Ontology Winnowing: A Case Study on the AKT Reference Ontology

Harith Alani, Stephen Harris, and Ben O'Neill
School of Electronics and Computer Science
University of Southampton, Southampton, UK
{h.alani,swh,bjon104}@ecs.soton.ac.uk

Abstract

Many ontologies are built for the main purpose of representing a domain, rather than to meet the requirements of a specific application. When applications and services are deployed over an ontology, it is sometimes the case that only few parts of the ontology are queried and used. Identifying which parts of an ontology are being used could useful for realising the necessary fragments of the ontology to run the applications. Such information could be used to winnow *an ontology, i.e., simplifying or shrinking the ontology to smaller, more fit for purpose sizes. This paper presents a study on the use of the AKT Reference Ontology by a number of applications and services, and investigate the possibility of using this information to winnow that ontology.*

1 Introduction

Ontologies normally grow to large sizes when the main purpose of building them is to provide an extensive representation of a domain; such as CYC[1], GO[2], SUMO[3], FMA[4] to name just a few. However, when building ontologies for the purpose of supporting certain applications, those ontologies are expected to be much smaller in size and be more focused towards meeting the requirements of those applications and services, rather than to provide a generic representation of a domain.

When designing an ontology, it is highly recommended to keep in mind what the ontology is to be used for to avoid over or under representing the domain [7]. This is meant to assure a more fit-for-purpose scoping of the ontology. However, it is not uncommon to find ontologies that are much larger than actually required by the SW applications and services that the ontologies support. It is logical to expect enduring much higher costs for hosting and running a large and complex ontology than a trimmed-down version of that ontology. These costs include maintenance, documentation, change management, visualisation, and scalability. Most SW users only need to use small portions of existing ontologies to run their applications [8]. Tools to reduce an ontology to one that better fits certain needs can aid ontology reuse [11].

In this paper, we will examine how our project's main ontology is being used by local and external applications and services. The aim of this work is partly to get a better understanding of how and where the ontology is being queried, and more importantly, how this information can be interpreted. We hope that this study will enable us to better scope the ontology and provide some insight into whether such analysis can be used to automatically *winnow* an ontology without affecting its usage. We use the term winnowing to refer to the process of removing any unused parts of an ontology, keeping only the parts that are needed to represent the existing data and run any dependent applications.

2 Related Work

There has been some work in recent years investigating various approaches to trim ontologies for various purposes. Stuckenschmidt and Klein [10] proposed the use of classical clustering algorithms to partition ontologies based on how strongly connected the classes are to each other, regardless of any application use.

Other work suggested generating specific *views* on complex RDFS ontologies using view-querying languages [12, 6]. The aim was to use these views to personalise or simplify ontology structures by creating virtual ones, based on given view-selection queries. Related to this work is an approach suggested by Noy and Musen [8], where they propose limiting the *view* of an ontology to only a user-selected class and its neighbourhood.

Bhatt et al [2] developed a distributed architecture for extracting sub-ontologies. In this approach, users are expected to specify manually which ontology entities to keep, which to remove, and which to leave for the system to decide. In [2], if a class is selected for keeping, then they also keep (a) all its superclasses and their inherited relationships, (b) all its subclasses and their relationships, and (c) all attributes

[1] http://www.cyc.com
[2] http://www.geneontology.org/GO.biblio.shtml
[3] http://ontology.tecknowledge.com/
[4] http://sig.biostr.washington.edu/projects/fm/AboutFM.html

with cardinality more than zero. These rules are detailed in [13].

Some work have considered analysing how an ontology is being used to improve ontology management. In [9], the authors analysed ontology use to help knowledge engineers to increase the efficiency of knowledge search. They logged user queries to the knowledge base as well as the answers received, and analysed the results to find out which classes and properties have been queried the most, and which have not been queried at all. If an ontology class or property is being queried at higher rates then this might indicate a too-broadly represented class, which could be detailed further in the ontology [9]. On the other hand, if the entity is never queried, then it will be flagged as a good candidate for removal, unless the entity is instantiated in the knowledge base.

Another work that took usage into account is reported in [4]. The aim here was to monitor the use of a simple ontology (the ACM topic hierarchy) by several users, then try to make change recommendations to the items in the hierarchy. The change recommendations were based on how the hierarchy has been queried and modified by the users. However, the ACM ontology which they experimented with is a simple isA hierarchy, and the user interactions and change recommendations where equally simple.

