

# A METHODOLOGY FOR CONVERTING CONCEPTUAL ONTOLOGIES TO OWL



## Document Information

This document has 12 page(s)

This document is intended to enable you to convert “conceptual ontologies” [Kovacs et al 2006] to logical ontologies encoded in OWL (the Web Ontology Language) [W3C – OWL].

The author is John Goodwin

Ordnance Survey and the OS Symbol are registered trademarks of Ordnance Survey, the national mapping agency of Great Britain.

## Table of Contents

1	Introduction .....	2
2	Step 1 – Set up the name spaces .....	3
3	Step 2 – Locate required ontology modules .....	3
4	Step 3 – Naming Convention .....	4
5	Step 4 – Add the Classes .....	4
6	Step 5 – Add the Properties .....	6
7	Step 6 – Add domain and range constraints for properties .....	7
8	Step 7 – Add Instances .....	7
9	Step 9 – Annotate the ontology .....	8
10	Step 10 – Run the Reasoner .....	8
11	Step 11 – Define all defined classes.....	8
12	Step 12 – Describe all classes .....	9
13	Step 13 – Run the classifier .....	9
14	Step 14 – Debug the ontology .....	9
15	Evaluating the Ontology Against the Competency Questions.....	10
16	References.....	11

## 1 Introduction

This document is intended to enable you to convert “conceptual ontologies” [Kovacs et al 2006] to logical ontologies encoded in OWL (the Web Ontology Language) [W3C – OWL]. This document is part of a collection of documents that aid the ontology development lifecycle [Hart et al], [Kovacs et al 2006], [Dolbear et al 2007] and [Hart and Goodwin 2007]. This document gives a step by step approach for populating an OWL ontology. The OWL ontology is converted from the conceptual ontology written in Rabbit, and the conversion rules can be found in [Dolbear et al 2007]. Where necessary the author should consult the accompanying modelling guidelines document [Hart and Goodwin 2007] to find out more information about some of the suggestions made in this document.

Much of this is based on Ordnance Survey’s experience in writing ontologies, the work of the W3C Semantic Web Best Practice Group [SWBP] and the work of Alan Rector [Rector] and the CO-ODE group [CO-ODE]. Readers unfamiliar with OWL basics should consult [W3C – OWL] or [CO-ODE]. Little or no reference will be made to the OWL syntax, and readers interested in the OWL syntax and how it relates to existing standards such as XML (eXtensible Markup Language) and RDF (Resource Description Framework) should consult [W3C – OWL].

All examples of OWL axioms will be written using the Manchester OWL syntax [Horridge et al.].

Furthermore, this document is not intended as a “Protégé user guide”. There are plenty of guidelines available on using Protégé at [CO-ODE].

## **2 Step 1 – Set up the name spaces**

Create a new ontology file using the ontology editor of your choice, such as Protege [Protege]. You will need a namespace for your ontology. This is basically a way to provide a unique resource identifier (URI) for the ontology, and its classes and properties so that it can be found or reference on the web. Anyone interested can easily find more information about namespace on the web by googling for “XML and namespace”.

All Ordnance Survey ontology namespaces should be of the form:

[http://www.ordnancesurvey.co.uk/ontology/vn/ontology\\_name.owl](http://www.ordnancesurvey.co.uk/ontology/vn/ontology_name.owl)

where vn is “v” followed by the version number of the ontology, e.g. v1, v2, etc...

All deprecated ontologies will be kept in their original location. This follows the usage pattern for XML Schema. See:

<http://www.ordnancesurvey.co.uk/xml/schema/index.html>

## **3 Step 2 – Locate required ontology modules**

This step of the methodology is provisional at the moment as we have so far had limited experience of this. As time goes on we hope to provide an increasingly large library of ontologies. For example we currently have ontologies for things like mereology (part-whole relationships) and topology. Our ontologies are available for download at <http://www.ordnancesurvey.co.uk/ontology>.

