knOWLer - Ontological Support for Information Retrieval Systems

Claudia Ciorăscu
Université de Neuchâtel
Pierre-à-Mazel 7
2000 Neuchâtel
claudia.ciorascu@unine.ch

Iulian Ciorăscu
Université de Neuchâtel
Pierre-à-Mazel 7
2000 Neuchâtel
iulian.ciorascu@unine.ch

Kilian Stoffel
Université de Neuchâtel
Pierre-à-Mazel 7
2000 Neuchâtel
kilian.stoffel@unine.ch

ABSTRACT

In addition to the human-readable contents of the shared documents, the new generation of WWW requires machine-readable formalization of data.

We present an ontology-based information management system called knOWLer, targeting semantic integration into large-scale information systems. In this paper, we are specially focusing on large-scale information retrieval systems. The semantic is provided through an ontology language (OWL), showing that ontological reasoning can be scaled to sizes of standard IR systems.

We propose an information management system for automatic document manipulation based on the semantics added by an ontology to the raw data. The main capabilities of our application are expressivity - provided by the ontology language - and scalability - induced by the persistent storage mechanism.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods; I.2.6 [**Artificial Intelligence**]: Learning—*knowledge acquisition*; D.2.2 [**Software Engineering**]: Modules and interfaces

Keywords

ontology, knowledge system, semantic language, information retrieval, document management

1. INTRODUCTION

For most of the data content documents intended for human consumption, it is difficult to locate user-relevant information with the standard web search engines. Because the information from data was automated based on keyword matches and not on their meanings, they cannot take into account the relationships between words, or to map between the terminology of different communities for the same knowledge model [8].

That leads us to the formation of the Knowledgeable Web [5] that aims at machine processable meaning of information, being more targeted at problem solving and query answering [10].

To extend the capabilities of the automation process in order to capture the semantics of the data it is necessary to specify somehow the meaning of the shared knowledge. This will provide the automated processes a meaningful controlled vocabulary of terms and their semantics.

In the field of information retrieval several people experimented with the idea of integrating ontologies / knowledge bases into information retrieval systems. WordNet^{TM}[1] was one of the candidate ontologies of potential interest [16]. These experiments were very difficult to conduct as on one hand the Ontology management system could not handle the large corpora used in IR and on the other hand the IR systems were not really set up to integrate semantic information in form of an ontology. Therefore it was very difficult to conduct large series of tests.

One advantage of constructing knowledge-based systems consists in the dynamics of the information expressed that can be used in several ways. Unlike most programs that hard-codes the process of how to solve a task or to encode a solution process, our ontology-based system depends on the formalization of the domain, embedding the required knowledge of how to perform that task. This gives our framework a great flexibility and comprehensibility, making its usage beneficial in IR applications.

Our contribution in this paper is to show that an ontology-base information management system can be successfully integrated into Information Retrieval Systems. This is illustrated by using an OWL-ontology derived from the WordNet^{TM} lexical database and a standard corpus obtained from the Wall Street Journal collection of TREC-4.

This paper is organized as follows: the first section overviews a subset of the relevant existing ontology representation languages and related applications; the next sections will focus

[1]WordNet is a trademark of Princeton University.

on the technical aspects and capabilities of our system, followed by a description of WordNet as an OWL ontology. A series of test cases for document indexing and retrieval will be presented in section 5, analyzing the performance of the system.

2. ONTOLOGY LANGUAGES AND TOOLS

Taxonomic hierarchies of classes, which have been used explicitly for centuries in technical fields (systematic biology, library science, government departments, etc.), made the first attempt in the direction of the knowledge description languages, by defining the common vocabulary in which shared information is represented. These structures permit the discovery of implicit information through the use of defined taxonomies and inference rules [7].

An ontology is "a formal, explicit specification of a shared conceptualization. The conceptualization refers to an abstract model of some phenomenon in the world which identifies the relevant concepts of that phenomenon. Explicit means that the type of concepts used and the constraints on their use are explicitly defined. Formal refers to the fact that the ontology should be machine processable, i.e. the machine should be able to interpret the information provided unambiguously. Shared reflects the idea that an ontology captures consensual knowledge, that is, it is not restricted to some individual, but accepted by a group" [3]. An ontology representation language is defined by two aspects: the syntax of a language and the semantics associated with that language [13].

