Scalable Ontology Implementation Based on knOWLer

Iulian Ciorăscu
(University of Neuchâtel, Switzerland
iulian.ciorascu@unine.ch)

Claudia Ciorăscu
(University of Neuchâtel, Switzerland
claudia.ciorascu@unine.ch)

Kilian Stoffel
(University of Neuchâtel, Switzerland
kilian.stoffel@unine.ch)

Abstract: In this paper we describe the knOWLer System - an ontology-based information system. The knOWLer project focuses on the aspects of making ontologies available to real world applications. Therefore, a lot of effort was made to create a stable and scalable system that is able to manage millions of ontological statements. In the paper, we will detail the basic architecture of the system and illustrate the performance of the system based on some benchmark queries encountered in a wide range of applications. The results are compared with those obtained from another state of the art systems namely Jena. We were able to show that knOWLer was much faster for the given ontologies and queries.

1 Introduction

Over the last decade, ontology-based information systems gained increasingly in importance. At the beginning, lot of effort went into the analysis of different languages to express ontologies. With the evolving standards for ontologies especially in the Semantic Web effort, led by the W3C, the issue about language became less important. Thus, it was possible to focus on the process of developing ontology-based systems.

This paper presents such a system called knOWLer. knOWLer was first intended to be used in medical information framework especially focusing on tariffication problems such as validating bills and analyzing the neutrality of cost in health-care. Later, because of its flexibility, it was easily integrated in applications in financial domains for specifying, analyzing and optimizing different investment strategies or in governmental systems for verifying conformity of a given behavior with a predefined regulation. All these applications have one common property: they are based on very large reference systems that are expressed in form of ontologies (described by an ontological language such as OWL [3]) linked to the large structured or unstructured databases that have to be analyzed with respect to the predefined reference systems.

As most of the information in these applications underlies strict data-protection rules we created an artificial domain in order to be able to make the comparisons of the different systems. The domain was created by building an ontology based on the WordNet^{TM} lexical reference system [1]. Several versions of this ontology can be found on the Web. We found most of them incomplete and therefore we built our own WordNet ontology, which can be found on the knOWLer web site[2]. Furthermore we extended the ontology by unstructured text documents from the Wall Street Journal Collection. All the details about this domain are given later.

The remainder of the paper is structured as follows. The first section overviews a subset of the relevant existing similar applications. The next sections focuses on the technical aspects and capabilities of our system. After a description of the OWL-ontology used for testing the system, a series of simple and complex queries will be presented to analyze the performance of the system by comparison with other systems.

2 Ontology-based Systems

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 [6].

Defined as an extension over HTML, SHOE [6] was the first attempt to annotate the web documents with machine processable data content. Focusing on the modeling and sharing knowledge databases over the web, specific languages were created. Most of these languages are based on RDF (e.g. DAML+OIL, OWL) and are currently undergoing a standardization process.

There are different tools for browsing or editing ontologies described in these languages. Most of them give support for development and maintenance of small ontologies with minimal ontological constructs. We can mention OntoEdit (AIFB, University of Karlsruhe) that provides graphical support, JOE (Java Ontology Editor - University of South Carolina) as a web browser or Chimaera (Stanford University) for maintaining distributed ontologies on the web.

There are few editors such as OILEd [2], Kaon [10] or Protégé [11] that extend the expressive power of the ontology languages (using OIL or DAML) and provide also support for reasoning over the ontologies.

[1] Wordnet^{TM} online lexical database is available at *http://www.cogsci.princeton.edu/~wn/*

[2] *http://www.unine.ch/info/KID*

Some systems provide reasoning support using description logic systems such as FACT [1] and Classic [4] and many others.

With the availability of useable tools, the number of ontologies increased rapidly. The need for storing and managing large knowledge bases became imminent. In a first phase a large number of systems were developed with the main purpose of storing and retrieving ontologies. A very popular system providing persistent storage is Jena [9]. It provides an interface for manipulating knowledge bases and can easily be integrated in other systems (Triple [14]) or wrapped by other applications. Another similar system is Sesame [5] used usually for web applications. As quering mechanisms several languages were proposed, most of them being essentially extensions of SQL (SiLRI[14], RQL [8]).

Parka-DB [16] is a frame-based system developed for manipulating very large knowledge bases. Starting from its ideas, we developed 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. This is a system well suited for intranet-based applications that requires semantic reasoning over large data collections.

