The Aberdeen University Ontology Reuse Stack

Edward Thomas, Derek Sleeman, Jeff Z. Pan, Quentin Reul and **Joey Lam**
Department of Computing Science
University of Aberdeen
Aberdeen, AB24 3FX
Contact emails: {d.sleeman,jeff.z.pan}@abdn.ac.uk

Abstract

This paper describes a set of tools which allow the large number of ontologies available on the Semantic Web to be discovered and reused for other applications (both in the Semantic Web and by the larger knowledge engineering community). Firstly, these tools address the problem of finding an existing ontology, with given characteristics. Secondly, we discuss tools which allow the knowledge engineer to detect and repair errors which occur in the ontology's vocabulary (CleOn) or are related to logical coherency/inconsistency (RepairTab). Once a knowledge engineer is satisfied that the ontology is lexically and semantically consistent, then he/she can extend the ontology if the application requires that. Once the extension has been done, it again would be prudent to use those same tools to check that the resulting ontology is consistent.

Introduction

This paper describes a set of tools which allow the large number of ontologies available on the Semantic Web to be discovered and reused for other applications (both in the Semantic Web and by the larger knowledge engineering community). By using these tools, instead of creating their own ontologies from scratch, people could reuse (parts of) existing relevant ones.

Firstly, these tools address the problem of finding an existing ontology, with given characteristics. Secondly, we discuss tools which allow the knowledge engineer to detect and repair errors which occur in the ontology's vocabulary (CleOn) or logical coherency/inconsistency(RepairTab). Once a knowledge engineer is satisfied that the ontology is lexically and semantically consistent, then he/she can extend the ontology if the application requires that. Once the extension has been done, it again would be prudent to use those same tools to check that the resulting ontology is consistent.

ONTOSEARCH2 (Pan, Thomas, & Sleeman 2006) is a tool designed to allow end users of ontologies (knowledge engineers, domain experts, or software agents) to discover appropriate ontologies on the Semantic Web. It maintains a large repository of ontologies which have been found from various sources, and allows structured queries to be performed on individual ontologies, or the entire repository using SPARQL or a simplified keyword based interface.

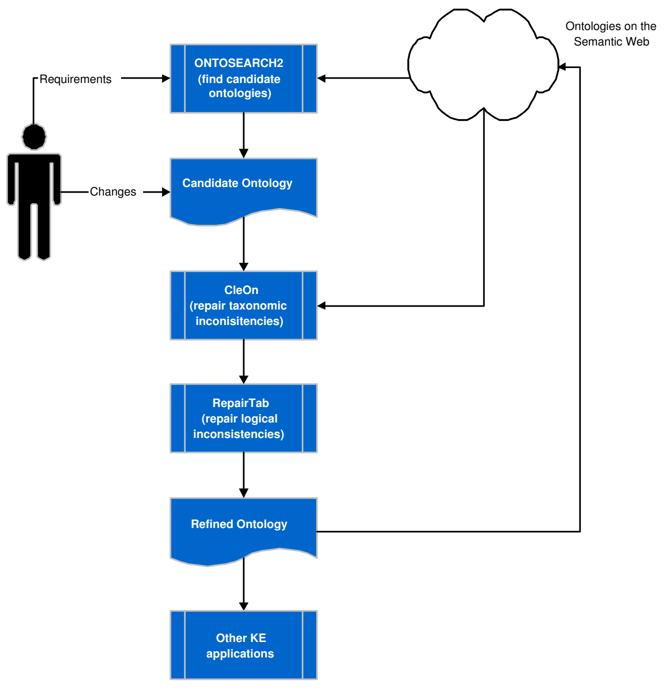

Ontologies on the Semantic Web
Requirements
ONTOSEARCH2 (find candidate ontologies)
Changes
Candidate Ontology
CleOn (repair taxonomic inconisitencies)
RepairTab (repair logical inconsistencies)
Refined Ontology
Other KE applications

Figure 1: The ontology toolkit

ONTOSEARCH2 creates a semantic approximation (Pan & Thomas 2007) of an OWL DL ontology in the tractable language DL-Lite. DL-Lite has a characteristic O_{log} performance for conjunctive queries, and because semantic approximation guarantees that no incorrect axioms are present in the approximation with respect to the original ontology, ONTOSEARCH2 is sound for all queries, further it is both sound and complete for all queries which do not include non-distinguished variables. This approach allows us to perform structural queries over very large repositories while still maintaining good scalability and performance.

The CleOn system[1] (Sleeman & Reul 2006) semi-

[1]The approach was formerly known as CleanOnto, but was renamed to avoid unnecessary confusion with OntoClean

automatically creates a lexically coherent ontology, when provided with an OWL ontology, and the corresponding source of taxonomic information. Currently, we assess taxonomic relationships by processing the linguistic information inherent in concept labels. This is achieved by assessing the lexical adequacy of a link between a parent and a child concept based on their lexical paths. To date these lexical paths have been extracted from sources such as WordNet 2.0 and a mechanical engineering thesaurus.

We have implemented a graph-based approach implemented in ReTAX++ (Lam, Sleeman, & Vasconcelos 2005) to help knowledge engineers browse ontologies and resolve logical inconsistencies. In fact, this has been implemented as the RepairTab plug-in for Protégé. With the help of a reasoner, the system detects and highlights the inconsistent concepts. We have implemented graph based algorithms to detect which relationships among concepts cause the inconsistencies, and provide options for the user to correct them. If an incomplete or inconsistent ontology is imported, a number of ontological fragments may still be formed. In this case, we aim to suggest to the user the best concept candidate to integrate the several fragments.

