Winnowing Ontologies Based on Application Use

Harith Alani, Stephen Harris, and Ben O'Neil

Advanced Knowledge Technologies (AKT), School of Electronics and Computer Science,
University of Southampton, Southampton SO17 1BJ, UK
{h.alani, swh, bjon104}@ecs.soton.ac.uk

Abstract. The requirements of specific applications and services are often over estimated when ontologies are reused or built. This sometimes results in many ontologies being too large for their intended purposes. It is not uncommon that when applications and services are deployed over an ontology, only a few parts of the ontology are queried and used. Identifying which parts of an ontology are being used could be helpful to *winnow* the ontology, i.e., simplify or shrink the ontology to smaller, more fit for purpose size. Some approaches to handle this problem have already been suggested in the literature. However, none of that work showed how ontology-based applications can be used in the ontology-resizing process, or how they might be affected by it. This paper presents a study on the use of the AKT Reference Ontology by a number of applications and services, and investigates the possibility of relying on this *usage* 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. However, when building or reusing 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. A study of the applications submitted to the 2003 Semantic Web (SW, [4]) challenge, showed that most of the ontologies used in those applications were relatively small and simple, but also sufficient for their intended purposes [12].

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 [14]. The first step of the methodology proposed by Uschold and Grüninger for building ontologies is to identify its purpose and scope [22]. Grüninger and Fox [8] suggested articulating the requirements for an ontology in the form of a list of *competency questions* that the ontology must be able to answer. This is meant to assure a more fit-for-purpose scoping of the ontology.

However, in spite of such recommendations and methodologies, it is not uncommon to find ontologies that are much larger than actually required by the SW applications and services that the ontologies are meant to support. Most SW users only need to use small portions of existing ontologies to run their applications [15]. Ontology engineers might sometimes prefer to build or reuse extensive and more detailed ontologies for their applications in preparation for a probable future expansion of requirements, or to allow for an imaginable automatic communication with external agents that perhaps will require a more exhaustive representation of the domain.

Nevertheless, 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. This includes costs of maintenance, documentation, change management, visualisation, and scalability. Tools to reduce an ontology to one that better fits certain needs can also greatly aid and encourage reusing existing ontologies [21].

Several approaches and suggestions for semi-automatic trimming or breaking-down of ontologies into smaller sizes have been introduced in the literature. However, to the best of our knowledge, none of the proposed approaches are driven by application needs, or study how applications might be affected by the shrinkage of their supporting ontologies. Manually selecting which parts or classes of an ontology to preserve, which is what most of previous work is based on, can be unreliable. The developer of the ontology or application might not be fully aware of *all* the parts that will be *required* and *used* by the application, and the ones that will never be needed or used. So in the end, the application itself might be the best judge here.

In this paper, we would like to examine how the usage of an ontology by applications can be used to drive the process of reducing the ontology size. To this end, we will study 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. These approaches vary in purpose and technique, as described below.

2.1 Ontology Partitioning

Stuckenschmidt and Klein [20] proposed the use of classical clustering algorithms to partition ontologies based on how their class hierarchies are structured. In their approach, each partition will contain classes that are more strongly connected to other classes in this partition rather than to classes in other partitions. The suggested ontology partitioning is therefore based on how the classes are connected in the ontology, regardless of how the ontology is, or can be, used. Nothing in this approach can guarantee that the ontology produced by each cluster is meaningful or usable.

If an application needs to interact with a set of classes that ended up being spread over many of the partitions suggested by the clustering process, then those partitions will have to be remapped or remerged together, thus rendering the whole partitioning process less useful. So even though the approach suggested in [20] is good for a general structure-based scaling down of ontologies, it is not suitable for usage-driven ontology winnowing.