Many of the existing ontology languages are based on description logic. Unfortunately, the systems using these languages are based on primary memory and thus limited to small ontologies (Fact [1], Classic [2]). Different languages have been proposed for modeling and sharing knowledge databases over the web, including RDF-based languages such as DAML+OIL and OWL. Only few systems are supporting this kind of languages but they are providing very simple inference for RDF. Among these there are some systems based on persistent storage, making them better candidates for managing larger knowledge bases, but they don't provide a lot of semantics. As an illustration, we mention Jena [11] that is a Java tool, supplying an API for manipulating knowledge bases. A better suited system for very large knowledge bases is Parka-DB [15]. Most of the existing systems have scalability limitations, making them unusable for large-scale ontologies.

We are presenting a system that has a good trade-off between expressivity and scalability, allowing it to handle applications having ontologies with a huge number of statements.

To achieve the goal of scalability and expressivity, our system implements an extension of the OWL Lite language (subset of OWL). Built upon the foundations of the DAML+OIL specification, OWL [4] is a web ontology language based on the current semantic web standards being more expressive than XML, RDF and RDF Schema.

In order to show how our system encompasses these limitations of the previous mentioned systems, we will present in the next section the architecture of the knOWLer system.

3. KNOWLER SYSTEM ARCHITECTURE

The knOWLer system is an innovative ontology-based tool for local or distributed information management. Starting from Parka's ideas and principles, knOWLer has a new design, removing some of the limitations of Parka-DB while keeping the performance at the same level.

By choosing OWL as the representation language, knOWLer makes an important step toward modern ontology languages but still having support for traditional frame-based systems. Developed by W3C team, the OWL language consists of three overlapped layers [14]. The simplest language among them, OWL Lite, supports classification hierarchy and simple constraint features. It provides a lower entry threshold to the language guarantying the computational completeness and decidability. On top of it, OWL DL is more expressive, but still preserve the type separation. The last and most general layer, OWL Full, includes the complete OWL vocabulary and gives the freedom of interpretation provided by RDF.

In the knOWLer system, we extended the OWL Lite layer by allowing the usage of *RDF statements*, giving users the possibility to treat them as individuals and to apply properties to them. This language will be referred as Extended OWL Lite in this paper. The reasoning provided by knOWLer System also follows the semantics defined by Extended OWL Lite language.

The architecture of our ontology-based information management system consists in four distinct modules (Figure 1).

3.1 knOWLer Kernel

The knOWLer kernel is the core component of the knOWLer system. It provides the main functionality required for manipulating the structure and the semantics defined by an ontology. It also has complete reasoning for Extended OWL Lite and for DAML (in fact the subset of DAML that is equivalent with Extended OWL Lite). The kernel also supports RDF/RDFS but the reasoning will assume Extended OWL Lite restrictions. The developing language (ANSI C++) facilitates porting knOWLer to the most common operating systems, without compromising the performance. Currently, we support Linux, Windows and MacOS/X operating systems and other ports are planned.

As part of the kernel, the reasoner is able to handle queries based on RDQL syntax, transforming them into one or more requests to the Storage Interface. Functional extension of RDQL syntax allows us to write more sophisticated queries (e.g. transitive closure over a property, inheritance).

Furthermore a caching mechanism is provided. It ensures that frequently used entities are kept in memory for fast access. There is no restriction regarding the type of entities that can be cached. Classes, properties and individuals are supported in the same way. The performance will increase if a larger cache is available, but we still get reasonable performance even if no cache is used at all.

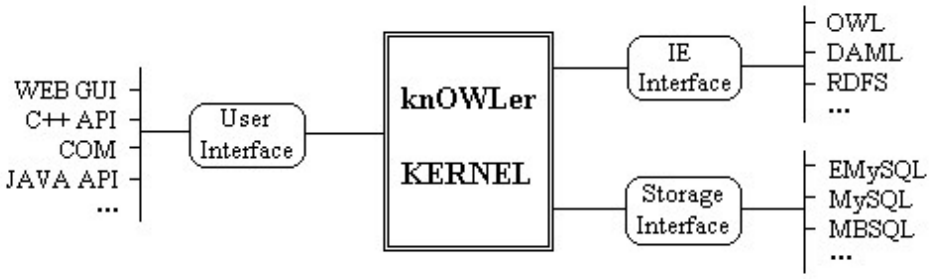

Figure 1: knOWLer's Architecture

3.2 knOWLer Interfaces

The knOWLer kernel uses external modules for importing/ exporting ontologies, storing data and communicating with user and/or client applications. Each module implements the corresponding interface.