To achieve the goal of expressivity, our system implements an extension of the OWL Lite language (subset of OWL [12]). Built upon the foundations of the DAML+OIL specification, OWL [3] 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 The knOWLer System Architecture

The knOWLer system is an innovative ontology-based system 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 towards modern ontology languages but is still providing support for traditional frame-based systems. Developed by W3C, the OWL language consists of three overlapped layers [15]. 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 preserves 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. The resulting language will be referred as Extended OWL Lite in this paper. The reasoning capabilities provided by knOWLer System follow the semantics defined by the OWL Lite language.

The architecture of our ontology-based information management system consists in four distinct modules (Figure 1): knOWLer kernel, Storage Interface, Import/Export Interface and User Interface. All modules are linked through the kernel, which is the main component.

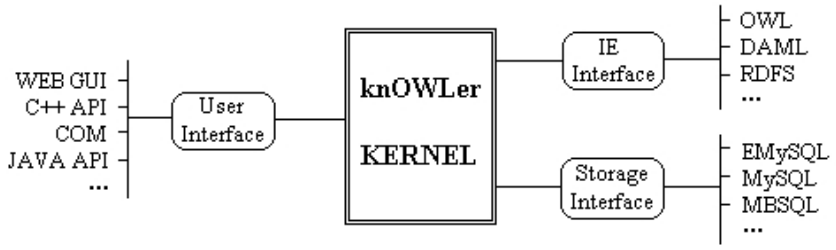

Figure 1: knOWLer's Architecture

3.1 The knOWLer Kernel

The central component of the knOWLer system is its kernel. It provides the functionality required for manipulating the structure and the semantics defined by an ontology. It also has complete reasoning for Extended OWL Lite as well as for the equivalent subset of DAML+OIL. The kernel also supports RDF/RDFS but the reasoning will assume Extended OWL Lite restrictions. A detailed description of how this is achieved can be found as a working draft on the knOWLer web site.

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. But one of the most important features is that it uses a C-like syntax interpreted language to handle even the most complex queries inside the server. The scripting language integrates useful features like regular expressions, efficient and easy to use data structures (arrays, sets, maps), functional support, Object Oriented programming, etc. The access to the data is implemented using the knOWLer API directly from the scripts. Using scripts as queries is a powerful mechanism

for distributed knowledge bases since all the computation can be done 'on-site', which can reduce drastically the network traffic and thus decrease the query time. Libraries of functions using the script language can be created and become an integral part of an ontology, which can lead to interesting results (i.e. using procedures as properties, etc).

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 cache has a visible impact when importing data, because it reduces the overhead produced by the increased number of small queries.

3.2 The knOWLer Modules

The knOWLer kernel uses external modules for importing/ exporting ontologies, storing data and communicating with user and/or client applications. Usually there are multiple modules that implement the same interface with different functionality.

The Import/Export (IE) Interface provides the full functionality for importing/exporting DAML, OWL and RDF/RDFS ontologies. The input data is converted to the internal representation and it is passed to the knOWLer kernel. For the moment, only XML/RDF and Ntriples notations are recognized. Other notations can be added by implementing the corresponding knOWLer Import / Export 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, MySQL-client and an own btree implementation of ISAM files.

OWL-specific syntactic checks 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 simple RDF parser cannot do these kinds of checks, OWL extensions being necessary. After importing data finishes, more complex checks for OWL compliance are executed.

The User Interface gives access to the importing / exporting and reasoning functionalities of the system. Client applications can access the system through its API, available in C++ and Java. Either way, they can choose to use directly the API calls or use the scripting language to perform queries. Although knOWLer is intended to be used mainly as a server, a demo client with web interface is available. Perl and Python support is planned for the near future.

4 Test Ontology

To show the capabilities of our system we generated a very large ontology from public data. The usefulness of this ontology was inspired by real world applications and describes a possible indexing mechanism for information retrieval systems. It comprises three different ontologies connected through the definitions of classes and properties. Each ontology was build in the OWL language in different 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 80 million statements in the N-triple form. The first ontology represents the content of the WordNet^{TM} lexical database. It gives support for the other two ontologies that provide definitions for stem and document manipulations.

4.1 WordNet OWL Ontology

Obtained from the data supplied by WordNet^{TM} 1.7.1 lexical database, the first ontology counts almost 1 million statements in N-triple form. Defined in the namespace *wordnet#*, the WordNet OWL-ontology consists of more than 30 definitions of classes and properties and more than 700,000 instances and relations. The full version of WordNet ontology in OWL language, including the definitions and instances, can be found on the knOWLer Project site.