2.2 Ontology Views

Other work suggested generating specific *views* on complex RDFS ontologies using view-querying languages [23, 13]. The aim was to use these views to personalise or simplify ontology structures by creating virtual ones, based on view-selection queries. Another related approach is to limit the *view* of an ontology to only a user-selected class and its neighbourhood [15]. This neighbourhood can be restricted in size by the number of connections it is allowed to span out, which can be set by the user for each property separately.

The above approaches can be useful for quickly limiting how an ontology is viewed or browsed. However, they are not designed, not well suited, for automatic extraction of ontology parts, or for trimming an ontology based on the demands of certain applications and services.

2.3 Ontology Segment Extraction

Bhatt and colleagues [5] developed a distributed architecture for extracting sub-ontologies. In this approach, users were expected to specify which ontology entities to keep, which to remove, and which to leave for the system to decide. This approach suggests using three main rules to minimise ontologies. For a class that is selected to be kept:

- Keep all its superclasses and their inherited relationships.
- Keep all its subclasses and their relationships.
- Keep all attributes with cardinality more than zero.

The experiments reported in [5] were driven by users manually selecting which entities to keep and which ones to throw away. It is not obvious how this approach could apply when the selection process is based on applications actively using the ontology to be winnowed, and if and how such applications might be affected by the process.

Another segmentation work is presented in [16]. They start from a specific class and follow certain paths along the ontology network to create a segment. They have only applied their approach to the GALEN ontology, and only considered segmenting based on a single class selection, rather than with multiple classes as would be the case when based on ontology use. The rules they applied are:

- Keep all superlasses and subclasses of selected class
- Include equivalent classes
- Include properties and classes of any restrictions on included classes.
- Keep all the classes that these properties point to
- Keep superproperties and superclasses of all the included classes from the last 3 steps.

This approach focuses on maintaining the semantics of the ontology, which is why it is more generous in its segmentation that the previous approaches.

2.4 Ontology Use Analysis

Analysing how an ontology is being used is an important step towards better ontology management [18]. In [19], the authors logged ontology usage information in the aim that this might help knowledge engineers to increase the efficiency of search within a given knowledge base (KB). They argued that if an ontology class or property is queried at higher rates then this might indicate a too-broad representation, which could be detailed further in the ontology [19]. 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 KB. They also looked at the problem where an entity is frequently queried but not many results are available. They regard this as an indication of a knowledge gap.

Note that the work reported in [18] and [19] only looked at direct user interactions with a KB, rather than at queries steadily coming from external applications. Furthermore, how the ontology is to be changed in line with the information they collected, and how that change will affect further use of the KB, seemed to be out of the scope of their reported work.

Another work that took usage into account is reported in [9]. 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 on 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 taxonomy, and the user interactions and change recommendations were 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 the AKT Reference Ontology

Ontology winnowing differs from the approach described in section 2 in that the process of trimming the ontology here is entirely based on *need*. Need is determined by the queries sent to the ontology by any supported applications, and by the underlying data.

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[1]) was developed over a period of six months by several partners of the AKT[2] project. This ontology built on a number of smaller ontologies previously developed at various AKT sites. AKTRO currently consists of 175 classes and 142 properties.

AKTRO models the domain of academia. It contains representations for people, conferences, projects, organisations, publications, etc. AKTRO is stored in a triple store;

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

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

namely 3Store [11], and is instantiated with information drawn from various databases and information gathering tools (currently stores around 30M triples in the KB). The AKTRO ontology is written in OWL, though 3Store is only capable of RDFS inferencing, and thus AKTRO was stored in 3Store in RDFS.

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 AKTRO Instantiations

As mentioned above, we maintain a KB with a large number of instantiations made against the AKTRO. Many classes in the ontology have no instances, while others are heavily instantiated. Figure 1 gives some idea on how **sparsely** instantiated the AKTRO is in our repository.

Even though some of these instances might not be required for running some of our applications, they represent an important and resourceful part of the KB and can be considered as a type of ontology use, and hence it was deemed important to make sure that all these instances remain intact.