How these three tools interoperate to find and refine ontologies is illustrated in Figure 1. More details of these three systems are given in the following three sections, and the paper finishes with an overall conclusion.

ONTOSEARCH2[2]

Searching for relevant ontologies is one of the key tasks to enable ontology reuse. Now the W3C ontology language OWL has become the defacto standard for ontologies and semantic data, there are progressively more ontology libraries online, such as the DAML Ontology Library and Protégé Ontologies Library. While the Web makes an increasing number of ontologies widely available for applications, how to discover ontologies becomes a more challenging issue.

Currently it is difficult to find ontologies suitable for a particular purpose. Semantic Web search engines like Swoogle (Ding *et al.* 2005) and OntoKhoj (Patel *et al.* 2003) allow ontologies to be searched using keywords, but further refinement of the search criteria based on semantic entailments is not possible. For example, if one wants to search for ontologies in which Professor is a sub-class of Client, using a keyword based approach is not satisfactory, as all ontologies that contain the classes Professor and Client would be returned, whether or not the subsumption relationship holds between them. In this paper, we present an ontology search engine, called ONTOSEARCH2, which provides three approaches to searching ontologies semantically, namely: a keyword-based search tool, a search tool based on query answering and a search tool based on fuzzy query answering. Detailed examples of the three searches are presented later in this section.

ONTOSEARCH2 (Pan, Thomas, & Sleeman 2006) has two principal components, namely an ontology repository and query engine. It stores approximations of OWL ontologies in DL-Lite, and allows queries to be executed over all or part of this repository using SPARQL (Prud'hommeaux & Seaborne 2006). By using a DL-Lite approximation ONTOSEARCH2 is significantly faster than other comparable tools which perform full OWL DL entailment (up to two orders of magnitude faster on larger datasets (Pan & Thomas 2007)).

In Figure 2 we can see the structure of the system. Source ontologies are first loaded into a DL Reasoner (Pellet), and are checked for consistency before being classified. The resulting class hierarchy is stored in the repository where it is represented as a DL-Lite ABox against a meta-TBox which defines the meta structure of all OWL-DL ontologies. The ontology is then approximated using the semantic approximation technique presented in (Pan & Thomas 2007). Ontology metadata is analysed and converted to a set of fuzzy property instances, relating ontological structures to representations of the keywords found in the metadata, with a weight based on the semantic strength of the relationship. Conjunctive queries can be performed over the repository; each query being expanded by the PerfectRef algorithm in DL-Lite to a set of SQL queries. These SQL queries are then executed on the relational database engine which manages the repository.

When a new OWL ontology is submitted to ONTOSEARCH2, the ontology is first examined for metadata. This is stored as a mapping between the objects in the ontology (OWL ontologies, classes, properties, and instances) and keywords that appear in the metadata attached to that object (currently, the label and comment properties, and the URL of the object). How the keyword is related to the object is used to give a weighting for each mapping: for example, a keyword occurring in a label is given the highest weighting, and a keyword in a comment is given the lowest. In addition, keywords can be inherited from parents of an object (super classes or super properties); instances inherit the keywords of their classes, and all objects in an ontology will inherit the metadata of the ontology itself. In all cases, the greater the semantic distance between two objects which have an inheritance relationship, the lower the weighting is for the keywords inherited. Punctuation is removed from keywords before they are added to the metadata repository. If a keyword is less than three characters long or is a common word such as “and”, “the”, or “some”, it is discarded.

From this fuzzy ontology, we find objects within the repository which match the requirements of the original query. The weightings are used to specify minimum weights required for each term in the query, or they are used to rank the results by relevance to the initial terms.

Queries can be made through a simple keyword based search form, or can be submitted as SPARQL queries, optionally containing fuzzy extensions that can specify the degree of confidence required for each term in the query. Keyword based queries are expanded into fuzzy SPARQL queries, so all searches use the same internal process. The most basic search is for a set of keywords, where the results will list ontologies containing all the keywords. The query can be made more specific by adding search directives to the query:

[2] http://www.ontosearch.org/

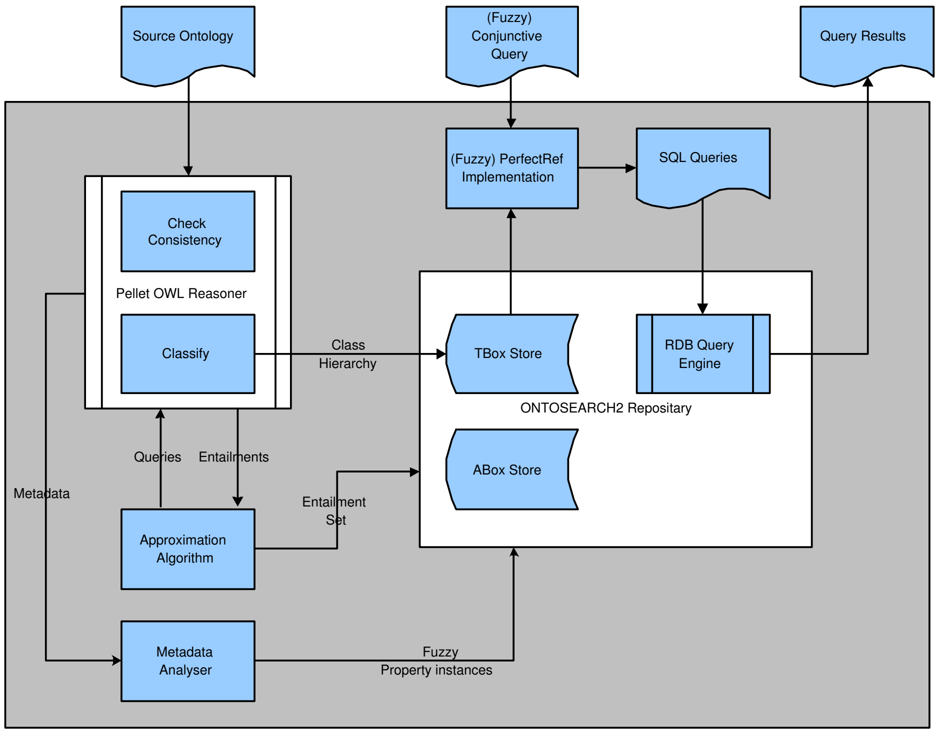


Figure 2: ONTOSEARCH2 structure diagram.

1. *TBox Searching* Restrictions on the search query can specify that a particular keyword should only be matched against a Class or a Property, but not against instance values in the repository. This is done by prefixing the keyword with *class:*, so the query *class:red wine* would match the keyword ”red” in class definitions only, and match the keyword *wine* across the entire ontology. Similarly a keyword can be restricted to only occur within property definitions within the ontology by prefixing the keyword with *property:*. For queries where all keywords should only match class and/or property definitions, the directives *pragma:Class* and/or *pragma:Property* can be added to the query.

2. *ABox Searching* To restrict a search term to only match within ABox (or instance) data, it can be prefixed with *instance:* Similarly the directive *pragma:Instance* will direct the search engine to search instance data only for every keyword. If used with both *pragma:Class* and *pragma:Property*, the search will exhibit its default behaviour, searching the entire contents of the repository (Class and Property definitions as well as instance data).

3. *Other Search Directives* By adding the search directive *pragma:Resource*, the search engine will find the object within each ontology which best matches the search terms. It will also cause the results to be listed as individual resources rather than as ontologies. Therefore if an ontology contains a single class or instance which has a very high match for the keywords, but which as an ontology has a low score, this class will be displayed above other matches. The default behavior is for the whole ontology with the highest sum of its objects' scores to be returned first.

Search results can be ranked as entire ontologies, or as individual objects within each ontology (by using the *pragma:Resource* directive in a query). In the first instance, we sum the total weightings for each object/keyword pairing that matched in an ontology. This total is used to sort the results, with the highest total being returned at the top of the rankings. By ranking results based on the semantic significance of the matches themselves, rather than by tracking the number of links to the ontology (Patel *et al.* 2003; Ding *et al.* 2005) we are able to find ontologies which are the closest match for a particular query regardless of their popularity.

In the case that results are to be returned as individual objects, the total sum of object/keyword weightings for each object in the repository is used to determine rank. This is used to return a list of all the different objects which matched the search terms.

Examples

Query-based Search Example This search uses the query answering facilities of ONTOSEARCH2 to find a class which has a particular label, which is a subclass of a class with a different label.

The search is specified in SPARQL. The query is for a class with a label of “Chablis” which is a subclass of a class with a label “Wine”. The query used is shown in listing 1.

```
SELECT ?X WHERE {
  ?X rdfs:label "Chablis" .
  ?X rdfs:subClassOf ?Y .
  ?Y rdfs:label "Wine" .
}
```

Listing 1: SPARQL Query

This is a standard SPARQL query which searches for some class with a *rdfs:label* of exactly "Wine" which has some subclass with a *rdfs:label* of exactly "Chablis". This query may return classes which are not direct subclasses because of the DL-Lite semantics which underlie the query engine in ONTOSEARCH2, a search engine that used RDF semantics would not match any indirect subclasses.

Fuzzy Query-based Search Example This search uses the fuzzy query engine in ONTOSEARCH2 to find a class which is a subclass of a different class, where both classes have particular metadata associated with them, with a certain level of confidence.

The search is specified using SPARQL with additional fuzzy values included as comments. This current query is for a class with a metadata keyword of "Chablis" with a degree of confidence ≥ 0.7 which is a subclass of a class with a keyword of "Wine" with a confidence of ≥ 0.5. The query is shown in listing 2.

```
SELECT ?X WHERE {
  ?X os2:hasKeyword "Chablis" . #FT# 0.7
  ?X rdfs:subClassOf ?Y .
  ?Y os2:hasKeyword "Wine" .    #FT# 0.5
}
```

Listing 2: fuzzy SPARQL Query

The property *os2:hasKeyword* is a fuzzy property in the ONTOSEARCH2 fuzzy metadata ontology which associates objects with keywords. For details on such fuzzy queries and related query answering algorithms, see (Pan *et al.* 2008).

ONTOSEARCH2: Future Work

We have presented three methods for semantically searching ontologies. By offering different methods of querying for data, we can allow users to choose the best tool available for their level of expertise and their requirements. Future work is centered around improving the metrics used to give different weightings. Additionally, we plan to investigate the use of machine learning techniques to evaluate the success of the weightings currently being used by comparing the ontologies users eventually select for their application, the position in the results which that ontology held, and the weightings used to generate the results. To help users find the best ontology or resource we are working on an ontology browser integrated with the search engine which will allow ontologies to be explored graphically, with matching resources highlighted, on the results page.

Another aim is to expand the metadata captured to include other recognised sources of metadata such as Dublin Core (DCMI 2003) and to eventually index all datatype properties present in an ontology as potential metadata.

