Characterisation of Knowledge Bases[1]

Derek Sleeman, Yi Zhang, Wamberto Vasconcelos
Department of Computing Science,
University of Aberdeen,
Aberdeen, AB24 3UE, Scotland, UK
Email: {dsleeman, yzhang, wvasconc}@ csd.abdn.ac.uk

Abstract

The process of determining the principal topic of a Knowledge base (KB), and whether it conforms to a set of user-defined constraints, are important steps in the reuse of Knowledge Bases. We refer to these steps as the process of characterization of a Knowledge Base. **Identify-Knowledge-Base (IKB)** is a tool, which suggests the principal topic(s) addressed by the Knowledge Base. It matches concepts extracted from a particular knowledge base against some reference taxonomy, where the taxonomy can be pre-stored or extracted from ontologies which are either stored on the local machine or are assessable through the WWW. The 'most specific' super-concept subsuming these concepts is said to be the principal topic of the knowledge base. Additionally, a series of **filters**, which check if a KB has particular characteristics have been implemented. This paper describes both the Identify-Knowledge Base system and these filters. Some empirical studies of IKB and the filters with a range of problems are also reported.

1. Introduction

Reuse of Knowledge Bases is a promising area in Knowledge Technologies [1]. Nowadays, researchers are focusing on how to reuse existing Knowledge Bases for further applications [2]. Such requests for reuse are often specified as a Knowledge Base (KB) characterisation problem:

Require knowledge base on topic T, conforming to set of constraints C [2].

To respond to such requests we need to:

- Decide what is the principal topic (T) of a given knowledge base.
- Decide whether a KB conforms to certain constraints C.

[1] This work is part of the Advanced Knowledge Technology (AKT) project, which is funded by EPSRC, [1]. The IKB system incorporates the ExtrAKT system [3, 4, 5] and interfaces with the Edinburgh Knowledge Broker [6, 7] which were built by the other members of the AKT consortium

In this report, we discuss these two processes in detail. In section 2 to section 5, IKB is introduced. In section 6 and section 7, we discuss the filters that help users find KBs which match their requirements. In section 8, we discuss possible further work.

2. Determining the principal topic of a KB

In order to reuse a Knowledge Base, we need the ability to identify its topic (T). One way to identify the KB's topic is to:

1. Extract concepts from the knowledge base.
2. Match these concepts against a reference taxonomy, which defines a particular domain of interest. The KB's principal topic is given by the 'most specific' super-concept, which subsumes these concepts.
3. If a null result is obtained with a particular taxonomy, then repeat the last 2 steps with other taxonomies thought to be of interest.

The **ExtrAKT** system from Edinburgh [2] can analyse a Prolog knowledge base, and extracts all the KB's predicates. The IKB system then takes the predicates extracted from the Prolog Knowledge base and compares them with a pre-defined reference taxonomy to identify the KB's principal topic. Below we give a simple example using a food taxonomy. Suppose we already have the taxonomy depicted below:

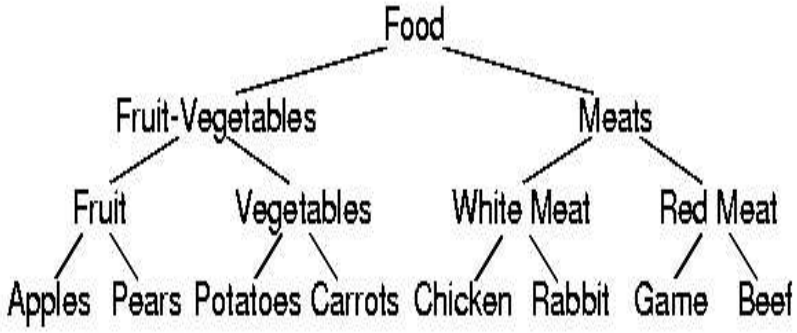

Figure-1 Food Taxonomy showing different kinds of food

If the concepts {Apples, Pears} are extracted (by ExtrAKT) and passed to the IKB system, the system would suggest that {Fruit} might be the focus of the knowledge base. Similarly, if the concepts {Apples, Potatoes, and Carrots} are extracted, {Fruit-vegetables} would be the output. If the set of concepts {Potatoes, Chicken, and Game} is provided, topic {Food} would be returned as the result.