3.3 Queries to AKTRO

The AKTRO and its KB are used to support a number of on-site and off-site applications, such as OntoCoPI[3], CSAktiveSpace[17], AKT Technologies dynamic web pages[3], Armadillo[7] from Sheffield University, and any other ad hoc works, such as [1], or even queries directly typed by users.

In our case study, we experimented with winnowing AKTRO based on its general use by the above applications and services. Then in a second experiment, we focused the usage analyses of AKTRO on two selected applications only, and extended the winnowing process to take into account query results. The two experiments are described in the following.

4 Experiment 1: Winnowing AKTRO Based on General Use Analysis

Our first winnowing experiment was based on the general use of AKTRO, where all queries from any application or ad hoc query were taken into account [2]. This section details this experiment and its results.

4.1 Query Log

We logged over 193 thousand RDQL queries that have been posed to the AKTRO by various sources. 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

[3] http://www.aktors.org/technologies/

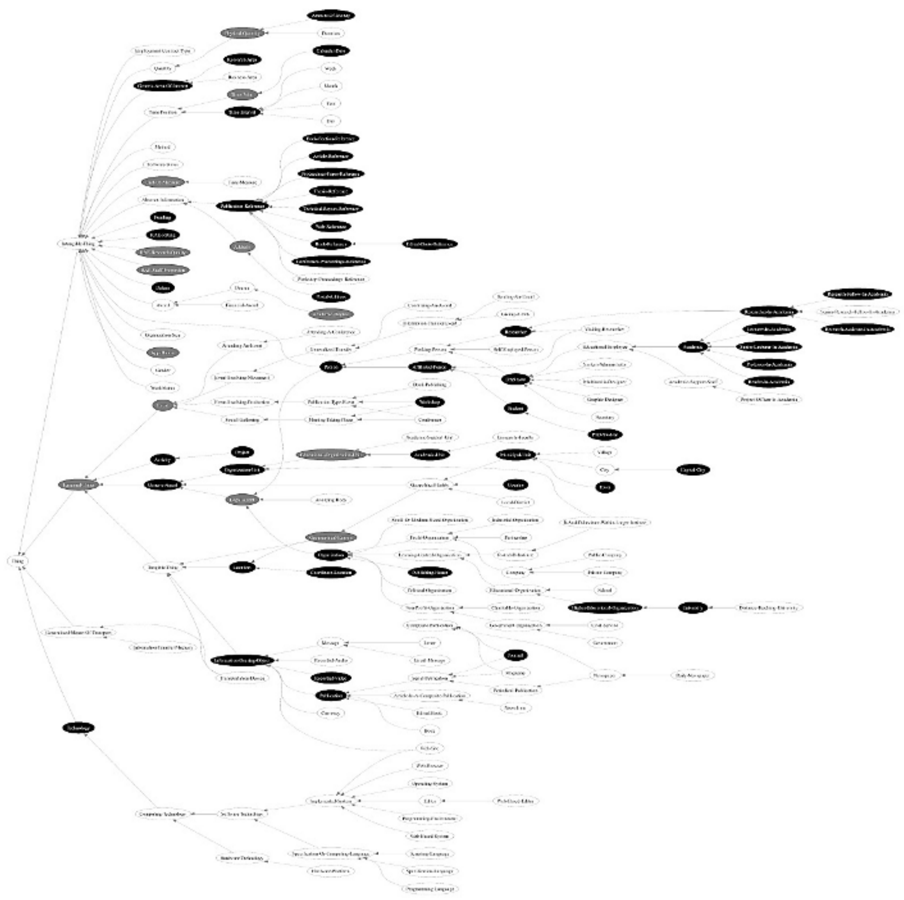

Fig. 1. Space view of the AKT Reference ontology. Classes that are instantiated are shown in black colour. Grey coloured classes are the domains or ranges of instantiated properties.

and properties were given in at least one RDQL query). These classes and properties are given in tables 1 and 2 respectively.