Once you have created a new ontology file with the correct name spaces then identify and “import” the required modules. The Rabbit ontology will tell you which modules to import based on the keywords “Refer to”. This can be done at any time, but it is useful to identify them early on to prevent “reinventing the wheel”. You may need to double check that the domain expert has identified all appropriate modules for reuse. If for example, “part-whole” relationships are specified in the conceptual ontology, then the mereology ontology can be imported. The conceptual ontology will list which conceptual ontologies have been reused, and it should be easy to identify the corresponding OWL ontologies. However, this only applies to ontology modules that Ordnance Survey has previously generated: the domain expert may need some help in identifying any externally-produced ontologies that are suitable for reuse, as to date, these are usually only written in a machine-readable form like OWL.

## 4 Step 3 – Naming Convention

OWL follows the convention of many object oriented programming languages in that classes traditionally start with an uppercase letter, and the subsequent words in a class name should start with a capital letter. For more specific naming conventions refer to [Kovacs et al 2006]. Typical class names would river "River", "BodyOfWater" etc. The most straightforward way will be to convert the Rabbit class names as set out in Annex A of [Dolbear et al 2007].

Properties start with a lowercase letter, and subsequent words in a property name begin with an uppercase letter. Examples include "flowsInto", "partOf" etc. Again conversion from Rabbit will be as specified in Annex A of [Dolbear et al 2007].

Instances are all lowercase. Subsequent words can be separated using an underscore. Examples include "england", "john\_goodwin" etc...

As OWL is an XML technology there are certain characters that are not allowed in the names of class, properties or instances. These include spaces, ampersands and the # character. The ontology editor should highlight any mistakes.

## 5 Step 4 – Add the Classes

In an OWL ontology everything is a subclass of *OWL:Thing*. However for reasons of clarity it is worth creating two top level concepts to help keep your ontology readable. Create a top level class called something like *DomainConcept*. This will house all classes relevant to the domain being created (both core and secondary).

From the conceptual ontology list all of the concept names, by locating all the sentences of the form "X is a concept". So for example, we might end up with a list like this from the conceptual ontology:

- River
  - Duck
  - Lake
  - Sea
  - Country
  - Region of some Country
  - Length
  - Long
  - Channel
  - Fish
  - Salmon
  - Region
  - Stream
  - Pond
  - Soil
  - Water
  - Stones
  - Width
  - Flow
  - Bank
- etc..

Also list the properties using all the sentences of the form "y is a relationship".

- flowsInto
  - hasLocation
  - hasLength
  - connectedTo
  - partOf
  - hasPart
- etc...

Identify which concepts are primitive and which are defined. This information should be available in the conceptual ontology. Defined concepts can be identified by the Rabbit sentences "X is uniquely defined as ...". Identify additional concepts that might be needed in the logical ontology. The conceptual ontology should have disambiguated most concept terms (e.g. using River (Bank) rather than just Bank) and these can be converted to OWL using an (e.g. River\_Bank). Any additional concepts needed should be disambiguated using the same convention, after checking with the domain expert.

Next arrange all the primitive concepts into subclass hierarchies – again most of this information should be contained in the conceptual model, based on the "is a kind of" Rabbit relationship. There needs to be an open dialogue between the domain expert and ontology engineer to ensure the hierarchy is correct both conceptually and logically. Make sure each class has at most one superclass – that is all the hierarchies should form a tree like structure. Copy all the Rabbit structured sentences into the rdf:comment of each concept you add – this acts as documentation for the concept (in OWL 1.1, each separate axiom could have its own annotation with the one relevant Rabbit sentence, but for now, all sentences relating to the concept are stored together)

Following the method used in [Rector et al.] create concepts to represent top level abstractions for the hierarchies. These concepts will all be direct children of the DomainConcept class. If this was not done correctly in the conceptual ontology, liaise with the domain expert to check which concepts should be children of the DomainConcept class.. Guidelines for creating top level abstractions can be found in section 4 of [Hart and Goodwin 2007].