The Import/Export (IE) Interface provides the full functionality for importing/exporting DAML, OWL and RDF/RDFS ontologies. These languages are converted to the internal representation that is "fed" to knOWLer kernel. For the moment, only XML/RDF and Ntriples notations are recognized. Any other notations can be added by implementing the knOWLer IE Interface.

The Storage Interface provides the mechanism for storing the data used by the kernel. It imposes a relational model that has to be supported by the Storage Module. The Storage Module can be implemented using a memory-based or persistent storage mechanism. The current implementation includes persistent storage based on MySQL-embedded and MySQL-client.

Once the considered ontology is loaded into the database, a verification step is applied on the imported data. All the syntactic checks with respect to the OWL syntax are done on the fly. An example of such a test is to forbid having more than one property inside an OWL restriction definition. A complete check of OWL compliance has to be started explicitly as a post-processing step. The compliance check has to be done after the whole ontology is imported into the system because there is not a predefined order for OWL statements.

The User Interface provides a way for clients and/or users to access the functionality of the system. The easiest way to use knOWLer from a client point of view is using its C++ API, the others being built on top of it.

4. THE WORDNET ONTOLOGY

For our test cases we defined three ontologies. All of them are built in the OWL language in different namespaces. The first ontology represents the semantics contained within WordNet. The other two ontologies extend the first one with definitions for stem and document manipulations.

Defined in the namespace *wordnet#*, the WordNet OWL-ontology consists of a small set of definitions of classes (as *wordnet#LexicalConcept*, *wordnet#Noun*, etc.) and properties/ relations (as *wordnet#wordForm*, *wordnet#hypernym*, *wordnet#antonym*, etc.). The full version of WordNet ontology in OWL language, including the definitions and instances, can be found on the knOWLer Project site[2].

The next subsections give a detailed description of these ontologies and their implementation in the OWL language.

4.1 Definitions of Classes

The WordNet OWL-Ontology was obtained from the WordNet^{TM} 1.7.1 lexical database. There are two important classes defined in this ontology: *wordnet#WordObject* class that represents lexical words:

```
<owl:Class rdf:ID="WordObject">
    <rdfs:comment>
        an word will be an instance of WordObject
    </rdfs:comment>
</owl:Class>
```

and *wordnet#LexicalConcept* corresponding to WordNet synsets:

```
<owl:Class rdf:ID="LexicalConcept">
    <rdfs:comment>WordNet synset</rdfs:comment>
</owl:Class>
```

Each syntactic category is derived from the *wordnet#LexicalConcept* class. Because some of the properties can be applied on multiple syntactic categories, we defined groups of synset types in order to maintain the definitions conform to the OWL Lite language. Both *wordnet#Nouns_and_Verbs* and *wordnet#Nouns_and_Adjectives* are directly derived from the *wordnet#LexicalConcept* class:

```
<owl:Class rdf:ID="Nouns_and_Verbs">
    <rdfs:subClassOf rdf:resource="#LexicalConcept"/>
</owl:Class>
```

[2] *http://taurus.unine.ch/GroupHome/Projects/knOWLer/*

```
<owl:Class rdf:ID="Nouns_and_Adjectives">
    <rdfs:subClassOf rdf:resource="#LexicalConcept"/>
</owl:Class>
```

Each synset type is defined as a subclass of the corresponding groups that includes it. For example *wordnet#Noun* is subclass of *wordnet#Nouns_and_Verbs* and also subclass of *wordnet#Nouns_and_Adjectives*:

```
<owl:Class rdf:ID="Noun">
  <rdfs:subClassOf rdf:resource="#Nouns_and_Verbs"/>
  <rdfs:subClassOf rdf:resource="#Nouns_and_Adjectives"/>
  <owl:disjointWith rdf:resource="#Verb"/>
  <owl:disjointWith rdf:resource="#Adjective"/>
...
</owl:Class>
```

Using *owl:disjointWith* property, we can separate the defined synset type from the others within the same group. In the case of *wordnet#Noun* class, we stated clearly that it is disjoint with *wordnet#Verb* and *wordnet#Adjective*. All the other restrictions included in the definition of *wordnet#Noun* class are related to properties and will be explained in the next section.

Similarly, the WordNet ontology implements class definitions for *wordnet#Verb*, *wordnet#Adverb*, *wordnet#Adjective* and *wordnet#AdjectiveSatellite*.

