predicates have to be grouped.

Another important property of the grouping step is that the transformation has to be reversible. Before verbalizing an answer set, we need to ungroup the predicates that have been grouped. In order to achieve reversibility, we use a more complex format than the one that is introduced here.

Altogether, the intelligent grouping gives us much flexibility. A sentence like "John owns a car" is treated as an atomic property of an object (John) or as relation between two objects (John and a car), whichever makes sense in the respective context.

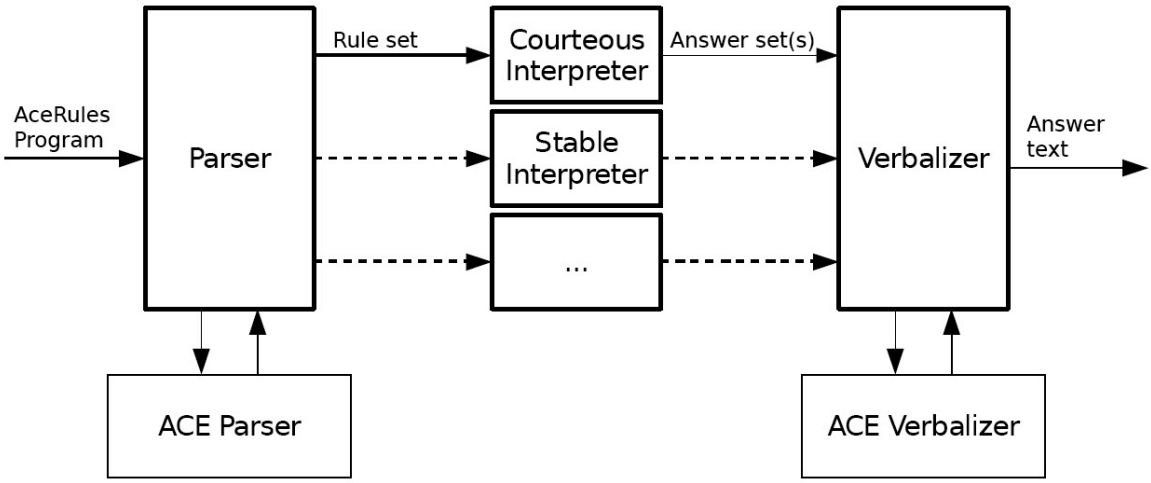
6.2.4. Trace

Sometimes, it can be very helpful to look inside of the process of the execution of a program, e.g. for debugging. AceRules allows to trace the program execution and it verbalizes this trace information in ACE. The trace output shows step-by-step how the final answer was deduced. Besides debugging, users can learn from the trace output in an intuitive way how a program is executed. In AceRules, this trace output can only be created for the courteous semantics. This is because the external programs used for the other semantics do not support any trace feature. The following picture shows how the AceRules web interface displays the trace output. A possible improvement would be to show not only the temporary answer sets, but also the rules that are applied.

6.3. Architecture

The picture below shows the architecture of AceRules. (Rectangles indicate modules and arrows indicate data flow.) Basically, a program is processed in three phases: parsing, interpretation, and verbalization. The core modules of AceRules are the parser and the verbalizer, whereas the

interpreter is exchangeable. An internal rule format using Prolog notation is the interface language between the parser and the interpreter and between the interpreter and the verbalizer.



We will now describe the details of the interpreter step. The parser step and the verbalizer step have been explained in the deliverable I2-D9 [Fuchs et al. 2006].

Since the interpreter component is exchangeable, this step has to be described in an abstract manner. The interpreter step can generally be subdivided into four sub-steps, where only the steps 1 and 3 are mandatory.