None of the work reported in this section focussed on processing queries sent by applications and services to their supporting ontologies as an input to software to perform the trimming of these ontologies. In this paper, we investigate applying some variations of the rules proposed in previous work to winnow a locally-developed ontology that have been in use by several applications for over three years.

3 Winnowing an Ontology: A Case Study

In this section we discuss a case study on the usage of a locally maintained ontology; the AKT Reference ontology.

3.1 AKT Reference Ontology

The AKT Reference Ontology (AKTRO[5]) was developed over a period of six months by several partners of the AKT[6] project. This ontology built on a number of smaller ontologies previously developed at various AKT sites. AKTRO is written in OWL and currently consists of 175 classes and 142 properties. AKTRO models the domain of academia, and thus it contains representations for people, conferences, projects, organisations, publications, etc. AKTRO is stored in a triple store; namely 3Store [5], and is instantiated with information drawn from various databases and information gathering tools (currently stores around 30M triples in the knowledge base).

[5]http://www.aktors.org/ontology/

[6]http://www.aktors.org

The AKTRO and knowledge base is used to support a number of on-site and off-site applications. When the AKTRO was first developed over three years ago, the intention was to create a reference ontology for the whole AKT consortium to avoid the use of several variant ontologies about the same domain within the project. In other words, the aim of that ontology was to provide a reference model, rather than to meet the needs of any specific application or service.

3.2 Usage of AKT Reference Ontology

There are different types of ontology utilisation that need to be taken into account, such as instantiations, queries, direct browsing, etc. Here we are mainly concerned with the type of usage that is initiated by ontology-dependent applications and services. In the following we discuss our findings for each of these type of usages of the AKT Reference ontology.

3.2.1 Query Log

We logged the last 193 thousand RDQL queries that have been posed to the AKTRO, and data cast in it by the aforementioned applications. After analysing the logged queries, we found that only 6 classes and 27 properties of our ontology have been explicitly queried (i.e. the URIs of these classes and properties were given in at least one RDQL query). These classes and properties are given in tables 1 and 2 respectively.

Table 1. Queried classes from the AKT Reference Ontology and the number of times they appeared in the logged queries

Class	Queries	Class	Queries	Class	Queries
Technology	63462	Organization	7554	Research-Area	985
Person	750	Academic	9	Thing	3

3.2.2 Instantiations

As mentioned earlier, we maintain a knowledge base with a large number of instantiations made against the AKTRO. Even though some of these instances might not be required for running our applications, they represent an important and resourceful part of the knowledge base, and hence it was deemed important to make sure that all these instances remain intact.

For this reason, we need to include any class and property that these instances are using. Many classes in the ontology have no instances, while others are heavily instantiated. Figure 1 gives an idea on how sparsely instantiated the AKTRO is in our repository.

Table 2. Queried properties from the AKT Reference Ontology and the number of times they appeared in the logged queries

Property	Queries	Property	Queries
has-title	22478	technology-builds-on	15092
has-key-document	14964	has-author	14809
addresses-generic-area-of-interest	13735	has-appellation	12620
has-email-address	12620	has-web-address	10386
has-date	10210	has-project-leader	9549
has-project-member	9551	owned-by	7602
family-name	7588	full-name	7562
has-relevant-document	7482	works-in-unit	5140
contributes-to	3133	has-telephone-number	2832
has-pretty-name	2034	has-research-interest	1543
sub-area-of	1288	unit-of-organization	960
has-affiliation-to-unit	110	contributes-to-rating	36
has-research-quality	36	given-name	1
has-academic-degree	1		

3.2.3 Property Domains and Ranges

In addition to instantiations and queries, we also need to find all the classes that are domains or ranges of properties that were queried or used by instances (i.e assigned values for some instances). This is important to make sure the properties remain semantically intact.

Note that according to our query log, only 6 classes out of 175 have been explicitly queried (table 1). However, membership of many other classes has been indirectly constrained through properties. For example, if the property *has-project-member* appears in a query, then it is implicitly restricting the bindings for its subject to members of the class of Project and its object to the class of Person, which are the domain and range of this property respectively.

