Contents

1 Software used

The mathematical nature of description logics has meant that domain experts find them hard to understand. This forms a significant impediment to the creation and adoption of ontologies.

1.1 ROO and Rabbit[1]

ROO is a …..

Rabbit is a Controlled Natural Language that can be translated into OWL with the aim of achieving both comprehension by domain experts and computational preciseness. We see Rabbit as complementary to OWL, extending its reach to those who need to author and understand domain ontologies but for whom descriptions logics are difficult to comprehend even when expressed in more user-friendly forms such as the Manchester Syntax.

The fundamental principles underlying the design of Rabbit are:

- To allow the domain expert, with the aid of a knowledge engineer and tool support, to express their knowledge as easily and simply as possible and in as much detail as necessary.
- To have a well defined grammar and be sufficiently formal to enable those aspects that can be expressed as OWL to be systematically translatable.
- To be comprehensible by domain experts with little or no knowledge of Rabbit.

[1] refs: Denaux, Intuitive ontology authoring using controlled natural language and Hart et al, Rabbit: Developing a Control Natural Language for Authoring Ontologies

- To be independent of any specific domain.

The original intention was for Rabbit to enable domain experts alone to author ontologies. However, practice showed that whilst domain experts could build ontologies, these ontologies often contained many modelling errors not related to the language but the modelling processes. None-the-less Rabbit still enables the domain expert to take the lead and to author ontologies with guidance in modelling techniques from a knowledge engineer. Rabbit also enables other domain experts to verify the ontology.

Rabbit contains three main types of language element. Those used to

- express axioms
- introduce (or declare) concepts and relationships
- import or reference other Rabbit ontologies.

2 Appendix

2.1 ACE syntax cheat sheet

2.2 Writing OWL ontologies in ACE

2.2.1 Introduction

This document describes how to write OWL ontologies in Attempto Controlled English (ACE).

To begin, here are some simple statements that are usually found in ontologies. The ACE representation is contrasted with a "formal language" representation that is normally used in such ontologies. Our work intends to show that natural language, if carefully controlled, provides a more readable and writable ontology representation, without a loss in generality and expressive power.

ACE	Formal language
Every man is a human.	man $\subseteq$ human
Every human is a male or is a female.	human $\subseteq$ male $\cup$ female
John is a student and Mary is a student.	{John, Mary} $\subseteq$ student
No dog is a cat.	dog $\subseteq \neg$ cat
Every driver owns a car.	driver $\subseteq \exists$ own car
Everything that a goat eats is some grass.	goat $\subseteq \forall$ eat grass
John likes Mary.	<John, Mary> $\in$ like
Everybody who loves somebody likes him/her.	love(X, Y) $\Longrightarrow$ like(X, Y)

Not all ACE constructions map nicely to OWL. Therefore, we describe only a subset of ACE. This subset does not contain adjectives, adverbs and prepositional phrases; intransitive and ditransitive verbs; queries and imperative constructions; modality and sentence subordination; and negation as failure.

The following OWL constructions are currently not supported by the ACE-to-OWL mapping.

- All meta elements, such as imports, versioning, annotations, etc.
- IRIs and namespaces. One cannot refer to concepts already (partly) defined in other ontologies.

- Data-valued properties are only partly supported, e.g. one can use only integers and strings as data values, and cannot define the domain and range of data-valued properties.

Note that some OWL elements will never be supported, because they are just syntactic sugar and have furthermore no clear ACE counterparts.

The following text is structured to separately discuss the three main building blocks of ontologies as used in OWL: individuals, classes and properties.

2.2.2 Individuals

One can talk about individuals, classify them and describe relations between them. For example,

- There is a man. The man is John.
- His age is 32 and he likes Mary.
- Mary loves herself.
- Bill sees something.
- John is not Bill.