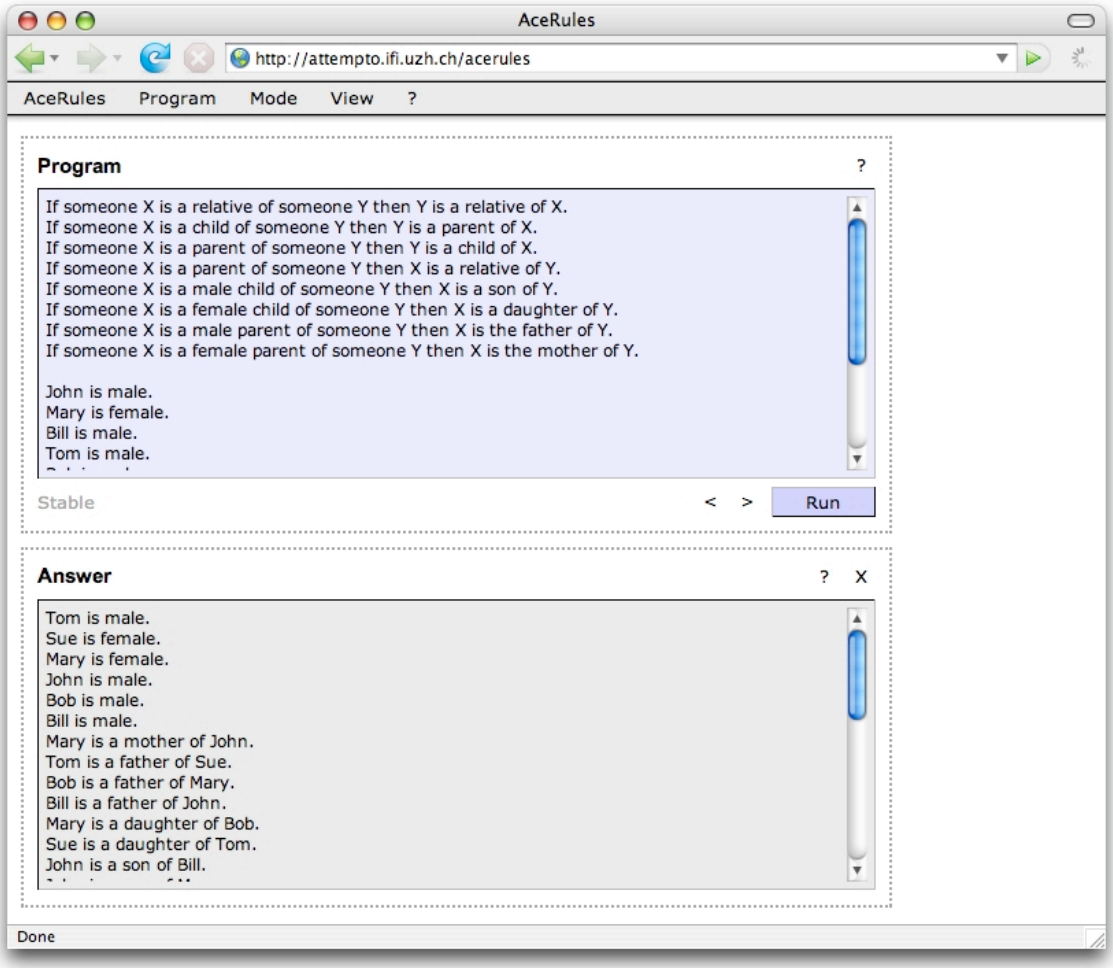
1. (Compliance check) First, the program is checked if it fulfills the semantics-specific restrictions. A sentence like "John is not a customer" is not allowed in stable model semantics without strong negation, but it is allowed in the two other semantics. Thus, this compliance check cannot be done by the parser but has to be done by the interpreter.
2. (Preprocessing) Depending on the inference algorithm, some preprocessing may be necessary. E.g. unbound variables have to be eliminated if the algorithm cannot handle them.
3. (Calculation of the answer set) This step does the actual answer set calculation. It can be delegated to an external program.
4. (Postprocessing) The resulting answer set may require some postprocessing. E.g. auxiliary predicates that were introduced during the answer set calculation have to be removed.

Furthermore, if an external program is used for the answer set calculation then the pre- and postprocessing steps perform the mapping to the required rule format.

6.4. Interfaces

There are different existing controlled natural language interfaces for rule systems or for the Semantic Web in general. Most of them are query interfaces like *Ginseng* [Bernstein et al. 05] or *LingoLogic* [Thompson et al. 05]. We will focus here on presentation and editor interfaces like *NORMA* [Halpin & Curland 2006] or *ECOLE* [Schwitter et al. 2003]. There are also interfaces like *GINO* [Bernstein & Kaufmann 2006] which can be used for queries and as an editor.