You should now have you primitive classes arranged in a hierarchy with top level abstractions to distinguish each tree – or high level branch of the hierarchy. For example:

- **BodyOfWater**
  - River
  
  - Stream

- Lake
- ...
- **AdministrativeUnit**
  - Country
- **LivingThing**
  - Animal
    - Fish
      - Salmon
  - Plant
  - ...

In Rabbit we assume that all sibling primitive classes are mutually disjoint, unless otherwise stated with the relationship “can be the same as”. This needs to be stated explicitly in the OWL ontology. Now make each top level class mutually disjoint, and make each sibling class disjoint. So in the above we should now have `BodyOfWater`, `AdministrativeUnit` and `LivingThing` as disjoint concepts. Similarly `Animal` and `Plant` should be disjoint, as should `River`, `Stream`, `Lake` and so on for every sibling class. If the domain expert thinks two primitive sibling concepts should not be disjoint then ask them why. Overlapping concepts are often an indication of hidden implicit information or modelling errors [Rector et al.]. Hierarchies of primitive classes should be open as we can never hope to capture a complete list of primitive classes. When we say the hierarchy should be open we mean that, for example, `LivingThing` is not the union of `Animal` and `Plant` [Rector et al.].

Finally add the defined classes. Double check with the domain expert that they have correctly differentiated the primitive and defined classes. Defined classes should go directly under `OWL:Thing`. These should not be made disjoint with anything. The reason for this is that we might have created a defined class “tidal water body” and another “salt water body”. The concept `Sea` will be classified as both “tidal water body” and “salt water body”, and as such these two classes are not disjoint.

A more detail rationale on why we do this can be found in section 4 of [Hart and Goodwin 2007]

## 6 Step 5 – Add the Properties

Now add your properties and arrange property hierarchies as specified in the Rabbit document. Document the meaning of each property from the glossary in the conceptual ontology, by copying the relevant sentences into the `rdf:comment` of the property. Place constraints on each property indicating where relevant which properties are transitive, symmetric, functional and inverse functional,

based on the Rabbit sentences. Note that a property cannot be both transitive and functional in OWL DL. Where a property is transitive construct a non-transitive subproperty of that property. For example, if the domain expert has made *hasPart* is transitive, then we need to create a subproperty of *hasPart* called *hasDirectPart*. Now think carefully whether you need to use the transitive property or its non-transitive subproperty. Guidance on this is provided in the section 4.6 of [Hart and Goodwin 2007]

Now where applicable record the inverses of each property. Inverse properties can be computationally expensive so double check with the domain expert on which ones are really needed.

## 7 Step 6 – Add domain and range constraints for properties

Now add the domain and range constraint on properties where applicable. These may have been identified in the Rabbit sentences with constructs like “has address can only have an Address as an object”, however, this can be difficult for the domain expert, so it is worth checking that they have been used correctly. Note that domain and range places global constraints on that property (that is, it applies to the property whichever concepts it’s used to relate) so use with care. Generally speaking the classes found in a domain and range constraint will be fairly high level classes. For example, the range of the property *participatesIn* will most likely be the class *Process*. It is probably not necessary to add a domain constraint for this property as a number of things can participate in processes. Domain and range constraints can be used to enforce the correct usage of a property in an ontology – though they should be used with caution on classes further down the tree structure as they are a likely cause of inconsistencies in the ontology.

## 8 Step 7 – Add Instances

In a domain ontology there should not be too many instances, but occasionally they do occur. Examples of these could be countries (such as England, Scotland and Wales etc.), days of the week (Sunday etc.) or seasons (winter etc.).