The following graph visualizes the model described by the story, with individuals (denoted by ellipses), their belonging to classes (boxes with labels), properties (solid arrows with labels), equality (dashed line) and difference (red line). Data-values are in a box with dotted border. In case the name of an individual is know then it is written into the ellips.

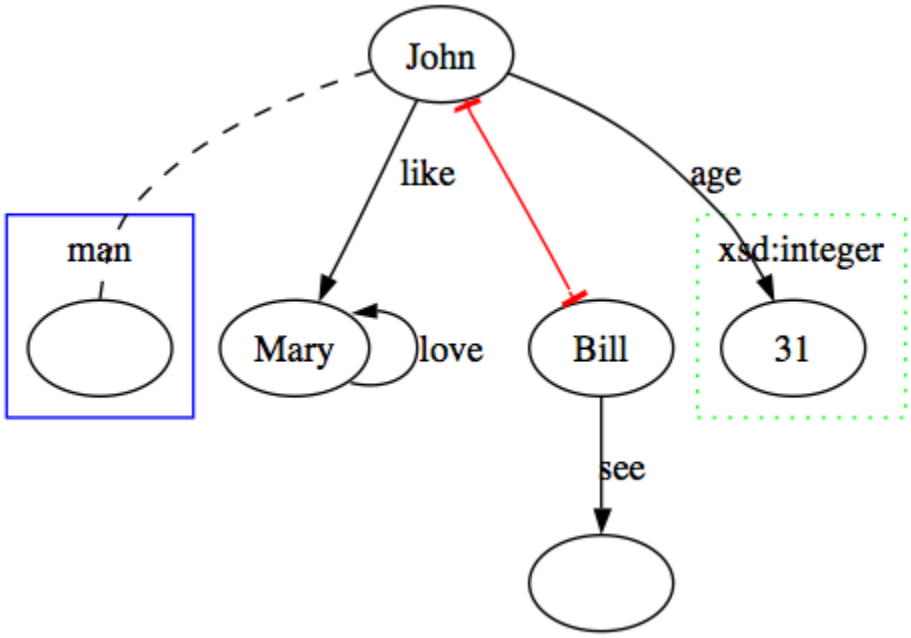

Figure 1: Picture. Representation of individuals as a Venn diagram.

OWL does not make a unique name assumption, i.e. an individual can have different names and one has to explicitly state the equivalence or difference of two differently named individuals. Therefore, whether Mary is John or is Bill or is different from both of them, is still open, we only know that John is not Bill. The fact that John belongs to the class *man* is not explicit in the picture, but can be derived by simple reasoning. Note also that John, Mary, Bill and the thing that Bill sees, all belong to the topmost class *owl:Thing*.

One can also classify the individuals by complex class descriptions, e.g.

- John is not a woman. (i.e. *John* belongs to the complement of the class *woman*)
- John is a programmer or is a manager. (i.e. *John* belongs to the union of classes *programmer* and *manager*)

2.2.3 Classes

One can describe classes by specifying relations to other classes. Such relations can be a simple inheritance from a named class ("Every man is a human.") or more complex property restrictions ("Every driver owns a car."). In OWL, the classes do not need names, they can be abstract descriptions, e.g. "(Every) man who owns a dog …". Such descriptions can be made arbitrarily complex by negation (*no*, *is not*, *does not*), disjunction (*or*, *or that*) and conjunction (*and*, *that*, *and that*).

Both the *every*-sentences ("Every man who has a driving-license drives a car.") and the *if-then* sentences ("If a man has a driving-license then he drives a car.") can be used. The *if-then* sentences are more flexible in the way the verbs can refer to existing nouns via anaphoric references (pronouns, variables, and definite noun phrases). However, it is often the case that OWL cannot express this kind of argument sharing between verbs. Therefore, it is a good idea to only use *every*-sentences and avoid all kinds of anaphoric references to nouns occurring in the same sentence.