Domain and range classes that have not been been explictly instantiated are shown in grey ovals in figure 1.

3.3 Winnowing the Ontology

As stated earlier, our aim is to study how an ontology can be automatically trimmed down based on analysing how the ontology has been queried by dependent applications and services.

So to complete our experiment, we winnowed the AKT Reference ontology following the rules below:

1. Keep all ontology classes that are instantiated with one or more instances. This lead to the inclusion of 54 classes from AKTRO (figure 1).

2. Keep all the ontology properties that are assigned values by at least one instance in the knowledge base. This totaled 69 properties.

3. Keep all classes and properties explicitly mentioned in one or more queries (tables 1 and 2), that are not already found in steps 1 and 2 above. This brought in 1 additional class; *Thing*, and 3 properties; *has-academic-degree*, *has-key-document*, and *has-relevant-document*.

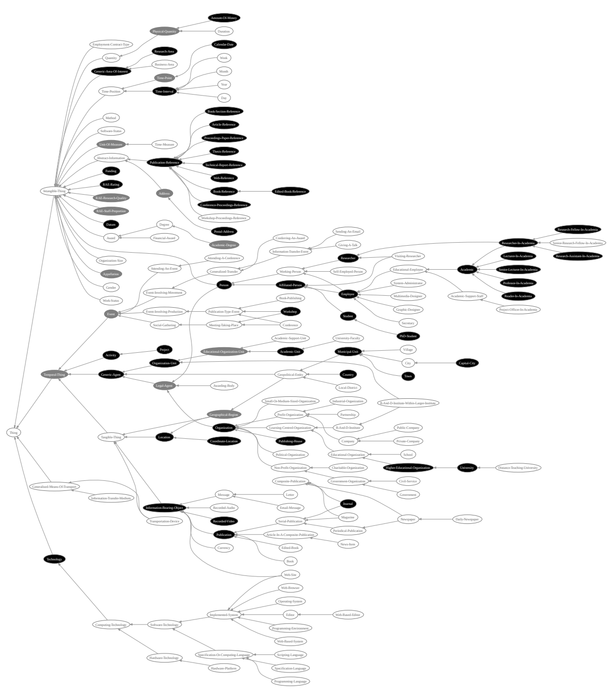

Figure 1. Space view of the AKT Reference ontology. Classes that are instantiated are shown in black. Grey classes are the domains or ranges of instantiated properties

4. Keep all classes that are domains or ranges of any property found in steps 2 or 3 above. This lead to the inclusion of 13 new classes. Note that some properties have multiple domains and ranges, not all of which are used by our applications. However, for the sake of completeness, all domains and ranges are included.

5. Remove classes and properties not identified in previous steps. Classes and properties will be shifted up the hierarchy if their superclasses are removed.

Remember that AKTO has 175 classes and 142 properties. After applying the rules above, only 68 classes (38.8% of AKTRO classes), and 72 properties (50.7% of AKTRO properties) were left (figure 2). Checking the resulting ontology (lets call it winnAKTRO) with an OWL reasoner (Pellet[7]) showed that the ontology remained consistent.

The AKTRO ontology is written in OWL, though 3store is only capable of RDFS inferencing. As it happens, none

[7]http://www.mindswap.org/2003/pellet/

of the restrictions present in the ontology were used in queries, so they were not preserved by the winnowing process. Clearly, for use with a more capable inferencing engine a more sophisticated set of rules would be required to maintain the OWL restrictions, see section 5.