The frequency with which a concept or a property has been queried gives a good indication of whether the query is directly typed in by a person (very low frequency), or rather coming from an application (higher frequencies). However, to make sure that all requests are met, any concept or property that has been queried at least once will be regarded as essential, and included in the winnowed ontology. In the second experiment, only application queries are considered (section 5.3).

Membership of many other classes can be 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. This indicates that in addition to instantiations and queries, we also need to find all the

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

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		

classes that are domains or ranges of properties that were queried or used by instances (i.e assigned values for some instances).

4.2 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 is being used by dependent applications and services. So to complete our experiment, we winnowed the AKT Reference ontology by following these rules:

1. Keep all ontology classes that are directly 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 KB. This totalled 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*.
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 had 175 classes and 142 properties. After applying the rules above, only 68 classes (61.2% reduction), and 72 properties (49.3% reduction) were left. Checking the resulting ontology (lets call it winnAKTRO-1) with a reasoner (Pellet[4]) showed that, in this particular case, the ontology remained consistent.

4.3 Evaluation of winnAKTRO-1

To evaluate the effect of winnowing AKTRO on its supported applications, we compared the results of nearly 1800 carefully selected logged queries using AKTRO against the results when using winnAKTRO-1 [2]. Scripts were used to compare 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 comparison revealed that the results of around 3% of the selected queries were different when using winnAKTRO-1 than when using AKTRO. A closer look at the results showed that the failed queries were querying 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 among the returned types for the Project class, as a way of verifying whether the instance in hand is a project or not. If the Project class is no longer in winnAKTRO-1, then this query will return a different answer than before.

5 Experiment 2: Taking Query Results into Account

The previous winnowing trial showed that, for some RDQL queries, the results were different before and after the winnowing process. This indicates that further steps might be required when winnowing an ontology to avoid such result-mismatches.

We decided to focus this second experiment on two specific applications (CSAktive Space [17] and OntoCoPI [3]) to make sure that the queries in the log can be traced back to their sources. This will also ensure that no arbitrary queries (not required by any application) are logged by mistake. Note that these two applications are the most active users of AKTRO.

In this experiment, we will also take the *results of queries* in the log into account when winnowing the ontology. This will hopefully reduce the results-mismatching problem encountered in experiment 1 (section 4.3).

5.1 Query Log

To make sure we log all possible queries from the two selected applications, the applications were put to extensive use for several hours, and their queries to 3Store were clearly tagged in the log. The result was a query log of just under 13 thousand queries.

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

Table 3. Properties in AKTRO that are queried explicitly by CSAktive Space and Ontocopi

Properties	Queries	Properties	Queries	Properties	Queries
has-project-leader	1011	has-project-member	1011	has-date	854
has-author	828	works-in-unit	528	address-generic-area-of-interest	506
has-grant-value	253	has-amount	253	has-funding	253
has-telephone-number	157	has-web-address	157	unit-of-organization	123
has-pretty-name	80	has-research-interest	63	sub-area-of	41
contributes-to-rating	22	has-research-quality	22		

After collapsing duplicates, 5 thousand unique queries remained. Number of queries is much less than in previous experiment because of the focus on two applications only.

When analysing the query log it was found that only one class and 17 properties have been explicitly queried. The queried properties and the number of queries that mentioned them is shown in table 3. The queried class was *Person*, and it appeared in 137 queries.

The fact that only one class was explicitly queried supports the observation from the first experiment that applications often rely on properties to filter out the results. OntoCoPI for example often queries the triple store for any individuals connected to a given person instance via specific properties [3]. Such a query will return instances of classes such as Project, Conference, Paper, etc. OntoCoPI then asks for the rdf:type of each returned URI to find their class types.

5.2 Query Results

As mentioned earlier, in this second experiment, the results of the logged queries will also be considered when identifying the class and property URIs to be maintained in the winnowed ontology.