- Every man is a human. (man $\subseteq$ human)
- If there is a man then the man is a human. (man $\subseteq$ human)
- Every professor is a scientist and is a teacher. (professor $\subseteq$ scientist $\cap$ teacher)
- Everybody who is not a child is an adult. (owl:Thing $\cap \neg$ child $\subseteq$ adult)

Equality of classes has to be spelled out, there is no shorthand construction like OWL's *EquivalentClasses*.

- Every carnivore is a meat-eater and every meat-eater is a carnivore.

Disjointness of classes can be expressed via ACE's negation which maps to OWL's *ComplementOf*.

- Every man is not a woman. (man $\subseteq \neg$ woman)
- No man is a woman. (man $\subseteq \neg$ woman)
- Nothing is a man and is a woman. (owl:Thing $\subseteq \neg$ (man $\cap$ woman))

Transitive verbs and adjectives introduce the *SomeValuesFrom* restriction.

- Every man who knows a publisher writes a book. (man $\cap \exists$ know publisher $\subseteq \exists$ write book)
- Every man likes a dog or likes a cat. (man $\subseteq$ ($\exists$ like dog) $\cup$ ($\exists$ like cat))
- If there is a woman then she does not like a snake. (woman $\subseteq \neg \exists$ like snake)

OWL's *AllValuesFrom* constraint e.g. that carnivores eat only meat (if they eat at all), can be expressed by double negation. Let's say we want to express that carnivores and meat-eaters are equivalent concepts (i.e. that carnivore = $\forall$ eat meat).

- No carnivore eats something that is not a meat.
- Everybody that doesn't eat something that is not a meat is a carnivore.

There is also a shorter ACES syntax for some patterns of *AllValuesFrom* (i.e. where *AllValuesFrom* is on the right side of the subsumption sign).

- Everything that a carnivore eats is a meat. ($\exists$ inv(eat) carnivore $\subseteq$ meat)
- Everything that is eaten by a carnivore is a meat. # the same using passive

Classes can also be described in terms of a property and its cardinality. (Note that the plural of 'thing' will be mapped to *owl:Thing*.)

- Every woman likes more than 2 things. (woman $\subseteq$ >= 3 like owl:Thing)
- Every millionaire owns at least 2 cars. (millionaire $\subseteq$ >= 2 own car)

Reflexive pronouns ('itself', 'himself', 'herself') can be used to describe (ir)reflexivity.

- Every narcissist likes himself.
- No river flows-into itself.

Classes can also be described in terms of a property and its value that is either an individual or data-value. This corresponds to OWL's *HasValue* construct. For individual-valued properties, one must use a transitive verb (or adjective) with an object that is a proper name or a reference to an individual that has already been introduced.

- Every man knows John. (man $\subseteq \exists$ know {John})
- There is a woman who everybody likes. (w $\in$ woman; owl:Thing $\subseteq \exists$ like {w})
- John likes every dog. (dog $\subseteq \exists$ inv(like) {John})
- Every dog is liked by John. # the same using passive

For data-valued properties, ACE makes use of the copula verb 'is', a genitive construction (Saxon Genitive, possessive pronoun or an *of*-construct), a noun which expresses the property, and finally a number or a string. For example

- Everybody whose age is 32 is a grown-up.
- If there is a man who lives-in Paris then the man's address is "Paris".
- For all water it is false that the temperature of the water is -1.

where the genitive construction is expressed by 'whose', 'man's', 'of the water', the property is expressed by 'age', 'address', 'temperature', and the data-value is '32', "'Paris"'and '-1'.

2.2.4 Properties

OWL properties correspond to ACE's verbs and transitive adjectives (e.g. 'taller than').

A superproperty for a given property can be defined as:

- Everybody who loves somebody likes him/her.

or in a more verbose manner:

- If there is somebody X and X loves somebody Y then X likes Y.

Disjointness of properties can be declared by using negation.

- Nobody who likes somebody hates him/her.

or more verbosely:

- If somebody X likes somebody Y then it is false that X hates Y.