AceRules comes with three interfaces. A webservice [AceRules-Webservice] allows to integrate the AceRules functionality into any other program. Furthermore, there are two web interfaces for human interaction. One is a technical interface (`http://attempto.ifi.uzh.ch/acerules_ti`) that is intended for advanced users that are interested in the technical background of the system. The main web interface (`http://attempto.ifi.uzh.ch/acerules`) aims to be convenient for potential end-users that are not familiar with formal notations. For the rest of this section, we will take a closer look at this main interface. The following picture shows a screenshot.

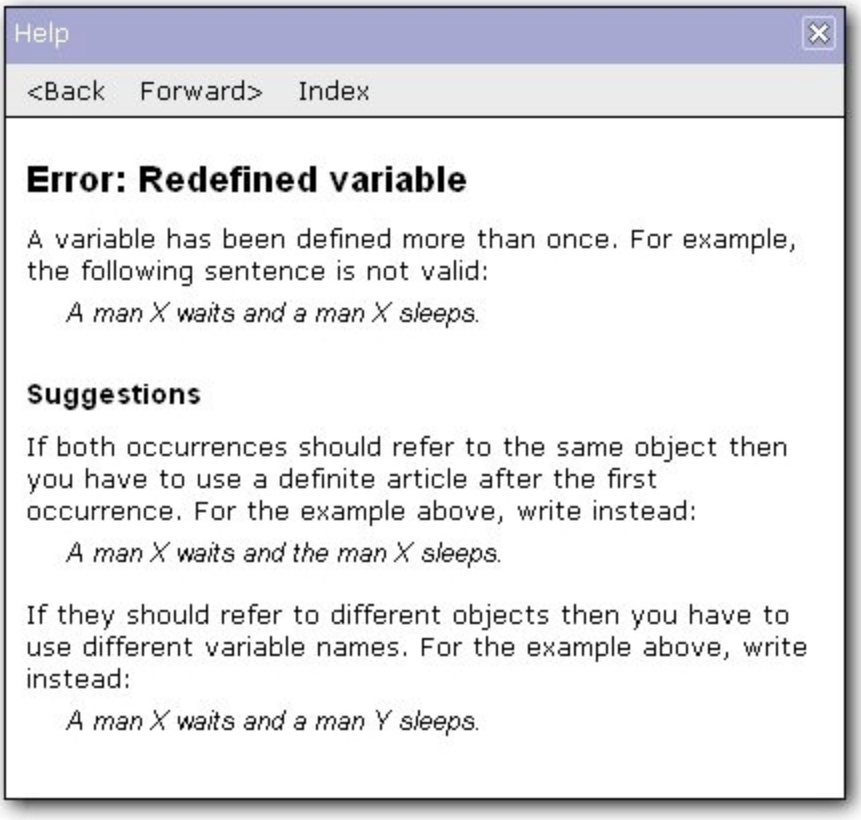


We claim that a Semantic Web interface for end-users (in the sense of an editor interface) should fulfill the following three properties which partly overlap. They hold for ontologies and rules in the same way.

1. Technical notations should be hidden. The users should not see any technical language (e.g. any XML-based language), but instead there should be a well-understandable and intuitive representation. Novice users should be able to understand the semantic representations after a very short learning phase.
2. The interface should guide the users during modification and creation of the formal representations. The users should not need to read any manuals or other documentation before they can start using the system, but they should be able to learn how to interact with the system while using it.
3. The users should be supported by a context-sensitive and comprehensive help feature. Especially in the case of errors, the users should be led immediately to a corresponding help article. These articles should be concise suggestions how to solve the problem.

Altogether, these three properties ensure that the Semantic Web interface has a shallow learning curve which we consider to be crucial for the success of the Semantic Web.

AceRules uses ACE as input and output language. No other notations are needed. Thus, AceRules fully satisfies the first condition. Furthermore, AceRules has a help feature which is shown in a browser-internal window. The help window is browsable in the sense that articles can contain links pointing to other articles. There is a help article for every error that can occur, as shown in the picture below. If an error has occurred, then the user is directed to the respective article. Thus, AceRules fulfills the third condition as well.