There are two top classes defined in this ontology: *wordnet#WordObject* class that represents lexical words and *wordnet#LexicalConcept* whose extent comprises all WordNet synsets. Each leaf class corresponds to a syntactic category (*wordnet#Noun*, *wordnet#Verb*, etc.). They are derived directly or indirectly from the *wordnet#LexicalConcept* class, being pairwise disjoint with each other.

All the properties defined in the WordNet OWL-ontology correspond to the WordNet^{TM} operators. The *wordnet#wordForm* object property groups one or more *wordnet#WordObject* individuals under the same lexical concept. There are two types of object 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). Some of them can have an inverse property (e.g. *wordnet#hypernymOf* inverse of *wordnet#hypernymOf*), can be transitive (e.g. *wordnet#entailsTo*) and/or can be symmetric (e.g. *wordnet#similarTo*).

As an example, the *wordnet#hyponymOf* relation is defined as a transitive object property and inverse of *wordnet#hypernymOf* relation. Its domain and range are restricted only to nouns and verbs, given the possibility to link any noun or verb with other noun or verb with the *wordnet#hyponymOf* relation.

```
<owl:TransitiveProperty rdf:about="&wordnet;hyponymOf">
    <rdfs:domain rdf:resource="&wordnet;Nouns_and_Verbs"/>
    <rdfs:range rdf:resource="&wordnet;Nouns_and_Verbs"/>
    <owl:inverseOf rdf:resource="&wordnet;hypernymOf"/>
</owl:TransitiveProperty>
```

The *wordnet#Noun* class is a subclass of *wordnet#Nouns_and_Verbs* and also a subclass of *wordnet#Nouns_and_Adjectives*, both of them being derived from *wordnet#LexicalConcept*. Using *owl:disjointWith* property, we stated clearly that it is disjoint with *wordnet#Verb* and *wordnet#Adjective*. To forbid the *wordnet#hyponymOf* -type relation between individuals of different syntactic categories, a new restriction over the domain and range of the *wordnet#hyponymOf* property is defined locally for each class.

```
<owl:Class rdf:about="&wordnet;Noun">
    <rdfs:subClassOf rdf:resource="&wordnet;Nouns_and_Verbs"/>
    <rdfs:subClassOf rdf:resource="&wordnet;Nouns_and_Adjectives"/>
    <owl:disjointWith rdf:resource="&wordnet;Verb" />
    <owl:disjointWith rdf:resource="&wordnet;Adjective" />
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&wordnet;hyponymOf" />
            <owl:allValuesFrom rdf:resource="&wordnet;Noun" />
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&wordnet;attributeRel" />
            <owl:allValuesFrom rdf:resource="&wordnet;Adjective" />
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

4.2 The 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 of type *stem#StemObject*.

The stem forms are generated by removing the commoner morphological and inflexional endings from words. We used the Porter stemming algorithm ([13], [7]) 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 >
```

4.3 The 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. It brings more than 13 million instances of the *corpus#DocEntry* class that link WordNet lexical words with documents from the collection. The classes and properties defined by this ontology can be used as an indexing structure for the documents. It is not the intention to propose this as a solution for document indexing, but as an illustration of how the knOWLer system can be used.

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. The statistical knowledge associated to a *corpus#DocEntry* consists of information about document reference (induced by *corpus#doclink* object property) or the frequency and first position in the related document by the definitions of *corpus#freqv* and *corpus#pos1* datatype properties:

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

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

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 *corpus#wordToDoc* relation that defines a direct

link between words and documents. This is automatically obtained from the *corpus#stemFromWord*, *corpus#stemToDoc* and *corpus#doclink* relations.

The mapping between stemmed forms and the word's container documents is realized through the instances of the *corpus#DocEntry* class. 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 a *wordnet#WordObject* individual from the WordNet ontology. In this way, we can use the WordNet as a reference for document processing.

5 Experiments

Using the ontology described in the previous chapter we conducted a series of tests to see how our system handles such big ontologies. The queries we present here are representative for the queries we saw while the knOWLer system was used in the different application domains.

As Jena is used as the underlying system in many other implementations of analogous systems (e.g. TRIPLE) we decided to make comparative tests using Jena in order to have a baseline comparison. We used Jena 1.6.1 using MySQL in the following tests. We tried to load the previously described ontology into Jena. However we stopped the process, as after several days the loading was still not finished (knOWLer finished loading the 80 million statement ontology in less than a day). Instead we decided to use only the WordNet ontology (700.000 statements) and reduce the comparison tests to queries that don't use the Stem and Document ontologies. Although this is not a perfect testbed, it is good enough to give an idea of how the two systems behave.