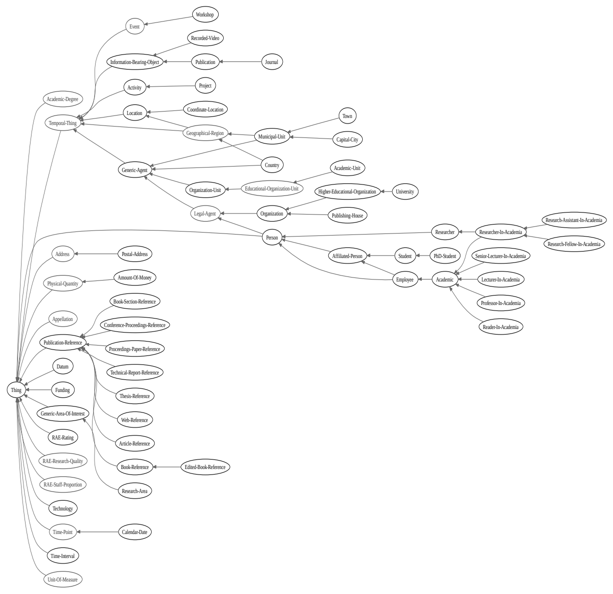

Figure 2. Space view of the winnowed AKT Reference Ontology

4 Evaluation

To evaluate the effect of winnowing AKTRO on its supported application, we compared the results of the logged queries using AKTRO against the results when using winnAKTRO.

We analysed the logged queries discussed in section 3.2 that have been sent to the original AKTRO. A list of *RDQL query templates* was distilled from the 193 thousand queries in the log. The templating process consisted of removing all literal constants and non AKTRO, OWL and RDFS schema URIs. The remaining templates can then be grouped, producing sets of queries with similar functions.

4.1 Query Subset

After collapsing identical query templates, only 928 *unique* templates remained (of which only 152 had been used twice or more). However, many of the remaining templates were only slightly different in syntax, but represented the same query. Example:

```
SELECT ?r,?l
WHERE (?s,?r,<aktro-uri>,?m),(?r,<rdf:label>,?l)
SELECT ?r,?l
WHERE (?s,?p,<aktro-uri>,?r),(?r,<rdf:label>,?l)
```

Furthermore, some queries in the log were found to be erroneous and also had to be filtered out. From the remaining templates, a set of 42 distinguished query templates where chosen for our experiment. These templates were selected carefully as representative of our query log. For each selected template, up to 50 original queries from the log were randomly chosen. The queries for this experiment totalled 1724 queries.

A copy of our AKTRO-based knowledge base was made, then AKTRO was replaced by winnAKTRO. A PHP script was written to run the selected queries on both knowledge bases and compare the results to find out how well they match. The script compares each binding returned for each query from both ontologies to determine whether the results obtained from the two ontologies are an exact match or not. The script produces a list of all affected queries (different results from both ontologies), the differing bindings, and the time it took to run each query on each of the two ontologies.

4.2 Results Comparison

The comparison of query results obtained from both ontologies revealed that:

- Out of the total of 1724 queries, 1674 were unaffected (97.1%). In other words, the results returned for these queries from winnAKTRO matched exactly the results from AKTRO. The results of 50 queries were different from winnAKTRO than from AKTRO (2.9%).
- The 50 affected queries belonged to 1 query template, out of the total 42 templates (2.4%). The number of query templates for which *none* of its queries failed is 41 (97.6% of all selected query templates).
- Running the 1724 queries on winnAKTRO took 4:02 minutes, while it took 9:13 minutes with AKTRO. In other words, there was a 44% reduction in query processing time when using the winnowed ontology.

The failed template queried the rdf:type of instances, as in:

```
   SELECT ?type
   WHERE (<instance-uri>, rdf:type, ?type)
```

This query could be issued by applications that use the type to choose how to render data relating to the instance. For example, an application might ask the above query, then search the returned types for the Project class, as a way of verifying if the instance in hand is a project or not. If the Project class is no longer in winnAKTRO, then this query will return a different answer than before. Note that according to our query log, 2080 of the queries involved rdf:type, but only 654 of them belong to the failing template above.

Other templates that include rdf:type passed the test, because the binging for rdf:type was further used in the query constraints, rather than being projected as a result. This meant that the interesting types had to either be explicitly mentioned in the query (in which case they were included in the required classes) or some property of the class was being constrained.

Ontological queries such as this one should be handled carefully to ensure that there is no significant difference in the results due to the winnowing process.

The increased speed of the execution of the queries is due to the reduced level of inference required of the query engine. 3store uses a mixture of forward– and backward–chaining [5], and the complexity of inference rules covered by either technique affects the query execution time. In the case of forward-chaining the reduction in complexity of the ontology creates fewer inferred triples, and in the case of backward chained rules the application of the rules can be optimised out in more cases, and/or the size of the intermediate candidate set can be smaller.