AceRules gives also some help for the modification of existing programs and for the creation of new sentences. Nevertheless, we have to admit that it can only partially fulfill the second condition. For a real guidance of the user, a predictive editor as in the ECOLE interface would be needed. Such an editor guides the user step-by-step through the creation of a sentence and makes it impossible to create syntactically incorrect representations. ECOLE uses a controlled natural language called *PENG* [PENG-Nutshell] which is very similar to ACE.

7. Attempto Webservices and Website

7.1. Webservices

The Attempto webserver was ported to a dedicated machine, resulting in overall performance gains of all tools by a factor 3 - 5.

Furthermore, there were several improvements to the Attempto Parsing Engine (APE):

- faster response times (up to 3x in case of short input texts)
- better error and warning messages
  - missing determiner: John owns car.
  - undefined propername: Peeter owns a car.
  - malformed file, in case the user lexicon doesn't conform to the specification
  - no longer warning message for possessive pronouns
- paraphraser
  - webclient now supports just one type of paraphrase
  - webservice continues to support both previously available paraphrasing modes
  - paraphrase helps users to realise that sentences like "a cat is an animal" are existentially, not universally, quantified
- new concise lexicon format
- lexicon now supports past participle
- modality in questions supported: Can John wait?

There are several new Attempto tools:

- AceRules webservice
- 2 AceRules webinterfaces
- AceWiki webinterface
- improved DRS ⇒ OWL translation supporting key features of OWL 1.1
- webinterface of the OWL verbaliser

7.2. Website

Several items were added to the Attempto website – among them

- screencast of the ACE ⇔ OWL translations
- screencast of AceRules
- listing of the built-in lexicon of APE
- OWL ACE construction rules
- proposal for ACE authoring tools

8. Cooperations

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:

- REWERSE I1 started to work on the translation of ACE into R2ML
- REWERSE I2 continues to work on using ACE as input language for the Protune system
- several REWERSE external groups make use of the APE web service: for instance the Rensselaer Polytechnic Institute (USA) and the Weizman Institute (Israel)
- N. E. Fuchs spent 2 months at Macquarie University (Sydney); result is an ESWC'07 paper on web annotations in controlled natural language
- a new cooperation with Yale University and other partners in the field of clinical practice guidelines started in April 2007; funding by NIH

9. Conclusions

Here is the updated contents of the two remaining deliverables of the controlled natural language track of I2.

9.1. Deliverable I2-D13: Reasoning, Rules, and Semantic Wikis

We will extend the ACE reasoner RACE to give explanations to “why/why not” queries and to investigate hypothetical “what if” queries. On the basis of the AceRules system we will provide an interface that facilitates the use of ACE for existing rule systems, for instance the Protune policy rules. We will extend and redesign the AceWiki system to demonstrate how ACE can be used in the context of semantic wikis. Furthermore, we will extend ACE by data structures and operations on them and by procedural attachments.

9.2. Deliverable I2-D15: Knowledge Assimilation and Abductive Reasoning

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”. Since this will be the last REWERSE deliverable, we will try to collect and to evaluate user feedback on the usability and acceptability of ACE.

10. References

[AceRules-Webservice] AceRules Webservice. Attempto Documentation, 9 January 2007, http://attempto.ifi.uzh.ch/site/documentation/acerules_webservice.html

[Antoniou et al. 2000] G. Antoniou, M.J. Maher, D. Billington. Defeasible logic versus Logic Programming without Negation as Failure. In The Journal of Logic Programming 42, 47-57, 2000

[Apt & Bezem 1990] Krzysztof R. Apt, Marc Bezem. Acyclic programs. In Logic Programming, 617-633, MIT Press, 1990

[Bernardi et al. 2007] Raffaella Bernardi, Diego Calvanese, and Camilo Thorne. Lite Natural Language. In IWCS-7, 2007.