4.2 Definitions of Properties

Except for the *wordnet#wordForm* relation, all the properties defined in the WordNet OWL-ontology correspond to the WordNet^{TM} operators. The *wordnet#wordForm* relation is an object property that links a *wordnet#LexicalConcept* entity with one or more *wordnet#WordObject* individuals with the same meaning. The limitations over the relation's domain and range are specified by the *rdfs:domain* and *rdfs:range* predefined properties:

```
<owl:ObjectProperty rdf:ID="wordForm">
    <rdfs:domain rdf:resource="#LexicalConcept" />
    <rdfs:range rdf:resource="#WordObject" />
</owl:ObjectProperty>
```

There are two types of properties: semantic relations defined by binary relations between word meanings (that links two *wordnet#LexicalConcept*s in a meaningful way) and lexical relations between word forms (properties that link *wordnet#WordObject*s instances). From all the properties defined in our WordNet ontology, we mention *wordnet#hyponymOf* and its inverse *wordnet#hypernymOf*, *wordnet#xMeronym*-type and their corresponding inverse properties *wordnet#xHolonym*-type and *wordnet#similarTo* as semantic relations. All of them are object properties but every one has different restrictions that define it.

In the definition of each property, we can specify the corresponding inverse relation if it exists. For example, we can state that *wordnet#hypernymOf* is the inverse relation of the *wordnet#hyponymOf* property by using *owl:inverseOf* in the definition of *wordnet#hyponymOf*. Being a transitive relation, we define *wordnet#hyponymOf* as an *owl:TransitiveProperty*. The domain and range for *wordnet#hyponymOf* are restricted to nouns and verbs:

```
<owl:TransitiveProperty rdf:ID="hyponymOf">
    <rdfs:domain rdf:resource="#Nouns_and_Verbs"/>
    <rdfs:range rdf:resource="#Nouns_and_Verbs"/>
    <owl:inverseOf rdf:resource="#hypernymOf"/>
    <rdfs:comment>
        The hyponymOf relation specifies that
        the subject is a hyponym of the object.
        This relation holds for nouns and verbs.
    </rdfs:comment>
</owl:TransitiveProperty>
```

The domain and range restriction tell us that we can link any noun or verb with other noun or verb with the *wordnet#hyponymOf* relation. This means that we can use the *wordnet#hyponymOf* property to say that a noun is hyponym of a verb, which is not correct. To forbid this, the domain and range are restricted not only by the definition of the property, but also in the definition of the corresponding lexical concepts.

The *wordnet#Noun* class definition (and similarly the *wordnet#Verb*) contains a restriction defined for *wordnet#hyponymOf* property that states that if an individual of type *wordnet#Noun* is linked by *wordnet#hyponymOf* relation with another individual, the last one is also a *wordnet#Noun*. The *owl:allValuesFrom* relation gives us the possibility to restrict the range of a property for a specific domain:

```
<owl:Class rdf:ID="Noun">
...
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hyponymOf"/>
      <owl:allValuesFrom rdf:resource="#Noun"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
...
  </rdfs:subClassOf>
</owl:Class>
```

wordnet#antonymOf property is a symmetric property, defined as a relation from a *wordnet#WordObject* individual to another *wordnet#WordObject* individual:

```
<owl:SymmetricProperty rdf:ID="antonymOf">
    <rdfs:domain rdf:resource="#WordObject"/>
    <rdfs:range rdf:resource="#WordObject"/>
    <rdfs:comment>
        The antonymOf relation specifies that the subject
        is antonym of the object.
```

</rdfs:comment>
</owl:SymmetricProperty>

4.3 Stems' Ontology

The second OWL-ontology extends the WordNet ontology by defining the *StemObject* class and its properties. Inside the *stem#* namespace, this ontology creates a link between a lexical word and its stem form. The *stem#StemObject* class has no restriction over the type or format of the stem individuals.

The stem forms are generated by removing the commoner morphological and inflexional endings from words. We used the Porter stemming algorithm ([12], [9]) for normalizing the lexical words and to obtain the related stems. Because different words can provide the same stem form, a *stem#StemObject* individual groups multiple grammatical forms of the same word into a single entity. This is realized through the *stem#stemFromWord* property which links a *stem#StemObject* with a *stem#WordObject*:

```
<owl:ObjectProperty rdf:about="&stem;stemFromWord">
    <rdfs:domain rdf:resource="&stem;StemObject" />
    <rdfs:range rdf:resource="&wordnet;WordObject" />
</owl:ObjectProperty>
```

4.4 Documents' Ontology

The last ontology is obtained from a collection of articles from the Wall Street Journal. Represented by the *corpus#* namespace, it maps human-readable documents to a large text-database.