5 Discussion

A number of approaches have been suggested in the literature to trim down ontologies to simply make them easier to manage (sec. 2). However, we noticed that none of this work investigated using application queries as a guideline to how an ontology should be winnowed, nor they studied the effect of winnowing an ontology on its current use. Unlike user queries, application queries tend to be *fixed* at development time. Therefore, analysing how an application is making use of an ontology can be a good base for deciding how the ontology is to be winnowed. This, of course, is only possible if the applications are fully developed and their use of the ontology is not expected to change very frequently. However, it is always possible to revert to the original ontology if, for example, the requirements of the applications changed, or new applications are developed.

Some ontology-trimming rules have already been proposed (e.g. [1, 13]). However, it was apparent that **new winnowing rules are required** when applications are involved in the process. For example, the authors of [2] proposed keeping all subclasses and superclasses of preserved classes (sec. 2). In our study, this would mean keeping the entire AKTRO class hierarchy, simply because the top class, Thing, is selected for preservation. Another rule proposed in [13] is to transfer properties of unrequested classes to other classes following certain guidelines. Such modification to the ontology might be required if a class is explicitly selected for removal. However, this will certainly break many application queries to the ontology.

In [9], the authors suggested removing classes that are not explicitly instantiated or queried. According to our study, such classes should not be removed *if* they are **associated with any required property**. They also suggested that classes that are queried very often should be broken down to further subclasses. When analysing our query log, we noticed that most queries targeted more general classes, rather than their subclasses. For example, there were 750 queries to the class Person, but only *one* of its 13 *instantiated* subclasses was queried, 9 times. This matches the observation reported in [3], which states that people tend to formulate their queries more generally than actually needed. **Query frequency of classes** is therefore not a reliable indication of whether the class needs further subclassification or not.

Another possible explanation of the high use of certain classes rather than others is, as mentioned earlier, that membership of many classes has been indirectly constrained through properties, rather than explicitly mentioned in the query. For example, the property *has-project-member* appeared in 9551 queries, while the class Project (the domain of this property) has never been explicitly queried.

One crucial element in our ontology-winnowing approach is of course the **query logs**. That is, the logs of queries sent to the ontology by the applications. For the log to be *complete*, one has to make sure that all the applications to winnow the ontology for, have been running long enough to ensure that all their query templates are logged.

From the experiment described in section 4.2, the rdf:type query template **failed to return the same results** before and after winnowing the ontology. This shows that for this query template (and perhaps others yet to be discovered), it is important not only to look at the log of queries, but also to **log query answers** when deciding what to keep in the winnowed ontology and what to remove. This finding agrees with the approach taken in [9], where they analyse query results to acquire information that could potentially be used to tune the ontology.

However, the danger is that by preserving *all* the results of *all* queries, there might not be much left to remove from the ontology. The point to consider here is how acceptable it is that certain queries return different results. For some queries, it might not be important for the application to receive the same answer every time. For example, if the sole aim of a query or a set of queries is to retrieve the ontology structure to display it on the screen, then it might not be a problem if the structure has changed. However, if the aim of the query is, say, to search for all Person instances with 10 or more publications, then some consistency is more likely to be expected.

There is no easy way of finding out from the query log which query results have to be preserved (i.e. unchanged before and after winnowing), and which are more flexible. Such knowledge will most likely require some **analysis of the applications** themselves.

When winnowing an ontology, it might be important to maintain its **semantic completeness and consistency**. In our approach, our main focus was to preserve only the necessary parts of the ontology to keep the applications running, rather than to hold on to any specific semantics in the ontology. For example, our winnowing process removed some of the restrictions in AKTRO because they were not used by any application. So even though winnAKTRO remained consistent, some detailed semantics have been lost.

In our study, we have chosen to keep all instantiated classes and properties in the winnowed ontology, irrespective of whether they have been queried or not by our applications. Stojanovic and colleague [9] believe that unused instances indicates a lack of awareness of their existence. However, to minimise the ontology further, one can remove any such entities, along with their instances, if none of these instances have ever been retrieved by the applications.

6 Conclusions and Future Work

If the ultimate goal of building an ontology is to serve certain applications, then it seems sensible to use these application to trim the ontology to smaller and easier to manage sizes. In this paper we described a preliminary study we performed on the AKT Reference Ontology, which is being used by several applications. We logged over 193 thousand queries sent to the ontology from these applications, and applied some rules to winnow the ontology (i.e. to throw away any useless parts) based on the analysis of our query log. The winnowed ontology turned out to have only 38.8% of the classes, and 50.7% of the properties of the original ontology.