3. Design of the IKB system

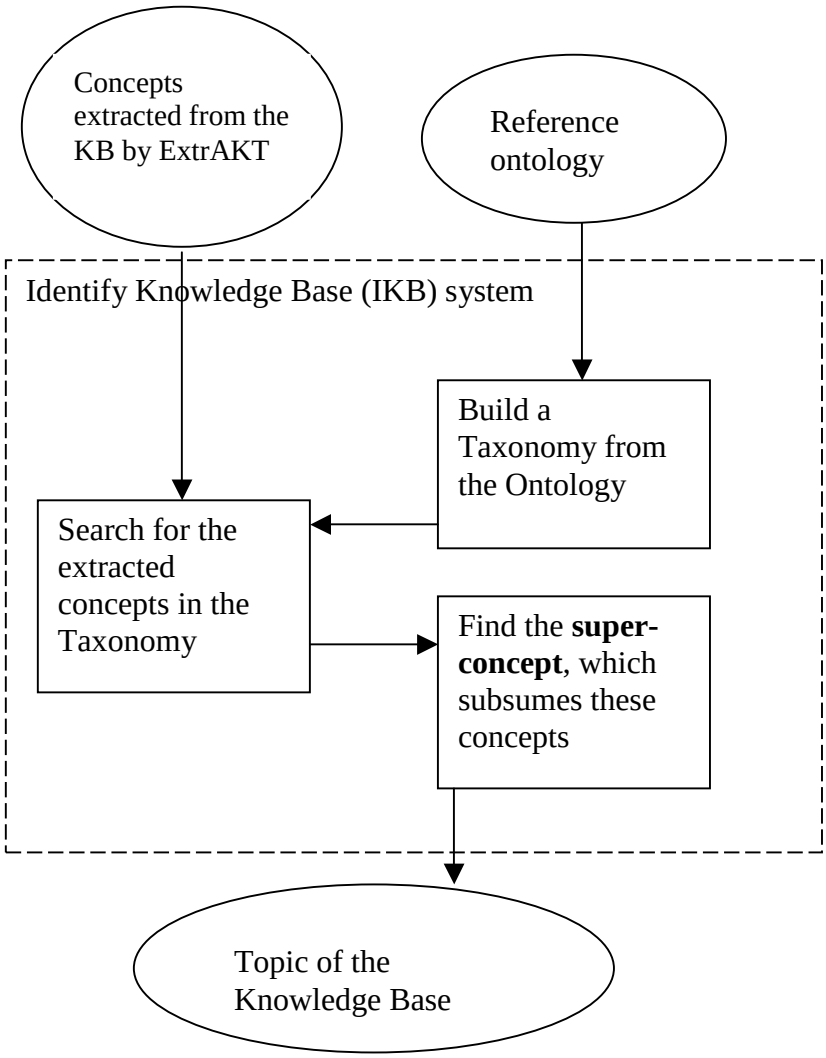

Figure-2 The main components of the IKB system[2]

As shown above, the system includes two important components:

- Taxonomy creator: it builds a taxonomy based on the reference ontology.
- Concepts searcher: it searches the taxonomy to match all the extracted concepts and finds their common parent.

[2] The rectangles in the figures represent processes while the ovals represent data or information.

4. Implementation of IKB

IKB system is implemented in the Java program. One reason for Java's selection is that it will now be easy to interface IKB to either the AKTbus [8, 9] or the Edinburgh Broker system, and thus provide a web-based service.