CleOn[3]

Given the central role of ontologies in the Semantic Web, it is highly desirable that all ontologies are assessed for logical consistency and against specific criteria of the particular application. Further, ontology evaluation should ensure that the information available in the ontology is complete and consistent with respect to the domain of interest. This evaluation should be performed during the ontology development process (i.e. ontology building) as well as during the ontology maintenance process (e.g. ontology evolution[4]).

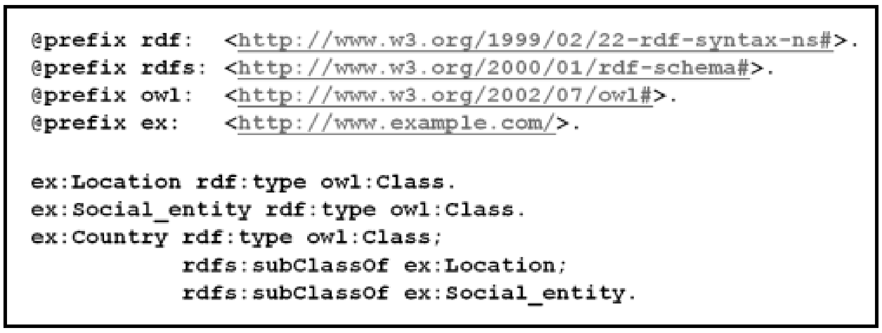
Although DL reasoners (e.g. FaCT++ (Tsarkov & Horrocks 2006) and Pellet (Sirin *et al.* 2007)) can determine the logical consistency of an ontology, they are unable to evaluate whether the vocabulary used for class labels is coherent with the domain represented. Additionally, Volker et al. (Volker, Hitzler, & Cimiano 2007) suggest that the terms chosen as class names in ontologies (e.g. OWL axioms) convey some of the meaning (semantics) of these axioms to the human reader.

Let's consider a team of knowledge engineers developing an ontology about the geography domain. The class labeled with the term **country** has been used by different knowledge engineers to be a subclass of both *'Location'* and *'Social_entity'* (Figure 3). However, this representation describes two distinct aspects of the domain through the same class (i.e. *'Country'*). On the one hand, the term **country** defines the social and political aspects of countries (e.g. government). On the other hand, the term is used to describe the territory occupied by a country. In information retrieval systems, this ontology would cause many inappropriate results to be returned and would require the user to extract the relevant information to his/her search.

The OntoClean methodology (Guarino & Welty 2000) assigns meta-properties; namely *Unity*, *Identity*, *Rigidity* and *Dependence*, to describe relevant aspects of the intended interpretation of the properties, classes, and relations that make up the ontology. The evaluation of the taxonomic structure is dictated by the constraints imposed on the different meta-properties. The notion of *identity* is based on intuitions about how objects interact with the world around them. OntoClean proposes to distinguish between properties that supply their "own" identity criteria and those that inherit their identity criteria from subsuming properties. In general, a country represents different social aspects and hence carries its own identity, whereas a geographical region inherits the identity of its location (e.g. fauna and flora). Therefore, OntoClean suggests adding a new class (e.g. *'Geographical region'*) to represent the physical aspects related to countries. However, Volker et al. (Volker, Vrandecic, & Sure 2005) have shown that this methodology was very time consuming and that the agreement among knowledge engineers was pretty low (only 38%). Moreover, they found that knowledge engineers based the assignment of meta-properties on the examples provided by Guarino and Welty

[3]http://www.csd.abdn.ac.uk/ qreul/CleOn.html

[4]Ontology evolution is the systematic application of changes to an ontology while maintaining the consistency of the ontology and all its dependent artifacts.



```
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl:  <http://www.w3.org/2002/07/owl#>.
@prefix ex:   <http://www.example.com/>.

ex:Location rdf:type owl:Class.
ex:Social_entity rdf:type owl:Class.
ex:Country rdf:type owl:Class;
           rdfs:subClassOf ex:Location;
           rdfs:subClassOf ex:Social_entity.
```

Figure 3: Example ontology.

rather than their formal definition, thus suggesting that it would be difficult to apply OntoClean to highly specialised domain ontologies.

As a result, we have developed an alternative approach, called CleOn (Sleeman & Reul 2006; Reul, Sleeman, & Fowler 2008), which detects lexical incoherencies in ontological structures given a lexical resource (e.g. thesauri). A thesaurus generally classifies linguistic terms by themes or topics. Often, this classification is achieved through the *hypernym* relation, which specifies that if a term **A** is a hypernym of a term **B** then **A** is more general than **B**.

Our overall approach contains three phases. We first extract a *lexical path* for each class in the ontology. For this example, we have extracted these paths from WordNet (Fellbaum 1998) as it is a general purpose thesaurus and provides the type of information that we require. A class is considered *lexically satisfiable* only if an exact match for its class label is found in the thesaurus. For example, **social entity** is not a term in WordNet and therefore the concept is *lexically unsatisfiable* and its lexical path is the empty set. Once an exact match for the class label has been found in the thesaurus, the lexical path of the class is created by adding its matching term in the thesaurus, then its hypernym, then the hypernym of its hypernym, and so on until the root term of the thesaurus is encountered. For example, an exact match for *'Location'* is found in the thesaurus. As a result, its lexical path, denoted LP(*'Location'*), contains the sequence $\prec$**location, physical_object, physical_entity, entity**$\succ$.