To examine how the applications might have been affected by the winnowing process, we selected 1724 representative queries from the log and compared their results before and after winnowing the ontology. Even though most of the queries returned identical answers (in 44% less time), the queries that asked for rdf:type without any further filtering failed, due to the removal of some classes or constraints from the ontology.

In future work we plan to experiment with logging not just the queries, but also the answers to these queries, and use that to feed into the winnowing process. We also plan to log larger amount of queries, and to compare the whole set against the winnowed ontology.

Acknowledgment

This work is supported under the Advanced Knowledge Technologies (AKT) Interdisciplinary Research Collaboration (IRC), which is sponsored by the UK Engineering and Physical Sciences Research Council under grant number GR/N15764/01. The AKT IRC comprises the Universities of Aberdeen, Edinburgh, Sheffield, Southampton and the Open University. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing official policies or endorsements, either express or implied, of the EPSRC or any other member of the AKT IRC.

References

[1] M. Bhatt, A. Flahive, C. Wouters, W. Rahayu, D. Taniar, and T. Dillon. A distributed approach to sub-ontology extraction. In *18th Int. Conf. on Advanced Information Networking and Applications (AINA)*, pages 636–641, Fukuoka, Japan, 2004.

[2] M. Bhatt, C. Wouters, A. Flahive, W. Rahayu, and D. Taniar. Semantic completeness in sub-ontology extraction using distributed methods. In *Proc. Int. Conf. on Computational Science and its Applications (ICCSA)*, pages 508–517, Perugia, Italy, 2004. LNCS, Springer Verlag.

[3] H. Chen and V. Dhar. Cognitive process as a basis for intelligent retrieval systems design. *Information Processing & Management*, 27(5):405–432, 1991.

[4] P. Haase, A. Hotho, L. Schmidt-Thieme, and Y. Sure. Collaborative and usage-driven evolution of personal ontologies. In *Proc. Second European Semantic Web Conference (ESWC)*, pages 486–499, Crete, 2005.

[5] S. Harris and N. Gibbins. 3store: Efficient bulk rdf storage. In *Proc. 1st Int. Workshop on Practical and Scalable Semantic Systems (PSSS'03)*, pages 1–20, Sanibel Island, FL, USA, 2003.

[6] A. Magkanaraki, V. Tannen, V. Christophides, and D. Plexousakis. Viewing the semantic web through rvl lenses. In *Proc. Second Int. Semantic Web Conf. (ISWC)*, pages 98–112, Sanibel Island, Florida, 2003.

[7] N. F. Noy and D. L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical Report KSL-01-05, Stanford Medical Informatics, March 2001.

[8] N. F. Noy and M. A. Musen. Specifying ontology views by traversal. In *3rd Int. Semantic Web Conf. (ISWC'04)*, Hiroshima, Japan, 2004.

[9] N. Stojanovic and L. Stojanovic. Usage-oriented evolution of ontology-based knowledge management systems. In *Int. Conf. on Ontologies, Databases and Applications of Semantics (ODBASE)*, pages 230–242, Irvine, CA, 2002.

[10] H. Stuckenschmidt and M. Klein. Structure-based partitioning of large concept hierarchies. In *3rd Int. Semantic Web Conf. (ISWC2004)*, Hiroshima, Japan, 2004.

[11] M. Uschold, P. Clark, M. Healy, K. Williamson, and S. Woods. An experiment in ontology reuse. In *Proc. Eleventh Knowledge Acquisition Workshop (KAW)*, Banff, Canada, 1998.

[12] R. Volz, D. Oberle, and R. Studer. Implementing views for light-weight web ontologies. In *Proc. IEEE Database Engineering and Application Symposium (IDEAS)*, Hong Kong, China, 2003.

[13] C. Wouters, T. Dillon, W. Rahayu, and E. Chang. A practical walkthrough of the ontology derivation rules. In *Proc. 13th Int. Conf. on Database and Expert Systems Applications (DEXA)*, pages 259–268, Aix-en-Provence, France, 2002.