All queries were provided to knOWLer system in the scripting query language format, while for Jena we wrote Java programs to handle the queries. For all our tests we used a 660 MHz Pentium III with 320 MB RAM and SCSI harddrive running the SuSE Linux 8.0 operating system.

5.1 knOWLer Evaluation

The simplest queries that we executed have the form *"find all documents that contains one or more specified words"*. Although these queries are just direct data extraction with no semantic reasoning, they can be used to establish the baseline performance of our system. However they can't be used in comparison with the other system, as we were not able to load the whole ontology into this system. The predefined words used by these queries were chosen in such a way

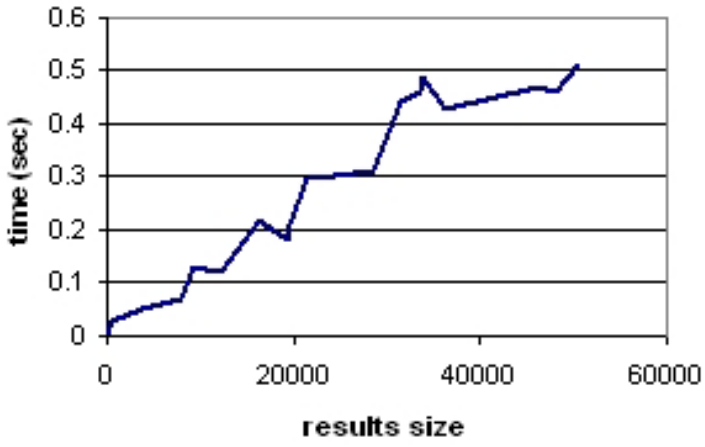

Figure 2: Results for one word based queries

that the results highlight the relation between response time and results size. As it can be seen in the Figure 2, the query is essentially solved in linear time.

In the second test we used queries looking for documents containing two given words. As word cases for these queries we used any word from {*physical*, *president*, *million*, *in*} list 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 in the previous type of queries. In the Table 1 we exemplify the time results obtained by the this type of query for each considered pair of words. The corresponding sizes of the results are shown in Table 2. These results are not a surprise, the complexity of this type of queries being $O(n \log n)$.

	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: Time results for two word based queries

	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 two word based queries

A more complex query is *"find all documents that contains a specified word or one of its synonyms"*. Using the class and relation definitions from our WordNet ontology and the *corpus#wordToDoc* property defined previously, these type of queries are translated into:

```
SELECT ?D
  WHERE  ( "word", <corpus:wordToDoc>, ?D )
union
SELECT ?D
  WHERE  ( ?C, <wordnet:wordForm>, "word" ),
         ( ?C,  <wordnet:similarTo>, ?C2 ),
         ( ?C2, <wordnet:wordForm>,  ?W2 ),
         ( ?W2, <corpus:wordToDoc>,  ?D  )
```

One of the "word" instances used for this query was *wordnet#beautiful*. In the WordNet ontology this word appears with three senses corresponding to the three *wordnet#LexicalConcept* individuals. Each of them is linked to similar concepts through the *wordnet#similarTo* relation. Finally there are 32 unique synonyms related to the word *wordnet#beautiful*. The query run in this case took only 0.18 seconds, the answer comprising more than 10 thousand documents.

Another type of query tested with our system has the form *"find all documents that contain a word as well as at least one of its antonyms"*.

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

5.2 Comparative Tests

Although knOWLer was able to handle the following queries over the complete ontology (of 80 millions statements), to be able to compare it with Jena we had to reduce the ontology to the WordNet part (only 0.7 millions statements).

All these queries are making use of transitive closure computation on a specific relation.

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

```
SELECT ?D
  WHERE  ( ?C, <wordnet:wordForm>, <wordnet:animal> ),
         ( ?C,  <wordnet:hyponymOf>*, ?C1 ),
         ( ?C1, <wordnet:wordForm>,  ?W1 ),
         ( ?W1, <corpus:wordToDoc>,  ?D  )
```

where * denotes the transitive closure over the specified relation.

This type of query is difficult to be handled by some of the existing systems. The reasoner of knOWLer provides the answer of the complete query in 2.3 seconds, based mainly on sequential 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 query using *wordnet#plant* as the starting point took 2.28 seconds and retrieved around 36 thousand documents.