Property chaining is a complex mathematical concept that does not have a simple verbalization in natural language. E.g. the transitivity of a property is formulated in ACE in the following ways (different levels of syntactic compactness are provided):

- If something X is taller than something Y and Y is taller than something Z then X is taller than Z.
- If something X is taller than something that is taller than something Y then X is taller than Y.

To declare an inverse property of a property, two sentences are needed.

- If something X is taller than something Y then Y is shorter than X.
- If something X is shorter than something Y then Y is taller than X.

The rest of the syntax that OWL provides for describing properties is just syntactic sugar, one can use the already described class relations, cardinality constraints and inverse properties instead.

Properties can have a domain and a range. The domain of a property can be expressed e.g. by

- Everybody who writes something is a human.

The range of a property can be expressed by:

- Everything that somebody writes is a book or is a paper.
- Everything that is written by somebody is a book or is a paper. # the same using passive

Properties can be functional and inverse functional. Functionality means that for a given subject, the object of the property (verb) is always the same. Inverse functionality means that for a given object the subject must always be the same. For functionality, one has to restrict the class *owl:Thing* to have at most one value for a given property (it is OK to have no values at all).

- Everybody orders at most one thing. (owl:Thing $\subseteq$ <= 1 order owl:Thing)

Inverse functional properties are expressed by switching the subject and the object of the verb.

- Everything is something that at most one thing orders. (owl:Thing $\subseteq$ <= 1 inv(order) owl:Thing)
- Everything is ordered by at most one thing. # using passive

Symmetric property is its own inverse. The various ACE syntaxes to express this are:

- If somebody X loves somebody Y then Y loves X.
- Everybody who is loved by somebody loves him/her.
- Everybody who somebody loves loves him/her.

Equivalent properties are expressed by declaring the two properties to be superproperties of each other.

- Everybody who hates somebody despises him/her.
- Everybody who despises somebody hates him/her.

2.2.5 Reasoning

User acceptance of the reasoning results obtained on the OWL level and perceived on the ACE level provides the basis for proving that the mapping from ACE to OWL is useful. Given the ACE text

- John is a man.
- No man is a woman.
- John is a woman.

the reasoner will find an inconsistency because John has been asserted as a man and men have been described as disjoint from women.

Here is another example.

- Every child is a man or is a woman.
- No child is a man.
- No child is a woman.

This text may seem strange, but it is not inconsistent, as far as OWL is concerned. All we have done, is constructed an empty class *child*, which is perfectly legal (one can say, though, that the class *child* is unsatisfiable). Only when we claim that there are elements (individuals) in this class, for instance by stating

- John sees a child.

do we end up being inconsistent.

Reasoning about property restrictions finds inconsistency in:

- Every man sees a dog.
- Nothing is a dog.
- John is a man.

Reasoning about properties of properties. The following text is consistent.

- No dog likes a cat.
- Fido is a dog.
- It loves a cat.

But it becomes inconsistent if we add information about the relation between 'like' and 'love':

- Everything that loves something likes it.

We can also reason about the domain and range of properties.

- Everything that somebody writes is a book.
- Everybody that writes something is a human.
- John who is a man writes a paper.
- Every man is a human.

Inconsistency is caused when we now claim that:

- No paper is a book.

Reasoning about cardinality, e.g. functional properties. Imagine a library policy saying that everybody can order only one item from the library, and that John places two orders.

- Everybody orders at most one thing.
- John orders a book X and orders a book Y.

This text is consistent, namely because the reasoner sees nothing strange about taking X and Y to denote the same item (individual). Only when we state that:

- X is not Y.

would it become inconsistent.

Reasoning about inverse functional properties is similar.

- Everything is ordered by at most one thing.
- Mary orders a book X.
- Bill orders the book X.
- Mary is not Bill.

Reasoning about transitivity and *HasValue* will find the following text inconsistent.