Jena, a Java [10] API for manipulating RDF [11] models, is used to read the ontology (RDFs file) into Java. The RDFs files can be imported from either a local disk or the Internet. After reading the reference ontology from the RDFs file, IKB transforms the ontology into the corresponding taxonomy, which is stored in a classification tree (JavaTree class). (Currently, when IKB converts an ontology into a taxonomy it is simply retained in memory, and so the taxonomy disappears between sessions, and when a new taxonomy is extracted from a further ontology.) The user can then browse the reference taxonomy, and modify the hierarchical structure as well by adding and deleting nodes. So the user can search for the principal topic of the KB with the support of potentially many taxonomies. If the ontologies stored locally on the machine, cannot meet the user requirements, the user can access further ontologies on the Internet.

The ExtrAKT system from Edinburgh is used to extract concepts from a Knowledge Base (currently, it can deal with Prolog or Clips). These concepts are then input of our IKB system. There are two different ways for the concepts to be imported to IKB system. Firstly, the concepts extracted by ExtrAKT can be stored in a text file and then read into IKB, or users can input the concepts interactively from the keyboard.

Different search strategies are available in the IKB system. One is the basic strategy that every concept has the same weight. The other one is the so-called flag strategy, which only reports concepts whose weighs are greater than or equal to the flag. Actually, the basic strategy can be considered as the special case of the flag strategy, where the flag is set to 1.

5. Testing of IKB

Next, examples of different reference taxonomies and sets of input concepts are used to test the IKB system. Two taxonomies are used here: a **FOOD** taxonomy (which is shown in section 2) and an **AUTO** taxonomy (which is about worldwide automobile manufacturers). Different input methods of the concepts and reference taxonomies (as introduced before, such as interactive input or file input, local or internet) are used in these examples. Also different search strategies are applied: the basic strategy and the flag strategy.

Three examples illustrate uses of the IKB system.

5.1 Food ontology with basic searching strategy

As shown in Figure-3, the food ontology is read from the RDFs file: *http://www.csd.abdn.ac.uk/~yzhang/food.rdfs,*

and the extracted taxonomy is subsequently stored, for further use by IKB. In this example, the concepts used are read from the file:

h:/forte/sampledir/work/predicates.txt (the extracted concepts are in fact: Pears Pears Potatoes Potatoes Potatoes Carrots Carrots Carrots Game)

IKB returns **[Food]** as it is the common parent of these concepts.