In order to compare the two systems we simplified the query to *"find all kinds of animal"*:

```
SELECT ?W1
  WHERE  ( ?C, <wordnet:wordForm>, <wordnet:animal> ),
         ( ?C,  <wordnet:hyponymOf>*, ?C1 ),
         ( ?C1, <wordnet:wordForm>,  ?W1 )
```

Using Jena API, we translated the query into a Jena program that runs sequential RDQL requests. The query generated 3973 words in 26 seconds. The same query run for *<wordnet:plant>* generated 4829 words in 33 seconds. In the case of knOWLer, each query took around one second.

A more interesting request is to find the pairs of words that are in the same time antonyms and directly or indirectly synonyms. Using our scripting query language we had the possibility to optimize the query by sequentially creating synonym equivalence classes of concepts. For every class we searched for antonyms that belong to that class. The query resulted in 22 distinct pairs of words and finished in 98 seconds. The similar algorithm implemented in Java for Jena gave the answer in more than 2000 seconds.

Query description	knOWLer System (sec)	Jena-based System (sec)
"animal"	1	26
"plant"	1.1	33
synonyms / antonyms	98	2000

Table 3: Comparative results

The overall results are presented in Table 3, where we can see that knOWLer is more than 20 times faster than Jena for the given setting.

6 Conclusions and Future Work

We presented in this paper an ontology 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 is highly dependent on the scalability of the specific implementation of the Storage Module. We have shown that its base implementation relying on a RDBMS (MySQL) is much faster than other comparable systems. The speed of the system can be further increased by replacing the generic RDBMS by our own special purpose DBS.

The future developments will be directed towards different paths:

- portability and availability: making knOWLer available on other platforms and Operating Systems like MacOS/X, Solaris, FreeBSD etc. Also implementing client support from Perl, Pyton, etc, and the Storage Module for other RDBMSes.
- implementing a multi-user secure environment, with transactional and versioning support.
- increasing scalability by implementing a distributed version of knOWLer system.

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. S. Bechhofer, I. Horrocks, et al. OilEd: A Reason-able Ontology Editor for the Semantic Web. *Lecture Notes in Computer Science*, 2174:396–408, 2001.
3. S. Bechhofer, F. van Harmelen, et al. OWL Web Ontology Language Reference, W3C Candidate Recommendation. World Wide Web Consortium, http://www.w3.org/TR/2003/CR-owl-ref-20030818/, August 2003.
4. 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.
5. D. Fensel, F. van Harmelen, et al. On-to-knowledge: Ontology-based tools for knowledge management. In *Proceedings of the eBusiness and eWork 2000 (EMMSEC 2000) Conference*, Madrid, Spain, 2000.
6. 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.
7. K. S. Jones and P. Willet. *Readings in Information Retrieval*. Morgan Kaufmann, San Francisco, 1997.
8. D. P. G. Karvounarakis and V. Christophides. Querying Semistructured (Meta) Data and Schemas on the Web: The case of RDF & RDFS. Technical Report no. 269, ICS-FORTH, Heraklion, Crete, Greece, Feb. 2000.
9. B. McBride. Jena: Implementing the RDF Model and Syntax Specification. In *Semantic Web Workshop*, 2001.
10. B. Motik, A. Maedche, and R. Volz. A Conceptual Modeling Approach for building semantics-driven enterprise applications. In *Proceedings of the First International Conference on Ontologies, Databases and Application of Semantics (ODBASE-2002)*, LNAI, California, USA, 2002. Springer.
11. N. F. Noy, M. Sintek, et al. Creating Semantic Web Contents with Protégé—2000. *IEEE Intelligent Systems*, 16(2):60–71, 2001.
12. P. F. Patel-Schneider, P. Hayes, and I. Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax. W3C Candidate Recommendation. World Wide Web Consortium, http://www.w3.org/TR/2003/CR-owl-semantics-20030818/, August 2003.
13. M. F. Porter. An algorithm for suffix stripping. In *Program*, volume 14/3, pages 130–137, 1980.
14. M. Sintek and S. Decker. TRIPLE—A Query, Inference, and Transformation Language for the Semantic Web. In *International Semantic Web Conference (ISWC)*, Sardinia, June 2002.
15. M. K. Smith, C. Welty, and D. L. McGuinness. OWL Web Ontology Language Guide. W3C Candidate Recommendation, http://www.w3.org/TR/2003/CR-owl-guide-20030818/, August 2003.
16. 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.