In some cases a class will be defined exactly by its members. For example the class *DayOfTheWeek* could be defined precisely by the days Sunday, Monday, Tuesday, Wednesday, Thursday, Friday and Saturday. This would be written in the Rabbit structured sentences as: “Only Sunday Monday Tuesday Wednesday Thursday Friday and Saturday can be a day of the week”. This can be represented using the OWL “oneOf” construct as follows

*DayOfTheWeek* = {sunday, monday, tuesday, wednesday, thursday, friday, saturday}

## 9 Step 9 – Annotate the ontology

Each concept and relationship has already been assigned some documentation from the Rabbit sentences in the `rdf:comments` as we have gone along. Now add `rdf:labels` for your classes and “has name” datatype (not annotation) properties for your instances. We recommend that every class has at least one label (this may just be the same as the class name, e.g. “river” or it may be the class name with spaces insert between each word, e.g. “body of water”). If the class name is something like `Pool_In_River`, and then we need two labels, one giving its commonly used name in the domain, i.e. “Pool”, and the other its class name with spaces i.e. “Pool in river”. See Annex A of [Dolbear et al 2007] for the conversion table. This step is important because the class identifier may not be what a user would naturally call a class and annotations aid the readability of the ontology when used in applications. In the future it may also be useful for searching applications. Some classes will also have a synonym, which again is an annotation property. An instance will have a unique name, for example `london_gor`, but the name commonly used for that instance can be stored as a string, for example, `london_gor hasName "London"`.

## 10 Step 10 - Run the Reasoner

At this stage it is worth running the ontology through the inference engine to make sure the range and domain constraints (if used) are not creating any undesirable inferences in the ontology.

## 11 Step 11 – Define all defined classes

Use Annex A in the Rabbit document [Dolbear et al 2007] to see how to translate the domain expert’s structured sentences to OWL. If you are unsure about anything then ask the domain expert to clarify what his or her intention is.

Now add the necessary and sufficient conditions that are required to define all defined classes. These are the sentences within the “A concept is uniquely defined as: ...” structure. For example, from the sentence “A Fresh Body of Water is uniquely defined as: A Fresh Body of Water is a kind of Body of Water; A Fresh Body of Water is made of Fresh Water.” we would add the information:

`FreshBodyOfWater = BodyOfWater AND madeOf some FreshWater`

If transitive properties like `hasPart` are being used in the definition of defined classes, be sure to use the full transitive property `hasPart` and not the non-transitive subproperty `hasDirectPart`. This ensures all possible inferences are picked up when the ontology is classified.

Now add any necessary conditions for the defined classes (These are other sentences involving the class as the subject, which aren't part of the "uniquely defined as" structure.)

## 12 Step 12 – Describe all classes

Now convert the structured sentences into necessary conditions for the primitive classes.

Use the Rabbit document [Dolbear et al 2007] to see how to translate the domain expert's structured sentences to OWL. If you are unsure about anything then ask the domain expert to clarify what his or her intention is.

If transitive properties like *hasPart* are being used in the description of primitive classes, be sure to use the full non-transitive subproperty *hasDirectPart* and not the transitive subproperty *hasPart*.

## 13 Step 13 – Run the classifier

Run the classifier to check for unsatisfiable classes and inconsistencies.

## 14 Step 14 – Debug the ontology

Note that some ontology tools (such as Protégé [Protege]) have built in debuggers. This area of technology is still in its infancy. However, they are still useful and worth using. If you do not have access to ontology debugging software then read on...

If you have inconsistent classes check for the following:

- 1) Had you made any defined classes disjoint?
- 2) Do the global domain and range constraints on properties conflict with local property restrictions constructed using those properties.
- 3) Do you have axioms of the form

$$A \rightarrow p \text{ some } C \text{ and } p \text{ some } D$$

Where *p* is a functional property, and *C* and *D* are disjoint classes?

- 4) Have you created conflicts using "someValuesFrom" and "allValuesFrom" restrictions on a class? Do you have axioms like:

$A \rightarrow p \text{ some } C \text{ and } p \text{ only } D$

where  $C$  and  $D$  are disjoint classes?

5) Have you created conflicting information in the hierarchy using invalid combinations of “allValuesFrom” and “someValuesFrom”? Do you have axioms like:

$A \rightarrow p \text{ only } D$

$B \rightarrow p \text{ some } C$

where  $B$  is a subclass of  $A$ , and  $C$  and  $D$  are disjoint.