Secondly, we determine the *adequacy* of every taxonomic relationship in the ontology. There are several reasons why a link might be considered to be inadequate. Initially, every taxonomic relationship containing one or more lexically unsatisfiable classes are removed from the ontology. For example, *'Country'* $\sqsubseteq$ *'Social_entity'* is removed from the ontology as **social entity** is present in the ontology. Then, CleOn inspects the *adequacy* of every remaining taxonomic relationship in the ontology with regard to the lexical paths of their constituents. A concept name *'A'* is adequately subsumed by a concept name *'B'* if one of the sequences in the lexical path of *'B'* is contained in one of the sequences in the lexical path of *'A'*, where *'A'* is the child node, and *'B'* corresponds to the parent node. Examining the taxonomic relationship between *'Location'* (parent node) and *'Country'* (child node), we retrieve the lexical paths for both these concepts as described previously:

- *LP(Location) [Parent node]*
  $\{\prec$*location, physical_object, physical_entity, entity*$\succ\}$
- *LP(Country) [Child node]*
  $\{\prec$*country, administrative_district, district, region, location, physical_object, physical_entity, entity*$\succ\}$

As all the elements of the parent path are included in the child path, we consider this link to be adequate. The removal of all inadequate links results in a skeleton tree, which we refer to as Tree-0. Furthermore, this process creates "detached" subtrees and/or *orphan* nodes[5].

Finally, CleOn places these orphan nodes and "detached" subtrees back onto Tree-0 so as to create a *lexically coherent* ontology (Definition 1). When there are several nodes at which an orphan element or detached subtree can be placed, the user is prompted to choose among the potential taxonomic relationships. It is important to note that we do not wish to imply that a taxonomic relationship is wrong. Instead we suggest that a taxonomic relationship is inadequate given a particular conceptualisation and a thesaurus.

Definition 1 (Lexically Coherent Ontology) *An ontology O is lexically coherent if all taxonomic relationships in O are adequate.*

In Sleeman and Reul (Sleeman & Reul 2006), we have introduced the CleOn system, which when provided with an OWL ontology (Bechhofer *et al.* 2004) and a thesaurus (e.g WordNet) in the appropriate format, collaboratively creates a lexically coherent ontology. The application is written in Java and uses Jena API[6] to process and manipulate OWL ontologies. Furthermore, we described several possible improvements resulting in four different modes; namely *conservative mode*, *include head mode*, *inclusive mode* and *interactive selection of word sense mode*. The conservative mode is the system default mode applying our approach to OWL ontologies and only considers the first sense in WordNet when creating the different lexical paths. Whereas the conservative mode determines the lexical satisfiability of a concept name based on exact match, the include head mode also searches for the head of the noun phrase when no prior

[5]An orphan node is a subtree with a single node.

[6]http://jena.sourceforge.net/ontology/index.html

match is found. The inclusive mode includes the intermediary nodes found in the lexical path of the child node but not present in the lexical path of the parent node as part of the *Creating a lexically coherent ontology* step of the algorithm. Finally, the interactive selection of word sense mode allows the user to choose the intended meaning when an ambiguous term is found in the thesaurus.

In Reul et al. (Reul, Sleeman, & Fowler 2008), we report the results of an experiment in which we used our methodology to evaluate a domain dependent ontology. The evaluation demonstrated that our method does not solely rely on WordNet, but relies on a thesaurus which is appropriate for the domain. In this case, we had to create this thesaurus as none was available for the engineering domain. Moreover, we showed that our method was consistently simpler to apply than the OntoClean methodology, which is a very encouraging result. The precision and recall metrics (Dellschaft & Staab 2006) demonstrated that the discordance was in the lexical layer rather than in the concept hierarchy. Therefore, the precision and recall metrics suggest that the terminology used in industry is somewhat inconsistent with the terminology used by mechanical engineering academics.

CleOn: Future Work

We realise that there is still room for improvements to both the approach and the system. Firstly, the *Creating a lexically coherent ontology* phase proposes several potential superconcepts when adding subtrees to Tree-0 based on the terms contained in the lexical path of a subtree. However, it could be argued that terms occurring in every lexical path convey less information than terms occurring only once. Resnik (Resnik 1995) proposes the *information content* of a concept which is calculated as the negative log likelihood of a concept occurring in a repository. The implementation of this assumption would reduce the number of choices offered as part of *Creating a lexically coherent ontology* step of the algorithm. Therefore, the system would only suggest the root term of the thesaurus if no other alternative is available.

Secondly, the current approach evaluates General Concept Inclusions (GCIs) between two concept names (e.g. *'Country'* $\sqsubseteq$ *'Location '*). However, many OWL DL ontologies are composed of GCIs between a concept name and a concept description (e.g. conjunction). In further work, we plan to use a DL reasoner (e.g. Fact++ (Tsarkov & Horrocks 2006)) to extract the implicit knowledge from GCIs composed of concept descriptions. Moreover, we plan to also consider a wider range of OWL DL constructs, such as `owl:unionOf` and `owl:equivalentClass`.

Finally, the CleOn methodology detects lexical incoherencies in these ontological structures given a thesaurus (e.g. WordNet). A significant issue is how one might acquire appropriate thesauri; one approach would be to use ONTOSEARCH2 (Pan, Thomas, & Sleeman 2006) to search the web for them; the second and probably more practical approach is to encourage domain experts to develop them. In any event, we would hope that these thesauri would be expressed in a standard formalism such as SKOS (Miles & Perez-Aguera 2007). If the second route is followed, editors will be needed to develop these thesauri, and we believe that the use of information retrieval techniques would facilitate this process.