[Bernstein & Kaufmann 2006] Abraham Bernstein, Esther Kaufmann. GINO - A Guided Input Natural Language Ontology Editor. 5th International Semantic Web Conference (ISWC 2006), 144-157, Springer, 2006

[Bernstein et al. 2005] Abraham Bernstein, Esther Kaufmann, Christian Kaiser. Querying the Semantic Web with Ginseng: A Guided Input Natural Language Search Engine. 15th Workshop on Information Technology and Systems (WITS 2005), 45-50, 2005

[Blackburn & Bos 1999] L. Blackburn, J. Bos. Representation and Inference for Natural Language, A First Course in Computational Semantics. Vol. II, Working with Discourse Representation Structures. 1999

[Bos 2004] J. Bos. Computational Semantics in Discourse: Underspecification, Resolution and Inference. Journal of Logic, Language and Information. 13:139-157. 2004

[Clark et al. 2005] Peter Clark, Philip Harrison, Thomas Jenkins, John Thompson, and Richard H. Wojcik. Acquiring and Using World Knowledge Using a Restricted Subset of English. In FLAIRS 2005, pages 506–511, 2005.

[Doerflinger 2005] Marc Dörflinger. Interpreting Courteous Logic Programs, Diploma Thesis. Department of Informatics, University of Zurich, 2005

[Dzbor et al. 2006] Martin Dzbor, Enrico Motta, Carlos Buil, Jose Gomez, Olaf G¨orlitz, and Holger Lewen. Developing ontologies in OWL: An observational study. In OWLED 2006, 2006.

[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.

[Gelfond & Lifschitz 1988] Michael Gelfond, Vladimir Lifschitz. The stable model semantics for logic programming. In Proceedings of the 5th International Conference on Logic Programming, 1070-1080, MIT Press, 1988

[Gelfond & Lifschitz 1991] Michael Gelfond, Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. In New Generation Computing, 9:365-385, 1990

[Grosof 1997] Benjamin N. Grosof. Courteous Logic Programs: Prioritized Conflict Handling For Rules. IBM Research Report RC 20836. Technical report, IBM T.J. Watson Research Center, 1997

[Halaschek-Wiener et al. 2006] Christian Halaschek-Wiener, Jennifer Golbeck, Bijan Parsia, Vladimir Kolovski, and Jim Hendler. Image browsing and natural language paraphrases of semantic web annotations. In First International Workshop on Semantic Web Annotations for Multimedia (SWAMM), Edinburgh, Scotland, May 22nd 2006.

[Halpin 2004] Terry Halpin. Business Rule Verbalization. In Anatoly Doroshenko, Terry Halpin, Stephen W. Liddle, Heinrich C. Mayr (editors), Proceedings of Information Systems Technology and its Applications, 3rd International Conference ISTA 2004, Lecture Notes in Informatics, 2004

[Halpin & Curland 2006] Terry Halpin, Matthew Curland. Automated Verbalization for ORM 2. In Robert Meersman, Zahir Tari, Pilar Herrero et al. (editors), On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops, Lecture Notes in Computer Science 4278, 1181-1190, Springer, 2006

[Hewlett et al. 2005] Daniel Hewlett, Aditya Kalyanpur, Vladimir Kolovski, and Chris Halaschek-Wiener. Effective Natural Language Paraphrasing of Ontologies on the Semantic Web. In End User Semantic Web Interaction Workshop (ISWC 2005), 2005.

[Horridge et al. 2006] Matthew Horridge, Nick Drummond, John Goodwin, Alan Rector, Robert Stevens, and Hai H Wang. The Manchester OWL Syntax. In 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 Prot´eg´e-OWL Plugin and COODE Tools. Edition 1.0. Technical report, The University Of Manchester, 2004. http://www.coode.org/resources/tutorials/.

[Horrocks et al. 2004] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission 21 May 2004. Technical report, W3C, 2004. http://www.w3.org/Submission/ 2004/SUBM-SWRL-20040521/.

[Jarrar et al. 2006] Mustafa Jarrar, Maria Keet, and Paolo Dongilli. Multilingual verbalization of ORM conceptual models and axiomatized ontologies. Technical report, Vrije Universiteit Brussel, February 2006.

[Kaljurand & Fuchs 2006] Kaarel Kaljurand and 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.

[Lukichev & Wagner 2006] Sergey Lukichev and Gerd Wagner. Deliverable I1-D6. Verbalization of the REWERSE I1 Rule Markup Language. Technical report, REWERSE, 2006. http://rewerse.net/ deliverables.html.