- If something X follows something that follows something Y then X follows Y.
- John follows Mary who follows Bill who follows John.
- No manager follows Mary.
- Bill is a manager.

Reasoning about inverse properties. In ACE, in many cases, one does not have to explicitly assert a named inverse property of a given property. One can simply use a passive verb to create an (unnamed) inverse property. An inconsistent text, containing an inverse property, can be constructed like this:

- John likes Mary.
- Nobody is liked by John.

This statement would be redundant:

- If somebody X likes somebody Y then Y is liked by X.

Reasoning about nominals (i.e. OWL's *OneOf* construct). The following text is inconsistent.

- Every student is somebody who is John or who is Mary.
- Bill is a student who is not John and who is not Mary.

2.3 Roo Language Elements

2.3.1 Declarations

Rabbit allows an author to explicitly declare **concepts, relationships** and **instances**.

Concepts are introduced using the form:

> is a concept [, plural].

for example:

> River is a concept. (plural defaults to Rivers.)
>
> Sheep is a concept, plural Sheep.

Homonyms are dealt with using brackets. For example a Pool can either be a pool of water on land or a pool of water within a river. These are quite different physical phenomena. The former is more common and so we would introduce it as:

> Pool is a concept.

whereas the later would be introduced as:

> Pool (River) is a concept.

Where it is unclear which was more common both would use the bracketed form.

In Owl these statements translate to: River -> Thing with the annotation "River" as a label. (Our notation here uses -> to indicate subclass.) Homonyms will all share the same annotation rdf:label so Pool and Pool (River) will both be annotated as "Pool" but the class names will be Pool and Pool_River respectively.

Relationships and instances are introduced in a similar way:

> is a relationship.
>
> is a .

For example:

> next to is a relationship.
>
> Derwent Water is a Lake.

2.3.1.1 Concept Descriptions, Definitions and Axioms A concept is described (necessary conditions) or defined (necessary and sufficient conditions) by a collection of axioms relating to that concept. Each axiom is represented by a single Rabbit sentence. The simplest sentence form is where one concept it associated with another through a simple relationship.

For example:

> Every is a kind of . (Subsumption.)
>
> Every .

E.g.

> Every House is a kind of Building.
>
> Every River contains Water.

Such statements translate to OWL as follows:

> House -> Building.
>
> River -> contains some Water.

A number of modifiers exist to enable cardinality and lists to be supported.

For example:

> Every River Stretch has part at most two Confluences.
>
> Every River flows into exactly one of River, Lake or Sea.

Concept descriptions comprise a series of axioms relating to the same concept. A definition comprises a group of axioms that make up the necessary and sufficient conditions. The axioms are grouped by an introductory statement and then follow as a list:

A School is defined as:

> Every School is a kind of Place;
>
> Every School has purpose Education;
>
> Every School has part a Building that has purpose Education.

Which translates into OWL as:

> School Place and (hasPurpose some Education) and (hasPart some (Building and hasPurpose some Education))

Rabbit at present contains a sentence form that cannot be translated into OWL, although its omission does not invalidate the OWL ontology it does make the OWL representation less complete. This is the ability of Rabbit to modify an axiom by adding “usually”. For example:

A Postal Address usually contains a Road Name.

As OWL is unable to express the existential quantifier for elements on the left hand side of an expression, this sentence cannot be converted to OWL. The reason for including it is to enable the domain expert to record frequent but not mandatory relationships, the absence of which would make certain definitions seem strange. Indeed without the use of "Usually" the only things that could be said about a British Postal address are:

A Postal Address contains exactly 1 Post Town Name.

A Postal Address contains exactly 1 Postcode.

2.3.2 Intersection and Union

Rabbit implements intersection in a number of different ways in order to promote the development of short sentences, rather than encourage the authoring of long sentence structures that could become hard to understand. Most common amongst these mechanisms is that all Rabbit sentences that refer to the same concept are converted to OWL as one long conjunction comprising all the Rabbit sentences. Within a sentence, "and" is supported, but in practice has rarely been used. Rabbit also supports "that" which is interpreted exactly the same as "and".