The classes and properties defined by this ontology are used as an indexing structure for the documents. Considering that a *stem#StemObject* individual is associated with a group of words, the *corpus#* ontology links every *stem#StemObject* instance with its container document. A container document contains at least one word associated with the considered *stem#StemObject* individual. These relations are defined through the *corpus#DocEntry* class that provides different properties for document indexing and manipulation. These relations are *corpus#doclink*, *corpus#freqv* and *corpus#pos1*.

The next paragraphs details the above properties and shows their OWL definitions.

4.4.1 The Word-Document Property

The mapping between a word's stemmed form and the word's container document is realized by the *corpus#stemToDoc* relation:

```
<owl:Class rdf:about="&corpus;DocEntry"/>
...
<owl:ObjectProperty rdf:about="&corpus;stemToDoc">
    <rdfs:domain rdf:resource="&stem;StemObject" />
    <rdfs:range rdf:resource="&corpus;DocEntry" />
</owl:ObjectProperty>
```

To each *wordnet#WordObject* individual in the WordNet ontology corresponds a *stem#StemObject* instance, possibly linked to a *corpus#DocEntry* individual. Hence, to each word in the document collection corresponds a *stem#StemObject* individual, possibly linked to an *wordnet#WordObject* individual from the WordNet ontology. In this way, we can use the WordNet as a reference for document processing. The stems that are not linked to both ontologies will be ignored by the inference engine.

4.4.2 The *corpus#DocEntry* Properties

Correlated with a *stem#stemObject* individual, a *corpus#DocEntry* instance comprises information about the document name through the *corpus#doclink* property:

```
<owl:ObjectProperty rdf:about="&corpus;doclink">
    <rdfs:domain rdf:resource="&corpus;DocEntry" />
    <rdfs:range rdf:resource="&owl;Thing" />
</owl:ObjectProperty>
```

Because most meaningful queries request information related to words and their containing documents, based on the properties defined until now, we extended our ontology with the *inDoc* relation that defines a direct link between words and documents. This is automatically obtained from the *stemFromWord*, *stemToDoc* and *doclink* relations.

Defined as datatype properties, *corpus#freqv* and *corpus#pos1* give statistical information relative to a specific word and one of its containing documents. The *corpus#freqv* relation is used for adding the frequency of the stem for a document in the knowledge base. The *corpus#pos1* gives the first position of the stem's word within the document:

```
<owl:DatatypeProperty rdf:about="&corpus;freqv">
    <rdfs:domain rdf:resource="&corpus;DocEntry" />
    <rdfs:range rdf:resource="&xsd;positiveInteger" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="&corpus;pos1">
    <rdfs:domain rdf:resource="&corpus;DocEntry" />
    <rdfs:range rdf:resource="&xsd;positiveInteger" />
</owl:DatatypeProperty>
```

4.5 Statistical Information about the WordNet Ontology

The WordNet ontology obtained from WordNet^{TM} lexical database counts almost 1 million statements in N-triple form. It defines more than 30 definitions of classes and properties and more than 700,000 instances and relations. This ontology supports the extensions provided by the *stem#* and *corpus#* ontologies described above.

The *corpus#* ontology described in the previous section holds more than 13 million instances of the *corpus#DocEntry* class that links WordNet lexical words with documents from Wall Street Journal Collection.

All three ontologies are joined into one ontology, keeping their original namespaces. The final ontology comprises a

total of 44 definitions of classes and properties, instantiated by 14.5 million individuals and more than 65 million property instances. The whole system is defined by more than 82 million statements in the N-triple form.

The ontology described in this chapter should be seen as an illustration of what can be realized. It is not the intention to propose this as a general solution. Other classes, properties and instances could be added as needed.

5. EXPERIMENTS

For our experiments we used the WordNet ontology described in the previous section and the Wall Street Journal Collection) (TREC 4). In the next subsection we describe different queries used in our tests. The result obtained for each query will be presented in the section 5.2.

5.1 The Queries

To illustrate the power of our system we present a large spectrum of different queries reaching from simple keyword-based queries out to complicated semantically driven ones.

5.1.1 Simple Queries

For our test cases we consider the most often used queries in an information retrieval system [6]. The simplest query that we experimented in our system, referred to as QC1, has the form *"find all documents that contains a specified word"*. In our tests the word instance specified in the query was chosen such that different-sized answers will be generated. A very small number of documents is obtained for words like *wordnet#Pseudomonas* that are used in domain-specific concepts. In contrast, words like *wordnet#one* or *wordnet#million* are words that appears in a large set of documents.