Table 4. Classes in AKTRO that appear in results of queries from CSAktiveSpace and OntoCoPI

Class	Results	Class	Results	Class	Results
Thing	504	Intangible-Thing	424	Publication-Reference	275
Abstract-Information	275	Person	218	Generic-Agent	189
Temporal-Thing	154	Legal-Agent	149	Proceedings-Paper-Reference	135
Affiliated-Person	120	Article-Reference	114	Employee	71
Academic	47	Researcher-In-Academia	29	Working-Person	13
Educational-Employee	13	Technology	9	Book-Section-Reference	7
Book-Reference	6	Researcher	6	PhD-Student	4
Conference-Proceedings-Reference	4	Student	4	Activity	3
Technical-Report-Reference	3	Project	2	Prof	2
Dr	2	Thesis-Reference	1	Professor-In-Academia	1

Table 5. Properties in AKTRO that appear in results of queries from CSAktiveSpace and OntoCoPI

Properties	Results	Properties	Results	Properties	Results
has-research-interest	781	works-in-unit	724	sub-area-of	305
works-for	168	unit-of-organization	152	has-affiliation-to-unit	128
contributes-to-rating	83	family-name	61	full-name	61
studies-in-unit	54	has-email-address	32	has-appellation	26
has-telephone-number	25	has-fax-number	20	has-affiliation	19
has-postal-address	16	project-involves-organization-unit	11	given-name	9
has-web-address	7	sub-unit-or-organization-unit	6	has-pretty-name	6

When analysing the results of the 13 thousand logged queries, the URIs of 30 classes and 21 properties were found (tables 4 and 5 respectively).