RepairTab[7]

Inconsistencies in OWL ontologies can easily occur, and it is a challenging task for ontology modelers, especially for non-expert ontology modelers, to resolve such inconsistencies. Therefore, we proposed an ontology debugging tool, called 'RepairTab' that can provide guidance on (1) selecting the axioms to modify, and (2) how to minimise the impact of proposed modifications on the ontology.

Ranking Axioms

Existing ontology debugging approaches (Schlobach & Cornet 2003; Wang *et al.* 2005; Meyer *et al.* 2006) can identify the problematic axioms for unsatisfiable concepts. After pinpointing these problematic axioms, the next step is to resolve the error by removing or modifying one of the pinpointed axioms. It is difficult for tools to make specific suggestions for how to resolve problems, because it requires an understanding of the meaning of the set of classes. Only a few existing approaches provide support for resolving inconsistencies in ontologies. Tools such as SWOOP (Kalyanpur *et al.* 2006) that provide this functionality are quite limited in their approach; the problematic axioms are ranked in order of importance. We illustrate their strategies with the following example.

Example 1 *An ontology contains the following axioms:*
(1) $\mathsf{Cows} \sqsubseteq \mathsf{Animals}$
(2) $\mathsf{Sheep} \sqsubseteq \mathsf{Animals}$
(3) $\mathsf{Lions} \sqsubseteq \exists\, \mathsf{eat.(Cows} \sqcap \mathsf{Sheep)}$
(4) $\mathsf{Tigers} \sqsubseteq \exists\, \mathsf{eat.(Cows} \sqcap \mathsf{Sheep)}$
(5) $\mathsf{Cows} \sqsubseteq \neg\, \mathsf{Sheep}$

Axioms (3) and (5) cause Lions *to be unsatisfiable; axioms (4) and (5) cause* Tigers *to be unsatisfiable. In this case, axiom (5) causes two unsatisfiable concepts; its removal can resolve two unsatisfiabilities. Thus, axiom (5) is regarded as less important than axioms (3) and (4). Moreover, removing axiom (5) is a change which preserves more of the information in the ontology, as otherwise two axioms would have to be removed (axioms (3) and (4)). By removing axiom (5) we can retain the information about* Lions *and* Tigers*. Existing tools use these heuristics, i.e., number of unsatisfiabilities caused by the axiom, and information lost by a modification. Hence, the user is suggested to remove/modify axiom (5).*

Intuitively, an axiom whose removal causes more information to be lost is regarded as more important, and should be preserved. However, it does not necessarily follow that such suggestions are the best results for all ontologies. A human ontology engineer may well propose a better solution in particular cases. For example, an ontologist may believe that sibling concepts are usually disjoint with each other. That means, the disjointness of Cows and Sheep

[7]http://www.csd.abdn.ac.uk/ slam/RepairTab/

should be kept in the above example. This heuristic has an opposite result to the factor of information lost. Moreover, Rector *et al.* (Rector *et al.* 2004) enumerate a number of common error patterns made by non-expert users in modeling OWL ontologies. For example, differences between the linguistic and logical usage of 'and' and 'or' often cause confusion. Non-expert users might interpret axiom (3) as that 'lions eat sheep and cows'; it actually means 'lions eat something which is both a cow and a sheep'. By using the common error patterns, we are able to evaluate axioms (3) and (4) as likely to be erroneous.

It is very difficult for tools to determine the optimal way to resolve such errors (semi-)automatically. Many tools simply leave it up to the user to determine how to resolve such errors. We believe that it will be useful to analyse the heuristics ontology engineers use to solve these kinds of problems, and to incorporate these heuristics in an ontology management tool. We firstly acquired a number of heuristics used by ontology engineers from an empirical study (see (Lam 2007) for more details). We then combine our acquired heuristics with those already incorporated in existing debugging tools and divide them into the following categories:

1. Impact of removal on the ontology – two heuristics are presented to analyse the loss of entitlements due to removing axioms (i.e., arity and number of lost entailments)
2. Knowledge of ontology structure – five novel heuristics are proposed to analyse:
   (a) disjointness between siblings,
   (b) disjointness between non-siblings,
   (c) sibling patterns,
   (d) length of paths in a concept hierarchy, and
   (e) depth of concepts in a concept hierarchy
3. Linguistic heuristic – one heuristic is presented to analyse the similarities of the names of the concepts

From the result in the empirical study, we learnt that some subjects (the ontology engineers) took common OWL modeling errors into account when resolving the inconsistencies in ontologies; some subjects took the process of constructing ontologies into account, such as integrating/merging ontologies. We therefore, clustered the heuristics into two sets: "common error heuristics" and "merging heuristics", then conducted a usability study with non-expert users. The results of our usability study (Lam 2007) show that (1) with heuristic support, the users were guided towards the axioms which would be the best to modify, and they achieved correct modifications in a shorter time; (2) the heuristics could increase the users' confidence in the modifications which they make; (3) the common error heuristics were useful for dealing with ontologies with common mistakes; and (4) the merging heuristics were useful for dealing with merged ontologies.

Impact of Changes

Furthermore, whenever (parts of) an axiom is removed, it is easy for the knowledge engineer to unintentionally remove implicit information from the ontology. In order to minimise the impact on the ontology, it is important to calculate the lost entailments (e.g., subsumption relationships) of named concepts which occur due to the removal of (parts of) axioms. We use the following example to illustrate our approach.