The next queries are a generalization of the previous ones. They depend on multiples instances for the word specified in the request, all the words being combined by a conjunction. This increases the difficulty level of the query but decreases the size of the results. In the case of two-word constraint, referred to as QC2, the query has the form *"find all documents that contains two specified words W1 and W2"*. The words from the QC2-type queries were instantiated with different combination of the words used in the first type of queries.

5.1.2 Ontological Queries

Because most of the inferences involve semantic computation, we tested our system using queries that require searches through the WordNet properties.

The QS1-type queries have the form *"find all documents that contains a specified word or one of its synonyms"*. Using the classes and relations definitions from our WordNet ontology and the *#inDoc* property defined previously, these type of queries are translated into:

$$\begin{aligned}&\forall\ \mathrm{D}\ ;\ \#inDoc(\mathrm{W,D}) \vee \\ &\quad (\ \exists\ \mathrm{C}\ ;\ wordnet\#wordForm(\mathrm{C,W}) \wedge \\ &\qquad \exists\ \mathrm{C2}\ ;\ wordnet\#similarTo(\mathrm{C,C2}) \wedge \\ &\qquad \exists\ \mathrm{W2}\ ;\ wordnet\#wordForm(\mathrm{C2,W2}) \wedge \\ &\qquad \#inDoc(\mathrm{W2,D})\ )\end{aligned}$$

More complicated queries that we propose are based on transitive closure computation on a specific relation or are doing a general search over multiple relations. These are very expensive queries because the whole ontology has to be used in order to compute the results.

We considered the following query identified as QTC_1: *"find all documents that talk about any kind of animal"*. Because the WordNet ontology defines *"is a kind of"* relation by *hyponymOf* predicate, the previous query is equivalent with *"find all documents X, such that X contains Y and Y hyponymOf animal"*. The property *hyponymOf* is a transitive relation, so we have to compute the transitive closure on the *hyponymOf* relation starting from the *animal* individual. This can be formalized as follows:

$$\begin{aligned}&\forall\ \mathrm{D}\ ;\ \exists\ \mathrm{C}\ ;\ wordnet\#wordForm(\mathrm{C,W}) \wedge \\ &\quad \exists\ \mathrm{C1}\ ;\ wordnet\#hyponymOf*(\mathrm{C,C1}) \wedge \\ &\quad \exists\ \mathrm{W1}\ ;\ wordnet\#wordForm(\mathrm{C1,W1}) \wedge \\ &\quad \#inDoc(\mathrm{W1,\ D})\ .\end{aligned}$$

where *wordnet#hyponymOf** denotes the transitive closure over the *wordnet#hyponymOf* relation. Similarly, we considered the same query using *wordnet#plant* as the starting point.

Another type of query tested with our system, referred to as QTC_2, implies a search through a long chain of relations in order to obtain the answer. Literally, it has the form *"find all documents that contain a word as well as at least one of its antonyms"*. Formally, using only the definitions from the imported ontology, the query is:

$$\begin{aligned}&\forall\ (\mathrm{D,W,A})\ ; \\ &\quad wordnet\#antonymOf(\mathrm{A,W}) \wedge \\ &\quad \exists\ \mathrm{SW}\ ;\ stem\#stemFromWord(\mathrm{SW,W}) \wedge \\ &\quad \exists\ \mathrm{SW\_DE}\ ;\ stem\#stemToDoc(\mathrm{SW,\ SW\_DE}) \wedge \\ &\quad corpus\#doclink(\mathrm{SW\_DE,\ D}) \wedge \\ &\quad \exists\ \mathrm{SA}\ ;\ stem\#stemFromWord(\mathrm{SA,A}) \wedge \\ &\quad \exists\ \mathrm{SA\_DE}\ ;\ stem\#stemToDoc(\mathrm{SA,\ SA\_DE}) \wedge \\ &\quad corpus\#doclink(\mathrm{SW\_DA,\ D})\ .\end{aligned}$$

Because the *inDoc* relation is equivalent with the expression $stemFromWord(\mathrm{s,w}) \wedge stemToDoc(\mathrm{s,\ d}) \wedge doclink(\mathrm{d,\ c})$ the previous query is simplified to the following form:

$$\begin{aligned}&\forall\ (\mathrm{D,W,A})\ ; \\ &\quad wordnet\#antonymOf(\mathrm{A,W}) \wedge \\ &\quad inDoc(\mathrm{W,D}) \wedge \\ &\quad inDoc(\mathrm{A,D})\ .\end{aligned}$$

The results obtained for all the described queries are presented in the next section.