For example:

Every Almshouse has part a Building that has purpose Housing of Elderly People.

Here we encourage the use of "that" rather than "and" as it both sounds better, and we also believe it will discourage long chains of conjunctions that would be better treated as separate Rabbit sentences.

Another mechanism used to implement intersection is the use of "of", "for" and "by" (again all semantically equivalent in their OWL translation). They are used in the sentences of the structure:

Every [of | for | by] .

e.g.

Every School has purpose Education of Children.

this translates into OWL as:

School -> hasPurpose some (Education and appliesTo some Child)

Here "of", "for" and "by" are translated to "and appliesTo" in OWL with appliesTo being a predefined Rabbit relationship.

"Or" is supported in both inclusive and exclusion forms. By default Rabbit treats "or" (and ",") as exclusive. So:

Every River flows into a Sea or Lake.

Is interpreted in OWL as:

River -> flowsInto some ((River or Lake) and not (River and Lake)).

However, we found that the exclusive or was very rarely used in ontology modelling, and where it did appear, usually indicated some mistake in the content of our model. In the above example, we would more accurately designate "flows into" as a functional property, and hence not need to consider the exclusive or.

Inclusive Or is implemented through the use of the modifier "One or more of":

Every Mission has purpose one or more of Christian Worship or Charitable Activities.

which in OWL is:

Mission -> hasPurpose some (ChristianWorship or CharitableActivity).

2.3.3 Ontology Referencing

No ontology can be an island unto itself: the point of ontologies is interoperability, and therefore they need mechanisms to include concepts and relationships from other ontologies.

OWL achieves this through the owl:import statement, although since it operates at the level of the whole ontology, it is a fairly crude approach.

The equivalent Rabbit construct is:

Use ontologies: from ; … from .

e.g.

Use ontologies:

Wildfowl from http://www.ordnancesurvey.co.uk/ontology/v1/Wildfowl.rbt

Transport from http://www.ordnancesurvey.co.uk/ontology/v1/Transport.rbt

Concepts and relationships taken from these ontologies are then identified using the notation:

[]

for example

Every Duck Pond is a kind of Pond.

Every Duck Pond contains Ducks [Wildfowl].

This indicates that the concept Duck is obtained from the Wildfowl ontology. Since repeatedly referencing Ducks [Wildfowl] can be a bit cumbersome, Rabbit also enables a local name to be applied:

Refer to Duck [Wildfowl] as Duck.

This produces the following OWL:

?Duck -> Thing

Duck))

As this creates a new local concept, it is then possible to extend the definition of the imported concept if required by adding further axioms to the local concept.

2.3.4 Property Restrictions - An Area of Weakness

All CNLs appear to struggle when it comes to implementing property characteristics. such as symmetry or transitivity. Fundamentally this is because these constructs are not well aligned to the way that people normally think about things and hence there are no easily understood natural language analogues. Probably the worst example is transitivity. In Rabbit such characteristics are defined as follows:

> The relationship is transitive.

e.g.

The relationship "is part of" is transitive.

However as discussed below, human subject testing has shown that this is very poorly understood by people, as are similar constructions. ACE and SOS have both taken a different approach. They attempt to define such relationships through example. SOS express the transitive relationship as follows:

> If X Y and Y Z then X Z.

> If X is part of Y and Y is part of Z then X is part of Z.

It may well be that such expressions are more understandable and we are planning to include such sentences in the next phase of human subject testing. However, we strongly suspect that such structures will still present domain experts with significant problems with interpretation, and take much longer to input when faced with authoring a large ontology. We are prepared to admit that there is no good solution within the language and so will have to be managed through training, tool support and guidance by the knowledge engineer.

Summary