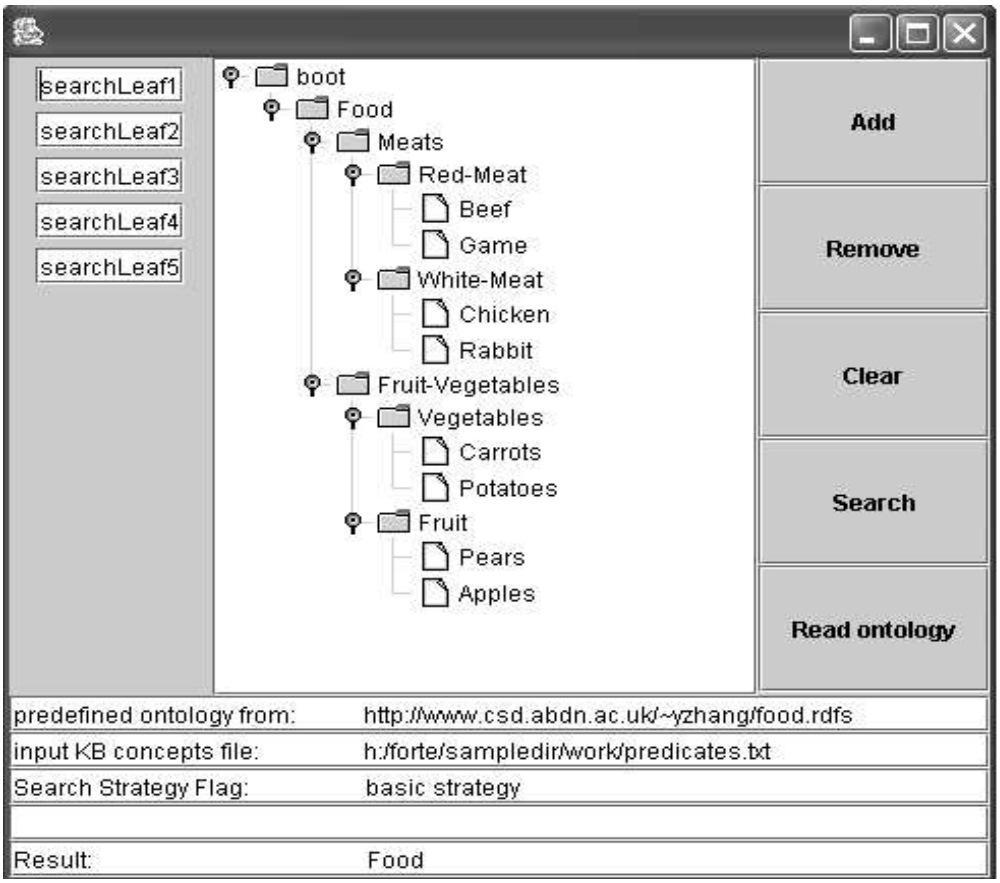

Figure-3 IKB system demonstration 1: Result is Food

5.2 Food ontology with flag searching strategy

The same reference taxonomy, Food, and the same set of concepts were used in this experiment, as were used in the first experiment. However, the search strategy was changed to flag and the value of flag was set to 2, with the result that only some of the concepts are matched, namely:

Pears (occurs twice), Potatoes (occurs 3 times) and Carrots (occurs 3 times). Thus, the search result is **[Fruit-vegetable]** this time.