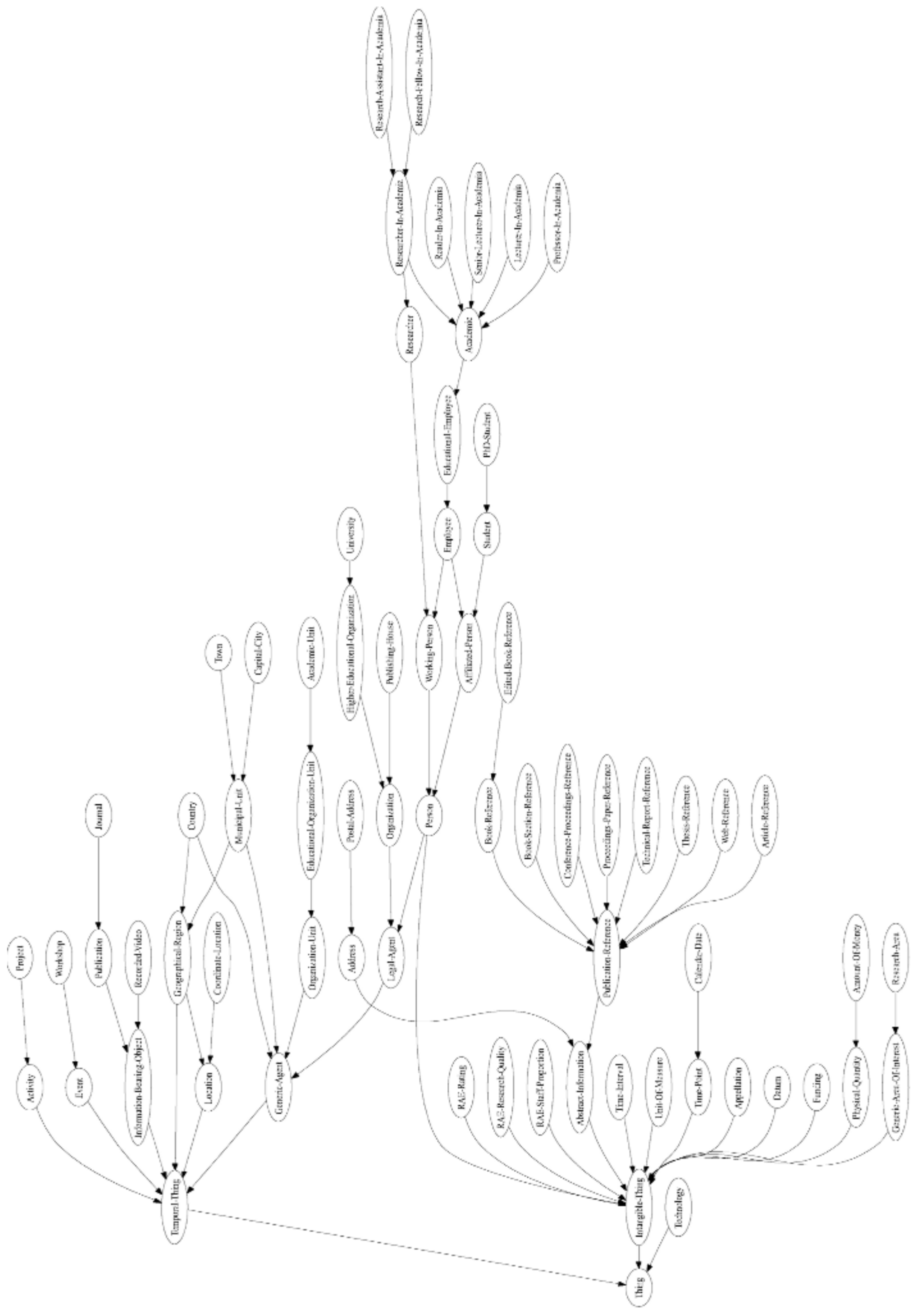

Fig. 2. Space view of the winnowed AKT Reference Ontology for experiment 2

5.3 Winnowing the Ontology

Similarly to section 4.2, the following steps were followed to winnow the ontology:

1. Keep all directly instantiated classes and properties (assigned values for some instances). As in experiment 1, 54 classes and 69 properties remained.
2. Keep all classes and properties that were explicitly mentioned in queries. Even though 1 class and 17 properties were explicitly mentioned in the logged queries, they were all instantiated and thus already identified in step 1.
3. Keep all domains and ranges of required properties. 10 new classes were added that were not identified in previous steps.
4. Keep all classes and properties that appear in the results of the logged queries. This includes 30 classes and 21 properties. However, only 7 classes and 0 properties have not been already identified in the previous steps.
5. Remove classes and properties not identified in previous steps. Classes and properties will be shifted up the hierarchy if their superclasses are removed.

Once the rules above were applied to AKTRO and winAKTRO-2 was produced, it had 71 classes, and 69 properties. This is a reduction of 59.5% in classes, and 51.4% in properties when compared to the original ontology (figure 2).

5.4 Evaluation of winnAKTRO-2

The results of all the queries logged in experiment 2 (5K queries) were compared as returned from AKTRO and winnAKTRO-2. The comparison showed a perfect match between the two sets of results, including queries on rdf:type which failed in experiment 1. In other words, the applications used in this experiment have not been affected by the winnowing process, and continued to function as usual. However, the ontology that supports them is now about half of its original size.

6 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 did they study the effect of winnowing an ontology on its current use. Unlike user queries, **application queries tend to be fixed** to some extent at development time. Therefore, analysing how an application is making use of an ontology can form a good basis 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 (sec. 2). However, we believe that such **rules need to be rechecked and perhaps changed** when applications are involved in the process. For example, some have proposed keeping all subclasses and superclasses of preserved classes ([5, 16]). In our first experiment (sec. 4, this would have meant keeping the entire AKTRO class hierarchy, simply because the top class, *Thing*, was selected for preservation.