5.2 Results

For all our tests we used a 660 MHz Pentium III with 320 MB RAM running the SuSE Linux 8.0 operating system.

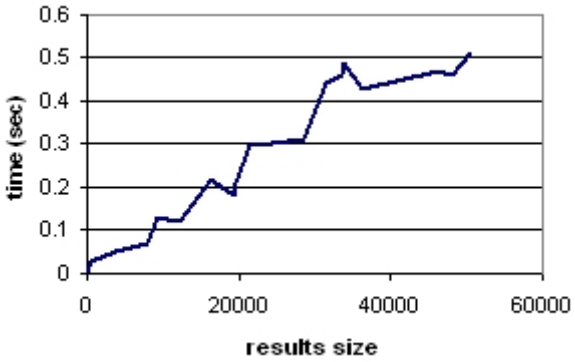

Figure 2: Results for QC1-type queries

The words used in the QC1-type queries were chosen such that they highlight the correlation between elapsed time and the size of the results. As it can be seen in the Figure 2, the inference for rare used words gives the results in no time. In the case of currently used words, the query is solved in linear time. For example, the QC1 query expressed for *wordnet#Pseudomonas* word gives three documents instantly. The same query instantiated for word *wordnet#one* is solved in 0.37 seconds and gives more than 36 thousand documents. Because the stemming process does not remove the stop words, the common prepositions and conjunctions generate the largest results.

The word instances from the QC2-type queries were selected based on the results obtained for the previous ones. The pairs of words that are instantiated in the QC2 queries form all possible combination between words used in the QC1 queries that are generating results of small, medium and large size. We will exemplify the results obtained for QC2 queries using any word from *physical*, *president*, *million*, *in* in combination with any word from *chariness*, *rose*, *marketing*, *new*. Both lists of words are sorted in ascending order relative to the result sizes obtained for the QC1-type queries. In the table 1 we illustrate the results for QC2 queries instantiated with pairs of words mentioned above. These times are relatively high because our system was not optimized for simple queries. Table 2 shows the size of the results obtained.

	physical	*president*	*million*	*in*
chariness	0	0	0	0
rose	0.02	0.19	0.21	0.24
marketing	0.01	0.56	0.5	0.56
new	0.02	0.88	0.77	1.99

Table 1: Elapsed times for QC2-type queries

The ontological queries (section 5.1.2) involve semantic computation in the imported ontology. One of the word instances used in the case of the QS1-type queries is *wordnet#beautiful*. In the WordNet ontology this word appears with three senses corresponding to the three *wordnet#LexicalConcept* individuals. Each of them is linked

	physical	*president*	*million*	*in*
chariness	0	3	5	6
rose	112	2257	5657	7667
marketing	232	6757	11709	16952
new	268	13586	18422	30072

Table 2: Result sizes for QC2-type queries

to similar concepts through the *wordnet#similarTo* relation. Finally there are 32 unique synonyms related to the word *wordnet#beautiful*. The QS1 query run in the case of word *wordnet#beautiful* took only 0.18 seconds, the answer comprising more than 10 thousand documents.

The first part of the QTC_1 query, namely *"finding all kinds of animals in WordNet"*, cannot be handled by the WordNet application, which gives the answer: <<`Search too large. Narrow search and try again...`>>. The reasoner of knOWLer provides the answer of the complete query in 2.3 seconds, based mainly on recursive interrogations over the Storage Repository, without using the cache mechanism described in section 3.1. We retrieved almost 38 thousand documents as result of the query. The same type of query run for *plant* concept took 2.28 seconds and retrieved around 36 thousand documents.

The QTC_2 query is a general request. It has to look over all possible combinations in order to find the answer. In our system, it took six minutes, giving more than 500 thousand document-word-antonym tuples associated with around 33 thousand distinct documents.

As we can see from the above results, the knOWLer system can be successfully used for document indexing using the semantics provided by an ontology language. The knOWLer system is still under development and the results are not conclusive at this stage, however the performance achieved during the tests conducted is very encouraging.

6. CONCLUSIONS AND FUTURE WORK

We presented in this paper an information management system called knOWLer. The special characteristic of this system is its scalability.