5.3 Auto ontology with basic searching strategy

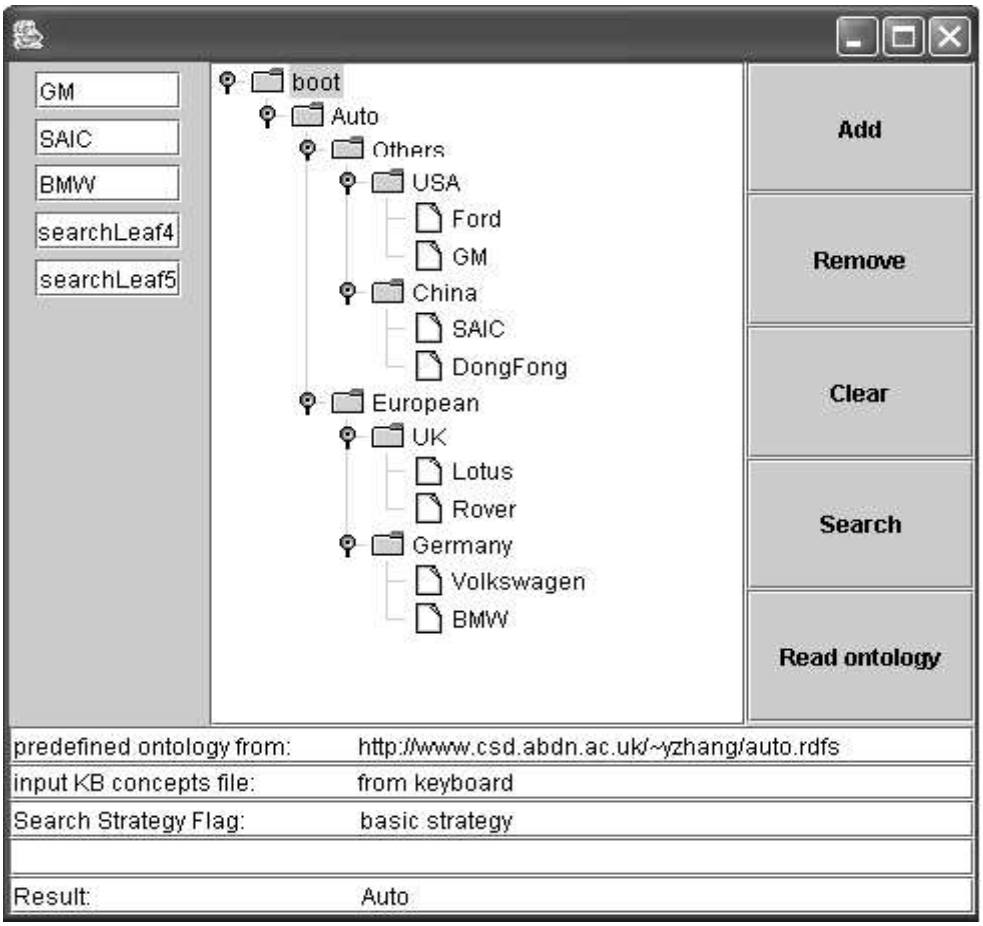

Figure-4 IKB system demmonstration3: Result is Auto

As shown in Figure-4, the ontology and the corresponding extracted taxonomy has been changed to Auto: *http://www.csd.abdn.ac.uk/~yzhang/auto.rdfs*

And this time the user types the concepts himself on the keyboard: (GM, SAIC, BMW). The basic search strategy is used again, and the system reports the highest achieved node, namely **[Auto]**.

To date, IKB has also been used with a number of ontologies, namely [12, 13, 14, 15, 16, 17].

6. Building Filters

Given the existence of IKB, we now discuss a filter which compares the topics of different KBs with the user's interest, T', so that we can find KBs required by the users. Secondly, a further filter, Filter-2, has been developed to determine whether the KB conforms to certain constraints, C (see section 1).