Example 2 *For a bird-penguin example, two axioms cause* Penguin *to be unsatisfiable:*
(1) $\mathsf{Bird} \sqsubseteq \mathsf{Animal} \sqcap \mathsf{CanFly}$
(2) $\mathsf{Penguin} \sqsubseteq \mathsf{Bird} \sqcap \neg\, \mathsf{CanFly}$

To resolve this problem, if we remove the part $\mathsf{Bird} \sqsubseteq \mathsf{CanFly}$ *from axiom (1), then the implicit information* $\mathsf{Eagle} \sqsubseteq \mathsf{CanFly}$ *will be lost (see Figure 4 on the left-hand side). If we remove the part* $\mathsf{Penguin} \sqsubseteq \mathsf{Bird}$ *from axiom (2), then the implicit information* $\mathsf{Penguin} \sqsubseteq \mathsf{Animal}$ *will be lost (see Figure 4 on the right-hand side).*

To minimise the loss of information, when $\mathsf{Bird} \sqsubseteq \mathsf{CanFly}$ *is removed from axiom (1), the modeler should be notified that the information* $\mathsf{Eagle} \sqsubseteq \mathsf{CanFly}$ *should be added back to the ontology. When* $\mathsf{Penguin} \sqsubseteq \mathsf{Bird}$ *is removed from axiom (2), the modeler should be notified that the information* $\mathsf{Penguin} \sqsubseteq \mathsf{Animal}$ *should be added back to the ontology.*

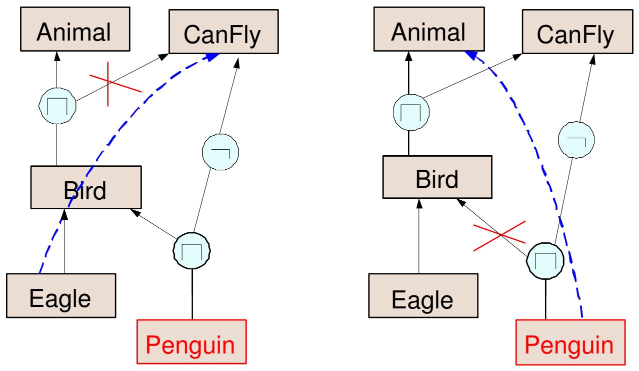


Figure 4: (Left-hand side) Option one: removing 'birds can fly', the implicit information 'eagles can fly' is lost (dashed arrow). (Right-hand side) Option two: removing 'penguins are birds', the implicit information 'penguins are animals' is lost. (NB: Deletion is indicated by a $\times$ on the appropriate link)

Whenever parts of an axiom or a whole axiom are removed, it frequently happens that intended entailments are lost. In order to minimise the impact on the ontology, we analyse the lost information (e.g., concept subsumption) of concepts due to the removal of (parts of) axioms. We propose a fine-grained approach which allows us to identify changes which are *helpful* in that they restore lost information due to removal of axioms, and those that are *harmful* in that they cause additional unsatisfiability.

Example 3 *Continuing the above example, when* $\mathsf{Bird} \sqsubseteq \mathsf{CanFly}$ *from axiom (1) is removed, the helpful change is* $\mathsf{Eagle} \sqsubseteq \mathsf{CanFly}$*; when* $\mathsf{Penguin} \sqsubseteq \mathsf{Bird}$ *from axiom (2) is removed, the helpful change is* $\mathsf{Penguin} \sqsubseteq \mathsf{Animal}$*. Harmful changes to replace the part* CanFly *in axiom (1) are* $\neg$Animal, Penguin, and Eagle. *This is because if the part*

CanFly *in axiom (1) is replaced by* ¬Animal*, then* Bird *will become unsatisfiable. If that part is replaced by* Penguin*, then* Penguin *is defined as a type of* Penguin*. If it is replaced by* Eagle*, then* Penguin *is defined as a type of* Eagle*.*

We conducted a usability evaluation to test its effectiveness by comparing RepairTab with existing ontology debugging tools (Lam *et al.* 2007). The results show that (1) the fine-grained approach greatly facilitated the subjects understanding of the reasons for concepts unsatisfiability, and (2) the subjects found the helpful changes very useful to minimise the impact of changes on the ontologies; but the harmful changes were less useful for them. For evaluating the runtime and memory performance of the proposed algorithms, we use a number of existing ontologies to evaluate the efficiency. The results demonstrate that our algorithms provide acceptable performance when used with real world ontologies (Lam 2007).

RepairTab: Future Work

Our debugging approach combines heuristic and formal approaches for dealing with inconsistencies in ontologies. It shows that the approach of acquiring human heuristics and encoding them in a tool is viable for ontology debugging. It also shows that the fine-grained approach for indicating lost entailments and recommending modifications is a powerful novel approach.

Conclusion

How these three tools interoperate to locate and refine ontologies, is illustrated in Figure 1 and will soon be the subject of various useability studies. The newly introduced W3C standards have made the interoperation of such systems viable. Further, the use of tools such as CleOn and RepairTab enhance and refine the set of ontologies available on the Semantic Web.

Acknowledgment

Support from the AKT project (GR/N15764/01), the EC Knowledge Web project (IST-2004-507842) and DTI/Rolls-Royce IPAS project (DTI project No. TP/2/IC/6/I/10292). We thank Dr. Wamberto Vasconcelos for helpful discussions of aspects of this work.

References

Bechhofer, S.; van Harmelen, F.; Hendler, J.; Horrocks, I.; McGuinness, D. L.; Patel-Schneider, P. F.; and Stein, L. A. 2004. OWL Web Ontology Language Reference. W3C Recommendation, World Wide Web Consortium.