In [19], the authors 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 the more general classes, rather than their subclasses. For example, in experiment 1, there were 750 queries to the class *Person*, but only *one* of its 13 *instantiated* subclasses was queried, 9 times. In experiment 2, none of Person's subclasses were queried, whereas Person appeared in 137 queries. Of course this is somewhat dependent on the applications, and on the type of query filtering used (see below). Nevertheless this observation seems to match the report in [6], which states that people tend to formulate their queries more generally than actually needed. **Query frequency of classes** is therefore not always a reliable indication of whether a 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 queries. For example, the property *has-project-member* appeared in 9551 queries in experiment 1, and in 1011 queries in experiment 2, while the class *Project* (the domain of this property) has not been explicitly queried in either experiment.

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 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.

In our study, we kept all directly instantiated classes and properties in the winnowed ontology, irrespective of whether they have been queried or not by applications. Stojanovic and colleague [19] believe that unused instances indicates a lack of awareness of their existence. However, we believe that in some cases, non-queried instances are simply not needed by the applications. To minimise the ontology further, one can remove any such classes (if none of their instances are ever queried), along with their instances.

In experiment 1, some queries that involved rdf:type **failed to return the same results** before and after winnowing the ontology (sec. 4.3). This shows that for this query template, and perhaps others, 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 throw away, or shelf!. This finding agrees with the approach taken in [19], where they analysed query results to acquire information that could potentially be used to tune the ontology. Taking query results into account when winnowing AKTRO in experiment 2 lead to a 100% match of query results to AKTRO and winnAKTRO-2.

However, a point to consider when reserving all classes and properties that appear in query results is how acceptable it is for **certain queries to return different results**. For some queries, it might not be important for the application to receive the exact 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 publications of John Smith prior to 2001, then some consistency is 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 practical oriented approach, the 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. For example, our winnowing process removed some restrictions in AKTRO because they were not used by any application.

7 Conclusions and Future Work

If the ultimate goal of reusing or building an ontology is to serve specific applications, then it seems sensible to use these application to limit the ontology to smaller and easier to manage sizes. In this paper we described a study we performed on the AKT Reference Ontology which is being used by several applications. We logged large number of queries sent to the ontology from several applications, and applied some rules to winnow the ontology and throw away or shelf any unnecessary parts, regardless of their position in the original ontologies. The winnowed ontology produced in the first experiment turned out to have only 38.8% of the classes, and 50.7% of the properties of the original ontology. Query results were taken into account when producing the second winnowed ontology, which had 59.5% less classes and 51.4% less properties than the original AKTRO.

We have shown through experiments that analysing query syntax to determine which parts of an ontology are being triggered is not enough without also analysing the *results* of those queries. Further developments to this work could include the processing of SPARQL queries, rather than RDQL. This has the advantage that the SPARQL language and protocol are both more tightly defined than RDQL, making the technique easier to apply to other platforms and applications.

We expect our winnowing approach to produce scruffier ontologies, with perhaps less semantic consistency than their originals. Further steps will be needed to maintain consistency while changing the ontology [10]. More of the ontology will need to be preserved if higher semantic consistency is required (e.g. if all constraints must remain in the winnowed ontology). However, this might not be required if the main goal is to simply shrink an ontology with respect to the exact needs of specific applications, without affecting any of their queries. If the applications' needs change, or the knowledge base changes, then the winnowing process should be rerun on the original ontology to produce a new winnowed ontology.

In addition to semantic consistency, we need to pay attention to semantic redundancy that might result from the winnowing process. For example in winnAKTRO-2, *Geographical-Region* is now a subclasses of *Location* as well as of *Temporal-Thing*, and *Location* itself is a subclass of *Temporal-Thing*. A classifier could be used to identify and sort out such cases.