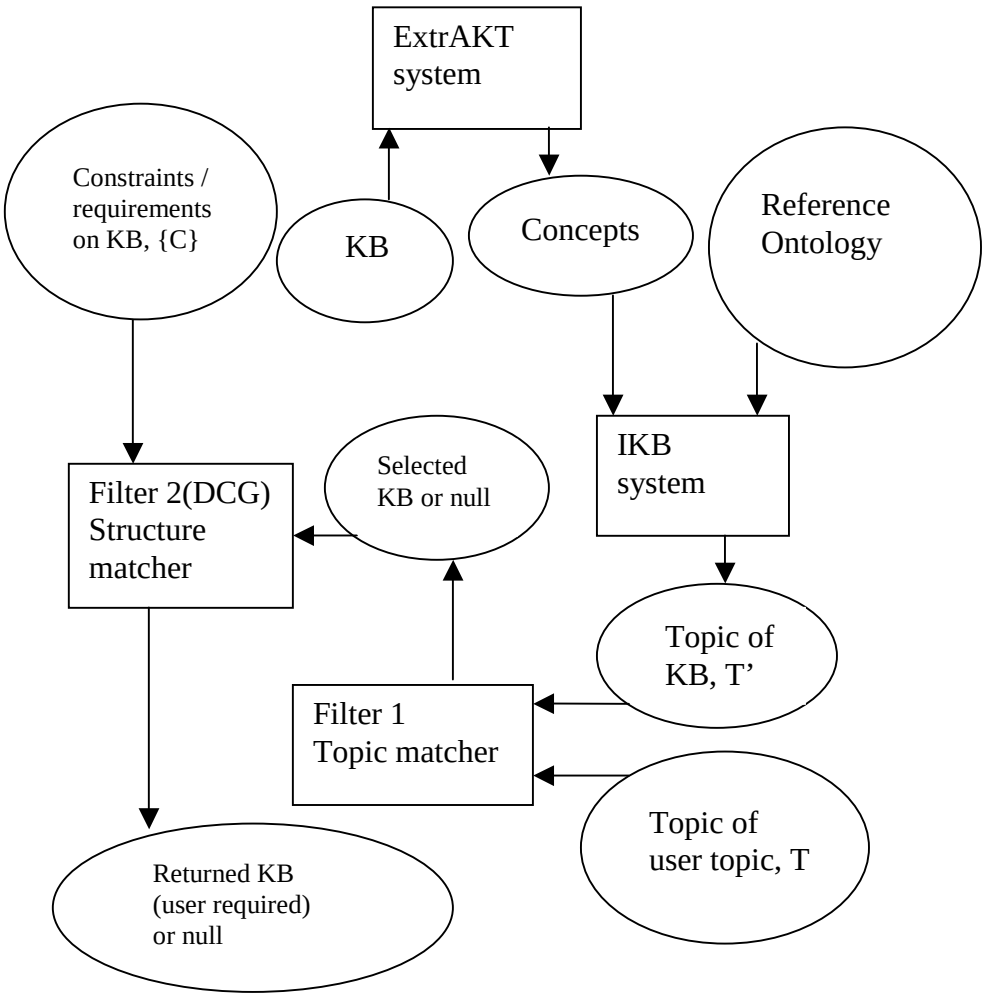

Figure-5 The components of KB Characterisation System

According to our proposed architecture, shown in Figure-5, we need two filters to find KBs, which match the user's requirements. Filter-1 is the **topic filter**. It compares the principal topic of the KB reported by the IKB system with the topic specified by the user. Filter-2 is the **structural filter**. It compares the structure of the KB passed by filter-1 with the user-defined requirements. If a KB satisfied both filters, it is reported to the user.

Filter-1 merely has to compare the required topic, T', provided by the user with the topic, T, returned by IKB for the KB. As filter-1 is trivial to implement, we focus here on the second, more demanding, filter-2.

We considered 2 approaches to checking whether a KB is consistent with the constraints specified by a user. Firstly, if the KB is associated with a header-file, which describes their contents, then it would only be necessary to verify that a header-file satisfies the user's constraints. Secondly, the user's constraints could be applied directly to the KB, which of course would generally be a more expensive process. We have in the end opted for the latter approach as we feel that there might only be a small number of KBs, which have an associated header-file; additionally there would be the problem of ensuring that header-files & KBs remain consistent throughout the KB's lifecycle.

We have employed **Definite Clause Grammar** (**DCG**) [18] to compare the user's requirements with the structure of the KB. DCG is an extension of context free grammars that has proved useful for describing natural and formal languages, and that may be conveniently expressed and executed in **Prolog**. A DCG rule in Prolog is executable because it is just a notational variant of a Prolog term that has the following general form: `Head -> Body.`

Suppose that we have a KB written in PROLOG which we know from the IKB is in the required domain, say geology/well-drilling. Further suppose that the user has specified his/her requirements as the DCG rules:

```
formation --> lithology
lithology --> ( rock, lithology-depth, lithology-length)
rock --> (rock-type, rock-hardness)
rock-type --> (shale | clay | chalk | granite| other)
rock-hardness --> (very-soft|soft|medium|hard|very-hard)
```

That is the user wants to check whether the KB contains a definition for a ***formation***, and that in turn requires the definition of a ***lithology***, which again is defined in terms of ***rock***, ***lithology-depth*** and ***lithology-length*** etc. So our task then, is to take the above constraints, express them as DCG queries, and for these to be run against the Prolog KB. In general if we wish to establish P we have to pose it as the query:

*** P ***

where * can match any structure, so that wherever in the KB, P is found, a match will be returned. (Posing the query simply as P would return a fail unless the KB contains **only P**). If the algorithm determines that the KB does not contain P, then it would be possible to relax the constraints and to see whether the KB could satisfy some of the "component" constraints. So in the above example, although the KB might not contain a definition for ***formation***, it might contain definitions for ***lithology***, ***rock***, ***rock-type***. The user then has the option to either review further KBs to find one which meets his exact requirements, or to relax his/her requirements given information about which aspects of the user requirements are not satisfied by the current KB.