DCMI. 2003. Dublin Core Metadata Element Set, Version 1.1: Reference Description. DCMI Recommendation, URLhttp://dublincore.org/documents/dces/.

Dellschaft, K., and Staab, S. 2006. On How to Perform a Gold Standard Based Evaluation of Ontology Learning. In *Proceedings of the 5th International Semantic Web Conference (ISWC 2006)*, 228–241.

Ding, L.; Pan, R.; Finin, T.; Joshi, A.; Peng, Y.; and Kolari, P. 2005. Finding and Ranking Knowledge on the Semantic Web. In *Proceedings of the 4th International Semantic Web Conference*, LNCS 3729, 156–170. Springer.

Fellbaum, C. 1998. *WordNet: An Electronic Lexical Database*. MIT Press.

Guarino, N., and Welty, C. 2000. Ontological Analysis of Taxonomic Relationships. In *Proceedings of the 19th International Conference on Conceptual Modeling (ER 2000)*, 210–224.

Kalyanpur, A.; Parsia, B.; Sirin, E.; and Cuenca-Grau, B. 2006. Repairing Unsatisfiable Concepts in OWL Ontologies. In *Proceedings of the 3rd European Semantic Web Conference (ESWC2006)*, 170–184.

Lam, J. S. C.; Pan, J. Z.; Sleeman, D.; and Vasconcelos, W. 2007. A Fine-Grained Approach to Resolving Unsatisfiable Ontologies. *Journal on Data Semantics* 10.

Lam, J.; Sleeman, D.; and Vasconcelos, W. 2005. ReTAX++: a Tool for Browsing and Revising Ontologies. In *osters & Demonstration Proceedings of ISWC-05 (Galway, Ireland). ISWC Conference Directorate*.

Lam, J. S. C. 2007. *Methods for Resolving Inconsistencies in Ontologies*. Ph.D. Dissertation, University of Aberdeen.

Meyer, T.; Lee, K.; Booth, R.; and Pan, J. Z. 2006. Finding maximally satisfiable terminologies for the description logic ALC. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-06)*.

Miles, A., and Perez-Aguera, J. 2007. SKOS: Simple knowledge organisation for the web. *Cataloging and Classification Quarterly* 43(3-4):69–83.

Pan, J. Z., and Thomas, E. 2007. Approximating OWL-DL Ontologies. In *Proc. of the 22nd National Conference on Artificial Intelligence (AAAI-07)*. To appear.

Pan, J. Z.; Stamou, G.; Stoilos, G.; Taylor, S.; and Thomas, E. 2008. Scalable Querying Services over Fuzzy Ontologies. In *Proc. of the Seventeenth International World Wide Web Conference (WWW 2008)*. To appear.

Pan, J. Z.; Thomas, E.; and Sleeman, D. 2006. ONTOSEARCH2: Searching and Querying Web Ontologies. In *Proc. of WWW/Internet 2006*, 211–218.

Patel, C.; Supekar, K.; Lee, Y.; and Park, E. K. 2003. OntoKhoj: a semantic web portal for ontology searching, ranking and classification. In *WIDM '03: Proceedings of the 5th ACM international workshop on Web information and data management*, 58–61. New York, NY, USA: ACM Press.

Prud'hommeaux, E., and Seaborne, A. 2006. SPARQL query language for RDF. W3C Working Draft, http://www.w3.org/TR/rdf-sparql-query/.

Rector, A.; Drummond, N.; Horridge, M.; Rogers, J.; Knublauch, H.; Stevens, R.; Wang, H.; and Wroe, C. 2004. OWL Pizzas: Practical experience of teaching OWL-DL: Common errors & common patterns. In *Proceedings of the European Conference on Knowledge Acquistion*, 63–81.

Resnik, P. 1995. Using Information Content to Evaluate Semantic Similarity in a Taxonomy. In *Proceedings of* *the 14th International Joint Conference on Artificial Intelligence*, 448–453.

Reul, Q.; Sleeman, D.; and Fowler, D. 2008. CleOn: Resolution of lexically incoherent concepts in an engineering ontology. Technical report, University of Aberdeen. To be published.

Schlobach, S., and Cornet, R. 2003. Non-standard reasoning services for the debugging of description logic terminologies. In *8th International Joint Conference on Artificial Intelligence (IJCAI'03)*.

Sirin, E.; Parsia, B.; Grau, B. C.; Kalyanpur, A.; and Katz, Y. 2007. Pellet: A practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web* 5(2):51–53.

Sleeman, D., and Reul, Q. 2006. CleanONTO: Evaluating taxonomic relationships in ontologies. In *Proceedings of 4th International EON Workshop on Evaluation of Ontologies for the Web*.

Tsarkov, D., and Horrocks, I. 2006. FaCT++ description logic reasoner: System description. In *Proceedings of the 3d International Joint Conference on Automated Reasoning (IJCAR 2006)*, 292–297.

Volker, J.; Hitzler, P.; and Cimiano, P. 2007. Acquisition of OWL DL axioms from lexical resources. In *Proceedings of the 4th European Semantic Web Conference (ESWC'07)*, 670–685.

Volker, J.; Vrandecic, D.; and Sure, Y. 2005. Automatic evaluation of ontologies (AEON). In *Proceedings of the 4th International Semantic Web Conference (ISWC2005)*, 716–731.

Wang, H.; Horridge, M.; Rector, A.; Drummond, N.; and Seidenberg, J. 2005. Debugging OWL-DL Ontologies: A heuristic approach. In *4th Internation Semantic Web Conference*.