The methodology used in this paper takes into account classes and properties that are explicitly mentioned in queries and/or results, but does not take into account those that are potentially included in the subgraph used by the query engine but not explicitly mentioned. For example, the query:

```
SELECT ?i WHERE (?i rdf:type ?class)
      (?class rdfs:label "WorkingPerson")
```

uses the class WorkingPerson during the query execution without ever mentioning it explicitly, and without it appearing in the results. As it happens no classes or properties are used in only this way by the applications in the study, but it is a possibility that we will investigate in future work. Another related issue is that it some cases (e.g. for clarification or mapping purposes) the ontology is required to contain certain classes or properties without them being instantiated or queried. One possible solution to force our winnowing approach to maintain such entities is for applications to add dummy queries to any classes or properties that they desire to keep in the winnowed ontology.

As mentioned earlier, 3Store only performs RDFS inferencing. AKTRO was stored in RDFS in 3Store. As it happens, none of the restrictions present in the ontology were used in the logged queries, so they were not preserved by the winnowing process. Clearly, for use with an OWL inferencing engine a more sophisticated set of rules would be required to maintain the OWL restrictions.

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. H. Alani, N. Gibbins, H. Glaser, S. Harris, and N. Shadbolt. Monitoring research collaborations using semantic web technologies. In *Proc. 2nd European Semantic Web Conf. (ESWC)*, pages 664–678, Crete, 2005.
2. H. Alani, S. Harris, and B. O'Neil. Ontology winnowing: A case study on the akt reference ontology. In *Proc. Int. Conf. on Intelligent Agents, Web Technology and Internet Commerce (IAWTIC'2005)*, Vienna, Austria, 2005. IEEE.
3. H. Alani, S. D. K. O'Hara, and N. Shadbolt. Identifying communities of practice through ontology network analysis. *IEEE Intelligent Systems*, 18(2):18–25, 2003.
4. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, May 2001.
5. 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.

6. H. Chen and V. Dhar. Cognitive process as a basis for intelligent retrieval systems design. *Information Processing & Management*, 27(5):405–432, 1991.
7. F. Ciravegna, S. Chapman, A. Dingli, and Y. Wilks. Learning to harvest information for the semantic web. In *Proc. 1st European Semantic Web Symp. (ESWS)*, Crete, Greece, 2004.
8. M. Gruninger and M. S. Fox. Methodology for the design and evaluation of ontologies. In *Proc. Workshop on Basic Ontological Issues in Knowledge Sharing, in IJCAI'95*, Montreal, Canada, 1995.
9. 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.
10. P. Haase, F. van Harmelen, Z. Huang, H. Stuckenschmidt, and Y. Sure. A framework for handling inconsistency in changing ontologies. In *Proc. 4th Int. Semantic Web Conf. (ISWC)*, Galway, Ireland, 2005.
11. 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, FL, USA, 2003.
12. M. Klein and U. Visse. Semantic web challenge 2003. *IEEE Intelligent Systems*, 19(3):31–33, 2004.
13. 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.
14. 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.
15. N. F. Noy and M. A. Musen. Specifying ontology views by traversal. In *3rd Int. Semantic Web Conf. (ISWC'04)*, Hiroshima, Japan, 2004.
16. J. Seidenberg and A. Rector. Web Ontology Segmentation: Analysis, Classification and Use. In *Proceedings 15th International World Wide Web Conference*, Edinburgh, Scotland, 2006.
17. N. Shadbolt, monica schraefel, N. Gibbins, and S. Harris. CS Aktive Space: or how we stopped worrying and learned to love the semantic web. In *2nd Int. Semantic Web Conf*, Florida, 2003.
18. N. Stojanovic, J. Hartmann, and J. Gonzalez. Ontomanager - a system for usage-based ontology management. In *Proc. FGML Workshop. SIG of Germal Information Society*, 2003.
19. 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.
20. H. Stuckenschmidt and M. Klein. Structure-based partitioning of large concept hierarchies. In *3rd Int. Semantic Web Conf. (ISWC2004)*, Hiroshima, Japan, 2004.
21. 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.
22. M. Uschold and M. Gruninger. Ontologies: principles, methods and applications. *The Knowledge Engineering Review*, 11(2):93–136, 1996.
23. 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.