This approach is likely to be computationally expensive for large KBs; if this becomes a major problem we will introduce a range of heuristics to help control the search space. Further, a more efficient and flexible approach, which supports partial specifications of the problem is being developed [19, 20].

7. Examples of Filters

Below we give examples of the use of filter-2. In effect, this filter aims to answer

*Is the definition of **Formation** in the KB consistent with the user requirements?*

To answer this question, we have to generate 6 sub-queries, each of which correspond to a component in the requirements, and the algorithm only records a positive result if all sub-queries are answered positively. These sub-queries, their translation into DCG, and the responses obtained are listed below. We shall use predicate ***match(DCG, KB, Answer)*** to check whether KB (the knowledge base represented as a list of Prolog clauses) contains DCG (the DCG rules stored as a list); and the predicate returns the result (yes or no) in the variable Answer.

```
The initial query concerns the top-level rule, which is
| ?- match([(formation --> lithology)],
           [...],
           Answer).
Answer = yes
```

That is, for a given knowledge base (shown above as "`[...]`" to save space) the system returns the ***Answer "yes"***, which means the definition of formation (if it exists) in the KB is consistent with the user's requirements.

Similarly, in the query below:

```
| ?- match([(rock -->(rock-type,rock-hardness))],
           [...],
           Answer).
Answer = yes
```

That is, the knowledge base has the definition of rock conforming to the user requirements as expressed by the DCG rule.

Another interesting example is

```
| ?- match([(rock-->(rock-type, rock-hardness)),
          (rock-type->(shale|clay|chalk|granite|other)],
          [...],
          Answer).
Answer = yes
```

In this query, we extend the previous query to include another user constraint, (rock-type).

If the KB's structure cannot match the user's requirements, the system will return "No". In this example, the user required rock-type are not found in the KB.

```
| ?- match([(rock-->(rock-type,rock-hardness)),
           (rock-type-->(hard-clay|soft-chalk|other)],
           [...],
           Answer).
Answer = no
```

8. Further developments

The IKB system is a useful tool to identify the likely topic of a given knowledge base by comparing the extracted concepts against a pre-defined reference taxonomy. The current filters have proven useful. Planned developments include:

- Extend the search strategy

There are two search strategies in our system at this moment: Flag strategy and Basic strategy (the later can be looked on as the Flag strategy with flag as 1). More complex strategy will be developed later, such as a Percentage Strategy, which matches the concepts discussed >P% of times.

- Interface to Edinburgh's Broker system [4]

One can envisaged a future broker service where the broker is asked to use a KB on a particular topic, T and with some structural constraints, C. It is envisaged that this functionality will be provided by linking IKB to the Broker possibly through the **AKTBUS,** [8, 9]. This would then, of course, allow IKB to provide a web-based service.

- More testing with different knowledge base

The usefulness of our system depends critically on the concepts extracted from the knowledge base. More knowledge bases, not only in Prolog, should be used with the ExtrAKT and the IKB systems.

- Creating a "directory" of the Taxonomies

Currently, when IKB converts an ontology into a taxonomy it is simply retained in memory, and so the taxonomy disappears between sessions, and when a new taxonomy is extracted from a further ontology. A further development will allow the user to name and retain the taxonomies on some form of persistent storage (i.e. in a file); the whole set of taxonomies will then be available to users.

- Use IKB with text files

Apply IKB on the words contained in a standard text file to find out the principal topic of that text. Nowadays, researchers in Natural Language Engineering have developed some very good tools to extract important keywords from text files, [21, 22]. We plan to apply the IKB system to these extracted keywords to see how effective the IKB is in detecting the principal focus of textual materials.