Scalability is made possible by using persistency at both storage and reasoning levels, allowing knOWLer to handle enterprise-sized information bases. Of course, its performance highly depends on the scalability of the specific implementation of the Storage Module.

knOWLer is as far as we know one of the first systems that can handle the size of ontologies described in this paper, i.e. ontologies composed of 100 Million statements. This is necessary for providing reasonable support for large-scale information retrieval systems. Furthermore we have shown that knOWLer can respond to simple but also very complex ontology-based queries on commodity hardware with very little time consumption. Simple keyword based queries as well as complex queries based on WordNet properties such as synonyms and antonyms were answered in fractions of a second, more complex queries using transitive closure op-

eration took a couple of seconds on a modestly equipped PC.

We believe that Ontologies will play a crucial role in enabling the processing and sharing of information between local or distributed applications. Using this information, query systems can provide more accurate responses than are possible with the search engines available on the Web.

In future series of tests we would like to analyze how IR systems could benefit form knOWLer with respect in improvement of their classical evaluation criteria such as recall and precision. To achieve this, specific IR techniques can be integrated (e.g. a desambiguation step for the identification of the proper synset). Furthermore we would like to integrate other ontolgies aside WordNet to be able to reason from different point of view. Our current results allow us to assume that we will be able to handle ontologies compose of up to 1 Billion statements. Therefore we would also be able to integrate a much larger document basis.

7. REFERENCES

[1] S. Bechhofer, I. Horrocks, et al. A proposal for a description logic interface. In P. Lambrix, A. Borgida, M. Lenzerini, R. Möller, and P. Patel-Schneider, editors, *Proceedings of the International Workshop on Description Logics (DL'99)*, pages 33–36, 1999.

[2] A. Borgida, R. J. Brachman, et al. Classic: A structural data model for objects. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pages 59–67, 1989.

[3] J. Broekstra, M. Klein, et al. Enabling knowledge representation on the Web by extending RDF Schema. *Computer Networks (Amsterdam, Netherlands: 1999)*, 39(5):609–634, Aug. 2002.

[4] M. Dean, D. Connolly, et al. Owl web ontology language reference. World Wide Web Consortium, W3C Working Draft 31 March 2003, http://www.w3.org/TR/2002/WD-owl-ref-20021112/, 2002.

[5] S. Decker, D. Fensel, et al. Knowledge representation on the web. In B. F. and S. U., editors, *Proceedings of the 2000 International Workshop on Description Logics (DL2000)*, Aachen, Germany, 2000.

[6] J. Han and M. Kamber. *Data Mining. Concepts and Techniques*. Morgan Kaufmann Publishers, 2001.

[7] J. Heflin and J. Hendler. Dynamic ontologies on the web. In *Proceedings of the 7th Conference on Artificial Intelligence (AAAI-00) and of the 12th Conference on Innovative Applications of Artificial Intelligence (IAAI-00)*, pages 443–449, Menlo Park, CA, July 30–3 2000. AAAI Press.

[8] J. Heflin, J. Hendler, and S. Luke. Applying ontology to the web: A case study. In *International Work-Conference on Artificial and Natural Neural Networks, IWANN'99*, 1999.

[9] K. S. Jones and P. Willet. *Readings in Information Retrieval*. Morgan Kaufmann, San Francisco, 1997.

[10] M. Klein. Supporting evolving ontologies on the Internet. *Lecture Notes in Computer Science*, 2490:597, 2002.

[11] B. McBride. Jena: Implementing the rdf model and syntax specification. In *Semantic Web Workshop*, 2001.

[12] M. F. Porter. An algorithm for suffix stripping. In *Program*, volume 14/3, pages 130–137, 1980.

[13] S. J. Russell and P. Norvig. *Artificial Intelligence. A Modern Approach*. AI. Prentice_Hall, Englewood Cliffs, 1995.

[14] M. Smith, D. McGuinness, et al. Owl web ontology language guide. World Wide Web Consortium, W3C Working Draft 31 March 2003, http://www.w3.org/TR/2002/WD-owl-guide-20021104/, 2003.

[15] K. Stoffel, M. Taylor, and J. Hendler. Efficient management of very large ontologies. In *Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI-97/IAAI-97)*, pages 442–447, Menlo Park, July 27–31 1997. AAAI Press.

[16] E. M. Voorhees. Using wordnet to disambiguate word senses for text retrieval. In R. Korfhage, E. Rasmussen, and P. Willett, editors, *Proceesings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 171–180. ACM Press, 1993. Pittsburgh, PA USA, June 27 - July 1.