Note this is not a complete list of things that can cause inconsistency, merely easily identifiable common mistakes.

Now check the inferred subsumption hierarchy. Check to see if all the intended subsumptions are present and to check for any unexpected subsumptions. Some inferences can be counter intuitive, and any unexpected subsumptions may indicate modelling errors in the ontology.

Once you have a consistent ontology, and all expected subsumptions are found on using the classifier you have completed your ontology.

## 15 Evaluating the Ontology Against the Competency Questions

In the absence of a query language for ontologies one way to check that your ontology correctly answers the competency questions is to use probe classes.

Create a subclass directly under `OWL:Thing` and called it `ProbeClass`. Do not make `ProbeClass` disjoint with `DomainConcept`. Any probe classes should now be created as subclasses of `ProbeClass`.

Probe classes are a useful way to “query the ontology”. The sorts of queries we can answer are those of the form “find me all things that...” (e.g. find me all things that flow into a sea, find me all thing that a part of a country, find me all things that have exactly 2 legs, etc...). Queries can be converted to OWL much in the same way that structured sentences in Rabbit can. A Rabbit sentence for a river might look something like

A River flows into a Sea

and this gets converted to

River → *flowsInto* some Sea

Similarly convert the query “find me all things that flow into a sea” by creating a defined class as follows:

Probe1 = *flowsInto* some Sea

Run the reasoner and check all the inferred subclasses of Probe1. All subclasses of Probe1 will be the classes that describe things that flow into a sea. In other words the answer to the question “find me all things that flow into a sea” will be the subclasses of Probe1.

This can be generalised for other queries. A useful set of queries to test would be the competency question list. Remove any probe classes before publishing the ontology.

## 16 References

[W3C – OWL] – OWL Documentation <http://www.w3.org/2004/OWL/>

[SWBP] – Semantic Web Best Practice Group  
<http://www.w3.org/2001/sw/BestPractices/>

[Rector] – Alan Rector’s Homepage <http://www.cs.man.ac.uk/~rector/>

[CO-ODE] – CO-ODE group <http://www.co-ode.org/>

[Rector et al.] – Rector, A., Modularisation of Domain Ontologies Implemented in Description Logics and related formalisms including OWL. in Knowledge Capture 2003, (Sanibel Island, FL, 2003), ACM, 121–128  
<http://www.cs.man.ac.uk/%7Erector/papers/rector-modularisation-kcap-2003-distrib.pdf>

[Welty et al.] – Welty, C. and Guarino, N. Supporting ontological analysis of taxonomic relationships. Data and Knowledge Engineering, 39 (1). 51–74.  
<http://www.loa-cnr.it/Papers/dke2001.pdf>

[Protege] – Protégé <http://protege.stanford.edu/>

[Horridge et al.] – Manchester OWL syntax [http://owl-workshop.man.ac.uk/acceptedLong/submission\\_9.pdf](http://owl-workshop.man.ac.uk/acceptedLong/submission_9.pdf)

[Dolbear et al 2007] Dolbear, C., Hart, G., Goodwin, J., Zhou, S. and Kovacs, K. The Rabbit language: description, syntax and conversion to OWL. Ordnance Survey Research Labs Technical Report IRI-0004 2007.

[Hart and Goodwin 2007] Goodwin, J. and Hart, G. Modelling Guidelines for constructing domain ontologies Ordnance Survey Technical Report IRI-0006 2007

[Kovacs et al 2006] A Methodology for Building Conceptual Domain Ontologies Katalin Kovacs, Catherine Dolbear, Glen Hart, John Goodwin and Hayley Mizen Ordnance Survey Research Labs Technical Report IRI-0002 2006

[Hart et al] Domain Ontology Development Glen Hart, Cathy Dolbear, John Goodwin, Katalin Kovacs, Ordnance Survey Research Labs Technical Report IRI-0003