- Workbench for characterising KBs

Create a workbench to characterize the KBs, which includes the IKB system and Filters. The workbench will be able to extract taxonomies from pre-defined ontologies. At the same time, it will get concepts (or keywords) from KBs from ExtrAKT-like systems. Finally, the user requirements on the KBs will also be provided as input to the system. Based on this information, our workbench will be able to characterize the KB, both by topic and by structural constraints.

Acknowledgements

This work is supported under the EPSRC's grant number GR/N15764 to the Advanced Knowledge Technologies Interdisciplinary Research Collaboration, http://www.aktors.org/akt/, which comprises the Universities of Aberdeen, Edinburgh, Sheffield, Southampton and the Open University.

References

1. Advanced Knowledge Technology (AKT project) http://www.aktors.org/akt/
2. Sleeman D, Potter S, Robertson D, and Schorlemmer W.M. Enabling Services for Distributed Environments: Ontology Extraction and Knowledge Base Characterisation, ECAI-2002 workshop, 2002
3. Schorlemmer M, Potter S, and Robertson D. Automated Support for Composition of Transformational Components in Knowledge Engineering. Informatics Research Report EDI-INF-RR-0137, June, 2002
4. Sleeman D, Potter S, Robertson D, and Schorlemmer W.M. Ontology Extraction for Distributed Environments, In: B.Omelayenko & M.Klein (Eds), Knowledge Transformation for the Semantic Web. publ Amsterdam: IOS press, p80-91, 2003
5. ExtrAKT system: a tool for extracting ontologies from Prolog knowledge bases. http://www.aktors.org/technologies/extrakt/
6. Potter S. Broker Description, Technical Document University of Edinburgh, 03/04/2003
7. Knowledge Broker: The knowledge broker addresses the problem of knowledge service location in distributed environments. http://www.aktors.org/technologies/kbroker/
8. Hui K.Y. and Preece A. An Infrastructure for Knowledge Communication in AKT Version 1.0. Technical Report, Department of Computing Science, University of Aberdeen. 2001
9. AKT-Bus: An open, lightweight, Web standards-based communication infrastructure to support interoperability among knowledge services http://www.aktors.org/technologies/aktbus/

10. Jena - A Java API for RDF
http://www-uk.hpl.hp.com/people/bwm/rdf/jena/
11. Resource Description Framework (RDF)
http://www.w3.org/RDF/
12. Newspaper ontology
http://www.dfki.unikl.de/frodo/RDFSViz/newspaper.rdfs
13. People ontology
http://www.i-u.de/schools/eberhart/ontojava/examples/basic/demo.rdfs
14. wordnet ontology
http://www.semanticweb.org/library/wordnet/wordnet-20000620.rdfs
15. Food ontology
http://www.csd.abdn.ac.uk/~yzhang/food.rdfs
16. Auto ontology
http://www.csd.abdn.ac.uk/~yzhang/auto.rdfs
17. Academic ontology
http://www.csd.abdn.ac.uk/~qhuo/program/academic.rdfs
18. Bratko I. Prolog Programming for Artificial Intelligence, 3rd Ed, Longman. 2000
19. Vasconcelos W.W, and Meneses E.X. A Practical Approach for Logic Program Analysis and Transformation. Lecture Notes in Computer Science (Advances in Artificial Intelligence), Vol. 1793, Springer-Verlag, Berlin, 2000
20. Vasconcelos W. W, Aragao M. A. T, and Fuchs N. E. Automatic Bottom-up Analysis and Transformation of Logic Programs. Lecture Notes in Computer Science (Advances in Artificial Intelligence), Vol. 1159. Springer-Verlag. Berlin. 1996
21. Jones G.J.F. An Introduction to Probabilistic Information Retrieval Models, Department of Computer Science, University of Exeter
22. Voothees E.M. and Harman D. Overview of the Seventh Text Retrieval Conference (TREC-7), NIST Special Publication 500-242, November 1998