[Mellish & Sun 2005a] Chris Mellish and Xiantang Sun. Natural Language Directed Inference in the Presentation of Ontologies. In 10th European Workshop on Natural Language Generation, Aberdeen, Scotland, August 8–10th 2005.

[Mellish & Sun 2005b] Chris Mellish and Xiantang Sun. The Semantic Web as a Linguistic Resource. In Twentysixth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence, Peterhouse College, Cambridge, UK, December 12–14th 2005.

[Motik et al. 2006] Boris Motik, Peter F. Patel-Schneider, and Ian Horrocks. OWL 1.1 Web Ontology Language Structural Specification and Functional-Style Syntax. Technical report, W3C, 2006. http: //www.w3.org/Submission/owl11-owl\ specification/.

[Niemela & Simons 1997] I. Niemelä and P. Simons. Smodels – an implementation of the stable model and well-founded semantics for normal logic programs. In Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning, Lecture Notes in Artificial Intelligence, volume 1265, 420-429, Dagstuhl, Germany, July 1997

[Nute 1994] Donald Nute. Defeasible Logic. In Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 3: Nonmonotonic Reasoning and Uncertain Reasoning, 353-395, Oxford University Press, 1994

[Patel-Schneider et al. 2004] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax. W3C Recommendation 10 February 2004. Technical report, W3C, 2004. http://www.w3.org/TR/owl-semantics/.

[PENG-Nutshell] PENG in a Nutshell. Macquarie University, Australia, 2 January 2003, http://www.ics.mq.edu.au/~rolfs/peng/nutshell.html

[Pratt-Hartman 2003] Ian Pratt-Hartmann. A two-variable fragment of English. Journal of Logic, Language and Information, 12(1):13–45, 2003.

[Pratt-Hartmann & Third 2006] I. Pratt-Hartmann, A. Third. More Fragments of Language. Notre Dame Journal of Formal Logic. Volume 47, Number 2. 2006

[Przymusinski 1991] Teodor C. Przymusinski. Stable Semantics for Disjunctive Programs. In New Generation Computing, volume 9, number 3/4, 401-424, 1991

[Rector et al. 2004] Alan L. Rector, Nick Drummond, Matthew Horridge, Jeremy Rogers, Holger Knublauch, Robert Stevens, Hai Wang, and Chris Wroe. OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns. In Enrico Motta, Nigel Shadbolt, Arthur Stutt, and Nicholas Gibbins, editors, Engineering Knowledge in the Age of the Semantic Web, 14th International Conference, EKAW 2004, volume 3257 of Lecture Notes in Computer Science, pages 63–81. Springer, October 5–8th 2004.

[Schwertel 2003] U. Schwertel. Plural Semantics for Natural Language Understanding: A Computa–

tional Proof-Theoretic Approach. PhD thesis. University of Zurich. 2003

[Schwitter & Tilbrook 2004] Rolf Schwitter and Marc Tilbrook. Controlled Natural Language meets the Semantic Web. In S. Wan A. Asudeh, C. Paris, editor, Australasian Language Technology Workshop 2004, pages 55–62, Macquarie University, December 2004.

[Schwitter et al. 2003] Rolf Schwitter, Anna Ljungberg, David Hood. ECOLE: A Look-ahead Editor for a Controlled Language. In Proceedings of EAMT-CLAW03, Controlled Language Translation, Dublin City University, 141-150, 2003

[Schwitter & Tilbrook 2006] Rolf Schwitter and Marc Tilbrook. Let's Talk in Description Logic via Controlled Natural Language. In Logic and Engineering of Natural Language Semantics 2006, (LENLS2006), Tokyo, Japan, June 5–6th 2006.

[Sowa 2004] John F. Sowa. Common Logic Controlled English. Technical report, 2004. Draft, 24 February 2004, http://www.jfsowa.com/clce/specs.htm.

[Syrjanen 2000] Tommi Syrjänen. Lparse 1.0 User's Manual. 2000. http://www.tcs.hut.fi/Software/smodels/lparse.ps

[Thompson et al. 2005] Craig W. Thompson, Paul Pazandak, Harry R. Tennant. Talk to Your Semantic Web. In IEEE Internet Computing, 9(6):75-79, 2005